SDN-based Malicious Traffic Defense Solution

Chien Wei Lan, Fang Lu, Chen-Hsuan Hsu, Prashanth Rajasekar

Computer Engineering
San Jose State University
San Jose, CA 95192

Electrical Engineering
San Jose State University
San Jose, CA 95192

***Abstract*—Software-Defined Networking (SDN), a hot topic of internetworking, is emerging in a newly-developed network. It purposed the separation between the control plane and the data plane. It dramatically improves the usage of the network utilization, simplifies the management for the administrator, and allows the network engineers to easily program the network. Malicious traffics, on the other hand are trying to slow down the environment, leveraging a large amount of traffic flow to cost the entire system down. In this paper, we investigate how SDN can be used to block the malicious traffic. This firewall takes action when a malicious traffic attacks a host. In order to block the traffic, Floodlight controller needs to implement a module to record the ICMP, TCP and UDP packet.**

Keywords—Software-Defined Networking, Firewall, Floodlight, OpenFlow, sFlow-rt, Distributed Denial of Service (DDoS)

I. INTRODUCTION

Software-Defined Networking (SDN), a newly-emerging technique fundamentally changed the architecture of the traditional IP. The control plane (which is the brain of making a decision about how to process a network flow) centralized the controller from different switches and routers, calculated the rules with a global view in the network, and formulated a flow entry back to the devices. The data plane (which is the muscle of forwarding the traffic) works closely with the network device such as switch and router to handle every arriving packet base on the rules made by the controller. However, since the SDN network becomes larger, the security part turns in to be an important role in the SDN. Firewall, a network security system that monitors and control the incoming traffic and outgoing traffic, is the first gate to block the malicious traffic in SDN, if the firewall misses the attack and failed to protect the targeting server, the server would either slow down the speed of operating normal traffic or block the system. Even worse, the target server will crash and cause severe consequences for the entire network.

In this paper, we investigate how SDN can be used to provide malicious traffic by filtering Hypertext transfer protocol (HTTP) request from an endpoint. HTTP is broadly used for browsing a web page, and it is the largest and widely use application layer protocol in the networking world. Floodlight, an open source, java-based project use to present the SDN controller, provides the programmer easy access to built up an SDN network. It offers a module loading system, which makes the programmer simply extend and exchange the existing module. Our goal is to defend the malicious traffic by filtering the HTTP request with the existing module and the newly-created module in a Floodlight controller.

II. FLOODLIGHT

Floodlight controller is an open source, community-developed, java-based controller use for the programmer to implement on the SDN network. It supports OpenFlow protocols version 1.0 through 1.4. The OpenFlow protocol works with both physical and virtual switches and provides a connection between the controller and the network devices, or southbound application program interface (API). Floodlight controller also provides an access for northbound API to communicate with the application develops by the engineers. The application can be a virtual switch, a circuit pusher, a load balancer or a firewall. As shown in Fig 1, the floodlight architecture can describe into three parts. REST API, Module application, and floodlight controller. REST API provides the application to communicate with the controller module and the module application. Programming languages can easily connect to the controller module and expose the REST API over HTTP. Module application allows the programmer to build a module based on its need. For example, a programmer might need to use an existing Static Flow Entry Pusher to insert or delete a flow rule. Besides, a programmer can use a firewall module to apply Access Control List (ACL) rules, which allows or deny the traffic based on the specific match. Floodlight controller is the main block in the architecture which makes the system works. Link Discovery took the responsibility for discovering and maintaining the status of a link in the OpenFlow network. Topology Manager and Routing maintains the topology information for the controller, as well to find routing in the network. Device Manager tracks the devices as they move around a network and defines the destination device for a new flow.

The Floodlight controller has a home-grown module system allows the programmer to decide what program should be configured and run. It defines what module are going to be loaded in a configuration file. It can swap the implementations of the module without modifying the module. It creates a platform and API for programmers to extend in the controller.

The lately released Floodlight source code version 1.2 we are used in our project. We will build up a new module for recording the layer three ICMP protocols, layer four UDP and TCP protocols. Finally add the module to support the HTTP filter, in order to block the malicious traffic from any place to a target server.

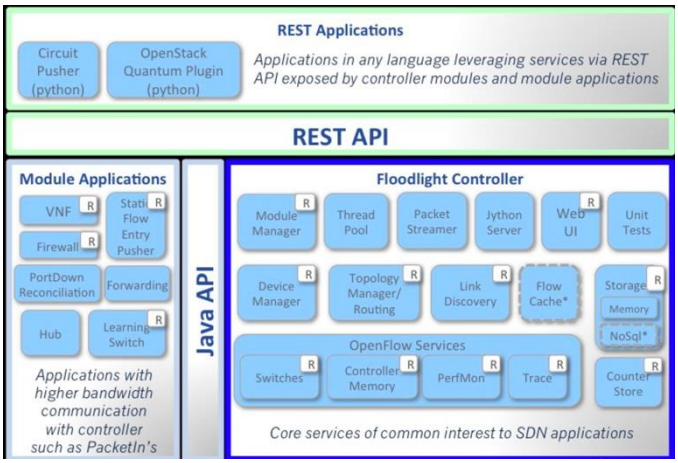

Fig. 1. Floodlight controller and the module

A. *Module*

We can develop a floodlight module based on our needs. When an OpenFlow packet received by a switch, if there is no flow rule to forward this packet, the switch will send a message named PACKET_IN to the SDN controller – floodlight. To get the network packet, we can develop a floodlight module that listens to the PACKET_IN message. Then we can extract information from the network packet, such as source IP address, source MAC address, destination IP address, the protocol of IP packets, etc. In addition, we can run our own flow rules defined by our algorithm. The self-defined floodlight modules are able to provide REST APIs for applications.

A self-defined module must inherit from two interfaces - "IOFMessageListener" and "IFloodlightModule".

IOFMessageListener is an interface act as a message listener. The interface has an important method named “receive”. This method will be called when a packet arrives at the controller. If we want to gather information from PACKET_IN messages, HeaderExtract must inherit from this interface.

IFloodlightModule is the second interface used for the programmer to define a module. If programmers want to develop their own floodlight module, this module must inherit from the interface.

B. *HeaderExtract Module*

The HeaderExtract module is a floodlight module developed by our group. This module is used to extract the headers of network IP packets.

When an OpenFlow packet received by a switch, if there is no flow rule to forward this packet, the switch will send a message named PACKET_IN to the SDN controller – floodlight. We can easily develops a floodlight module which listens to the PACKET_IN message in order to get the network packet. Then we extracts the information from the network packet, such as source IP address, source MAC address, destination IP address, the protocol of IP packets, etc. In addition, we can collect some information which needed by the applications, and provides REST API for these applications to gain the information.

Now we introduced the details of this floodlight module.

HeaderExtract module includes three main parts – a main class named HeaderExtract, some resource classes, and a routable class.

1) The main class - HeaderExtract

Besides the "IOFMessageListener" and "IFloodlightModule" interface, the HeaderExtract class is inherits from an interface named "IHeaderExtractService".

IHeaderExtractService, is an interface that defined by the methods which provides data for the resource classes.

The HeaderExtract class includes the following important member variables:

1.floodlightProvider: Reference of IFloodlightProviderService, which is an interface exposed by the core bundle that allows the developers to interact with connected OpenFlow switches.

2.ICMPMap: A key-value map which stores the source IP addresses that sent by ICMP requests and the count.

3.ICMPDstMap: A key-value map stores the destination IP addresses that sent by ICMP requests and the count.

4.TcpUdpPortMap: A map that records the destination IP Address and its own TCP or UDP ports which requested by the other hosts.

5.restApi: A reference of IRestApiService. It is an interface that indicates the HeaderExtract class. The REST API request is able to handle by the HeaderExtract class.

We implement the following functions of the HeaderExtract class.

1.isCallbackOrderingPostreq: A function used for telling the floodlight controller to run our module before the switches have forward the incoming packets.

2.init: A function that initiated by the member variables in HeaderExtract class.

3.startup: A function allows us to add the PACKET_IN message to the message listener. It also adds the URLs of the REST APIs into this class.

4.receive: A function that extracts the data from network packets and stores the information. The procedure of this method shown in Fig.2.

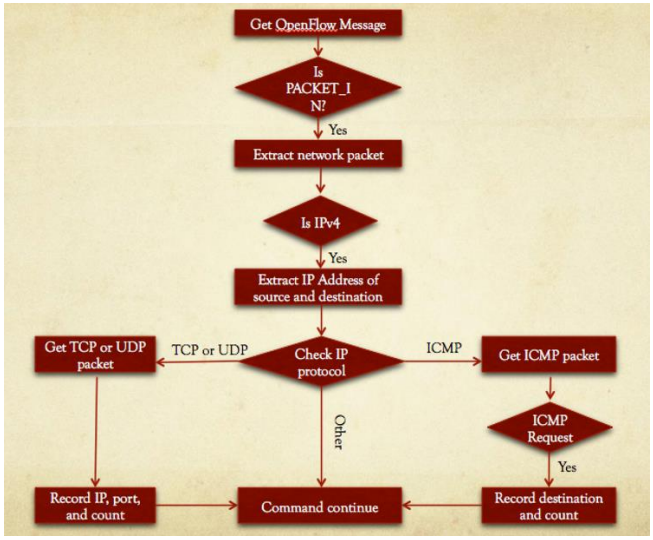

Fig. 2. A block diagram for HeaderExtract

The procedure begins when the OF switch has no flow rule to forward this packet. A PACKET_IN message will be sent by the switch directly to the controller. When the controller received the packet, HeaderExtract will check whether the packet is an IPv4 or not. If the incoming packet is an IPv4 packet, HeaderExtract will extract the source IP address and the destination IP address of the packet. The next step is to check the type of the IP protocol. HeaderExtract will process the network packet according to the IP protocol of the packet. If the packet is an ICMP Request packet, HeaderExtract will record the source IP address and the counts, which will accumulate when an ICMP request has been monitored. Meanwhile, records the destination IP address and the counts; If the packet is a TCP packet, records the destination IP address, TCP port number and the count that every port receives the requests. If the packet is a UDP packet, records the destination IP Address, UDP port number, and the count that every port receives the requests. If the packet is not a TCP, UDP or ICMP packet, leave the HeaderExtract module and continue to the other module.

2) Resource classes
There are three resource classes in this module.
1.ICMPSrcResource: A function which returns the source IP address and the count that it sends ICMP request.
2.ICMPDstResource: A function which returns the destination IP address and the count that it receives ICMP request.
3.TCPUDPPortResource: A function which returns the destination IP address, TCP or UDP ports, and the count that every port receives requests.

3) The routable class - HeaderExtractRoutable
HeaderExtractRoutable defines the URL of the REST APIs, which are provided by the module.
This class includes two functions:
1.basePath, which defines the base URL "/wm/headerExtract" of a REST APIs
2.getRestlet, which defines the specific REST API URL for every resource.

C. HTTPFilter Module

This module is used to block the HTTP requests for specific IP address. Now we introduce the details of this floodlight module. HTTPFilter module includes three main parts – a main class named HTTPFilter, a resource class, and a routable class.
1) The main class - HTTPFilter
Besides the "IOFMessageListener" and "IFloodlightModule" interface, the HTTPFilter class is inherits from an interface named " IHTTPFilterService ".
IHTTPFilterService, is an interface that defined by the methods provides data for resource classes.
The HTTPFilter class includes the following important member variables:
1.floodlightProvider: Reference of IFloodlightProviderService, which is an interface exposed by the core bundle that allows the developers to interact with connected OpenFlow switches.
2.IPAddressSet: A set which stores the IP addresses. The function will store the IP address for those HTTP requests was blocked.
3. restApi: A reference of IRestApiService. It is an interface that indicates the HeaderExtract class. The REST API request is able to handle by the HeaderExtract class.

We implement the following functions of the HTTPFilter class.
1.isCallbackOrderingPostreq: A function used for telling the floodlight controller to run our module before the switches have forward the incoming packets.
2.init: A function that initiated by the member variables in HTTPFilter class.
3.startup: A function allows us to add PACKET_IN message to the message listener. Is also adds the URLs of the REST APIs into this class.
4.receive: A function that extracts the data from network packets and stores the information. The procedure of this method shown in Fig.3.

The procedure begins when the OF switch has no flow rule to forward this packet. A PACKET_IN message will be sent by the switch directly to the controller. When the controller received the packet, HTTPFilter will check whether the packet is an IPv4 or not. If the incoming packet is an IPv4 packet, the program will extract the source IP address and the destination IP address of the packet. The next step is to check whether the protocol is TCP or not. If the packet is a TCP packet, HTTPFilter will extract the payload of the packet. If it is not a TCP packet, leave the HTTPFilter module and continue to the other module. While getting the payload from extracting the TCP packet, we check whether the packet is a HTTP request or not. If it is a HTTP request and the source IP address is in the block list, generate a flow rule by "Match" class, then drop the packet.

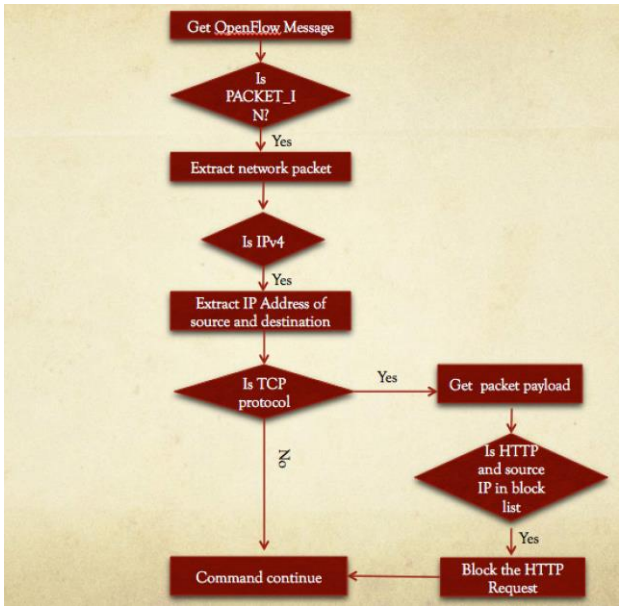

Fig. 3. A block diagram for HTTPFilter

2) Resource class
The resource class – HTTPFilterRoutable, defines three methods in order to get / put data from the applications or to the application.
1.getAllIPAddress: A function which responses a HTTP GET request. This function will return all the IP address which blocked by the module.
2.store: A function which response a HTTP POST request. This function will add an IP address to the block list.
3.remove: A function which response a HTTP DELETE request. This function will remove an IP address from the block list.

3) The routable class - HTTPFilterRoutable
HTTPFilterRoutable defines the URLs of the REST APIs, which are provided by the module.
This class includes two functions:
1.basePath, which defines the base URL "/wm/httpfilter" of a REST APIs
2.getRestlet, which defines the specific REST API URL for the HTTPFilterResource class.

III. Application for DDOS

This is an application which we built in order to block the malicious traffic. The purpose is to filter the ICMP packet when it increases the threshold that we defined. In our case, the threshold is 10000 packets. We had used the sFlow for monitoring and detecting the incoming traffic, and our program will capture the monitoring result via REST API. When we track the traffics and identify the host, our program will set up a rule to block the specific IP address with Floodlight controller.

A. sFlow

sFlow-rt is an application monitors high-speed switch and router in order to to analyze and deliver a real-time visible data to the SDN controller. It developed by InMon and has an analytic engine monitoring the network devices continuously. If a network engineer needs to communicate with the sFlow-rt, sFlow agent, which represent the agency from sFlow-rt, will allow the engineer to do this. sFlow agent has to build in the network device, host, and the application. It provides an actionable graph and can easily access through REST API. Several functions such as configuring the customized measurement, retrieving the metrics, setting the threshold or receiving the notification.
sFlow, short form for "sampled flow", is an industry standard for packet export at Layer 2 i.e., Datalink layer of the OSI model. It provides a means for exporting truncated packets and real time monitoring, together with user-interface counters. Maintenance of the protocol is performed by the sFlow.org online consortium which is the authoritative source of the sFlow protocol specifications. The current version of sFlow is v5.

Previous versions:-

Version	Comment
v1	Initial version
v2	(Unknown)
v3	Adds support for *extended_url* information.[12]
v4	Adds support BGP communities.[12]
v5	Several protocol enhancements.[13] This is the current version, which is globally supported.

Version 5 is the latest one, this version has lots and lots of keys and metrices which can be used specifically to filter the incoming packets or the outgoing packets according to our needs. We can genereate the events according to our needs… and these events can be retrieved by a self-made application….
The sflow monitor is a highly advances monitoring system.
We can monitor the flows according to our needs.it deals with any type of protocols, it is highly user friendly and makes our work easy.
Operatin:-

sFlow uses the sampling to achieve scalability and this is, for this reason, applicable to the high speed networks (gigabit per second speeds and higher and higher). sFlow is supported by a multi-network device manufacturers and network management software vendors like zeroStack..

An sFlow system consists of multiple devices which are performing generally two types of sampling: random sampling of packets or the application layer operations, and time-based sampling of the counters. The sampled packet/operation and counter information, called as the *flow samples* and *counter samples* respectively, are sent as *sFlow datagrams* to a central server in wich there is a running software that analyzes and reports on network traffic; called the *sFlow collector*.

1) Sample Flows

Based on the defined sampling rate, an average of 1 out of the n packets/operations is randomly sampled at a time. This type of sampling does not provide a 100% accurate result, which is possible by no software, but it does provide a result with quantifiable accuracy which is highly desirable..

2) Counter samples

A polling interval defines that how often the network device sends interface counters. sFlow counter sampling is more efficient than SNMP polling while monitoring a large number of interfaces.

1)sFlow datagrams

The sampled data is sent as a UDP packet to the specified host and port. The port for sFlow is port 6343. The lack of reliability in the UDP transport mechanism of sflow does not significantly affect the accuracy of measurements obtained from an sFlow agent which is the computer which has the openflow switch. If the counter samples are lost then new values are generated and will be sent when the next polling interval has passed. The loss of a packet flow samples results in a slight reduction of effective sampling rate.

The UDP payload contains the main *sFlow datagram*. Each datagram provides the information about sFlow version, the originating device's IP address, sequence number, the number of samples which it contains and one or more flows and/or counter samples.

B. Monitoring with SDN

sFlow is adaptable to any type of controller/ networks. sFlow is easy to configure, which allows us easily to monitor the traffic flow with different controller, DevOps and Orchestration by using REST API. It can support different network devices such as routers and switches, virtual openswitches, and hosts such as web server, mail server, and DNS server. It also supports applications such as firewall, load balancing, and Apache Licensing PHP. As shown in the figure 4.

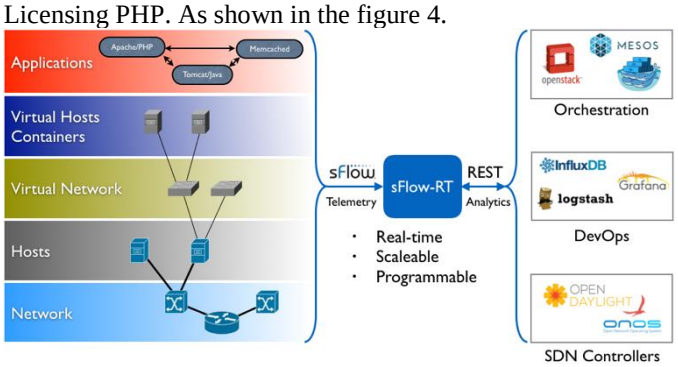

Fig. 4. sFlow monitoring system

C. Working flow

sFlow works perfect with SDN controllers, in figure 4., we have shown the working flow chart of how sFlow interacts with SDN controller and REST API.

Figure.5 tells us about the developed firewall application. Before the program starts, we can define our threshold and the flow rules we want to put. Then when we start the program, the rules and the limitation will automatically post to sFlow monitoring system. Next step starts when there is an event capture by sFlow, events will be generate whenever there is breach of the said threshold value. it will check if the flow rule is a new rule or has been stored in the system. If it is an old rule then it will wait for the new rule. If it is a new event, retrieve the flow from the event, constructs the blocking flow entry and post it to the floodlight controller using REST API. The floodlight controller will then push the rule into the openflow switch and the switch will start blocking the incoming traffic for the specific host which was told by the python application.

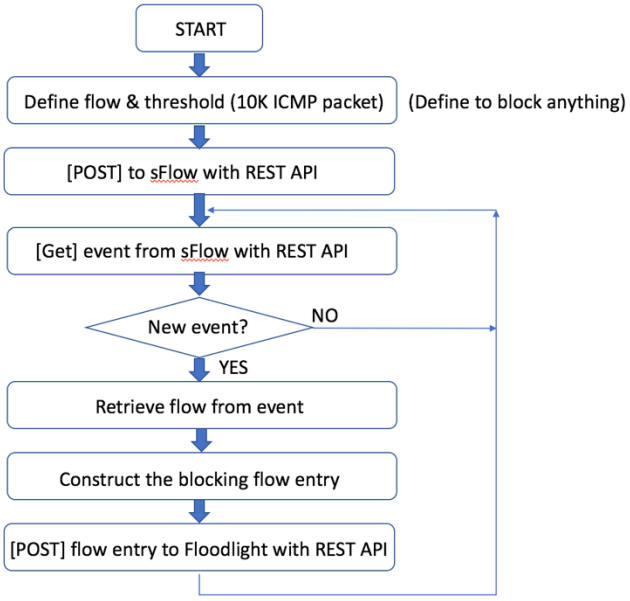

Fig. 5. Firewall application working flow implementation

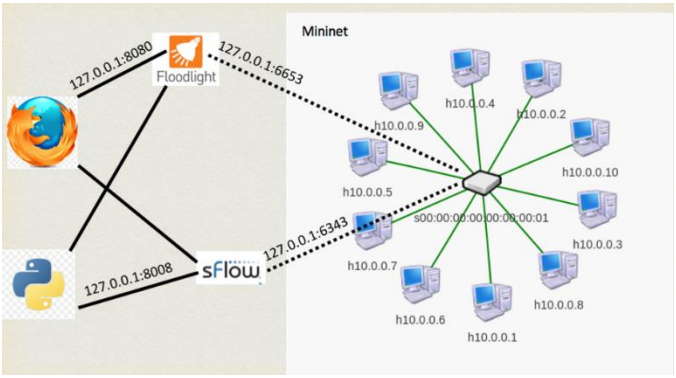

Fig 6. Overview of the topology

The above topology shows how the softwares are inter-linked with each other. Sflow which is the live monitoring system and the Floodlight controller are not directly connected with each other. There is the firewall application which bridges both of them. Sflow monitors the open flow switch about the traffic that is flowing within it. Simultaneously floodlight controller is monitoring the openflow switch.

sFlow works on port-6343 and floodlight works on port-6653. Both of them resides on the same system. Using firefox internet browser we can monitor the flow graph generated by sFlow for the packets which are getting transferred within it.

The firewall application program runs on the algorithm which is shown in figure-5.

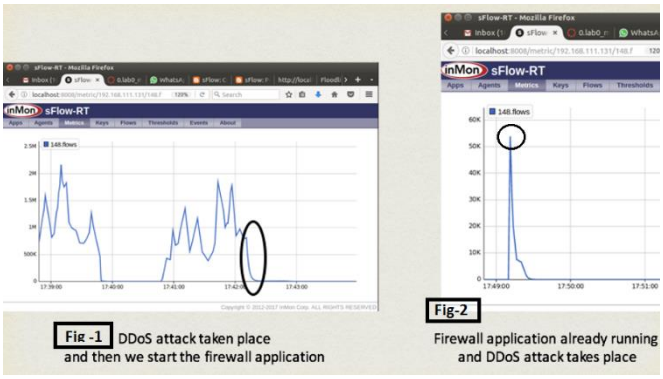

Figure -1, shows that the system is being attacked by a DDoS attack and then after realizing it we launch our firewall application. In fog-1 there is a point indicated, that is the point where the firewall application gets activated, as soon as it gets activated, the application gets the event from sFlow stating that there is a flooding of packets in the in the network and fetches the source and destination ip address, by using the address the flow rule is created by the application to block the network in which the source and destination ip addresses are communicating. These two ip addresses can't communicate with each other, but they can communicate with other hosts. typically for simulation we have created a topology with 10 hosts, I switch and 1 controller.

Figure-2 shows that, initially we turn the firewall application on, and then suddenly DDoS attack occurs. In this case when the DDoS attack happens, there is a sudden rise in the number of packets.as soon as the packets crosses the threshold value assigned, it triggers an event and send it to the firewall application and the application posts the rule to the openflow switch which is seen in figure-7. This is sudden process, that's why there is sudden decline in the in the graph, it shows that the packets are getting discarded.

The design of the application is in such a way that it works on both the situation. It is like an escape key when DDos attack has taken place at your system is a t risk. Just launch the application, you will be safe. When the application is already running then you are at the safest level. Nobody can attack you.

These are the list of rules that are getting posted in the openflow switch which can be seen through the floodlight controller..

Floodlight Dashboard Topology Switches Hosts

Flows (4)

Cookie	Table	Priority	Match	Apply Actions	Write Actions	Clear Actions	Goto Group	Goto Meter	Write Metadata	Experimenter	Packets	Bytes	Age (s)	Timeout (s)
45035999572119925	0x0	30000	eth_type=0x0x800 ipv4_src=10.0.0.2 ipv4_dst=10.0.0.1	---	---	---	---	---	---	---	4061	397978	59	0
45035997935621334	0x0	29999	eth_type=0x0x800 ipv4_src=10.0.0.1 ipv4_dst=10.0.0.3	---	---	---	---	---	---	---	3884	380632	40	0
45035996299122743	0x0	29998	eth_type=0x0x800 ipv4_src=10.0.0.1 ipv4_dst=10.0.0.7	---	---	---	---	---	---	---	2500	245000	33	0
0	0x0	0		actions:output=controller	---	---	---	---	---	---	16	1120	3743	0

Fig-7 rules added by the firewall application

From figure-7 it can be seen that there are 3 rules which are posted in the openflow switch which is seen through the floodlight controller. These rules are posted by the firewall application using the REST API..

According to the figure, there will not be any communication happening between the hosts with ip-10.0.0.1-10.0.0.2, 10.0.0.1-10.0.00.3 and 10.0.0.01-10.0.0.7, but these individual hosts can communicate with other hosts like 10.0.0.5,10.0.0.4,10.0.0.6, etc.

We have included other features in our firewall application which deals with the sFlow, we have created 3 commands as follows:-

- LIST- lists all the rules in the openflow switch
- DELETE RULE # - Hash denotes the id number of the rule, this is used for deleting a specific rule, this command can be used when we later want to allow an ip to communicate with the desired host.
- RESET – This command is used to delete all the rules which is there in the openflow switch.

```
list
restricted
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   568    0   568    0     0  18908      0 --:--:-- --:--:-- --:--:-- 19586
[
    {
        "action": "DENY",
        "id": 1,
        "nw_dst": "10.0.0.2/32",
        "nw_dst_maskbits": 32,
        "nw_dst_prefix": 167772162,
        "nw_proto": 0,
        "nw_src": "10.0.0.1/32",
        "nw_src_maskbits": 32,
        "nw_src_prefix": 167772161,
        "tp_dst": 0
    },
    {
        "action": "DENY",
        "id": 2,
        "nw_dst": "10.0.0.3/32",
        "nw_dst_maskbits": 32,
        "nw_dst_prefix": 167772163,
        "nw_proto": 0,
        "nw_src": "10.0.0.1/32",
        "nw_src_maskbits": 32,
        "nw_src_prefix": 167772161,
        "tp_dst": 0
    },
    {
        "action": "DENY",
        "id": 3,
        "nw_dst": "10.0.0.7/32",
        "nw_dst_maskbits": 32,
        "nw_dst_prefix": 167772167,
        "nw_proto": 0,
        "nw_src": "10.0.0.1/32",
        "nw_src_maskbits": 32,
        "nw_src_prefix": 167772161,
        "tp_dst": 0
    }
]
reset
all rules deleted
```

REFERENCES

[1] I. F. Akyildiz, A. Lee, P. Wang, M. Luo and W. Chou, "Research challenges for traffic engineering in software defined networks," in IEEE Network, vol. 30, no. 3, pp. 52-58, May-June 2016.

[2] Nhu-Ngoc Dao, Junho Park, Minho Park and Sungrae Cho, "A feasible method to combat against DDoS attack in SDN network," 2015 International Conference on Information Networking (ICOIN), Cambodia, 2015, pp. 309-311.

[3] J. Collings and J. Liu, "An OpenFlow-Based Prototype of SDN-Oriented Stateful Hardware Firewalls," 2014 IEEE 22nd International Conference on Network Protocols, Raleigh, NC, 2014, pp. 525-528.

[4] N. I. G. Dharma, M. F. Muthohar, J. D. A. Prayuda, K. Priagung and D. Choi, "Time-based DDoS detection and mitigation for SDN controller," 2015 17th Asia-Pacific Network Operations and Management Symposium (APNOMS), Busan, 2015, pp. 550-553.

[5] P. Kampanakis, H. Perros, T. Beyene, "Sdn-based solutions for moving target defense network protection", A World of Wireless Mobile and Multimedia Networks (WoWMoM) 2014 IEEE 15th International Symposium, pp. 1-6, 2014.

[6] S. Lim, J. Ha, H. Kim, Y. Kim, S. Yang, "A SDN-oriented DDoS Blocking Scheme for Botnet-based Attacks", 2014 Sixth International Conference on Ubiquitous and Future Networks, Jul. 2014.

[7] Hongxin Hu , Wonkyu Han , Gail-Joon Ahn , Ziming Zhao, FLOWGUARD: building robust firewalls for software-defined networks, Proceedings of the third workshop on Hot topics in software defined networking, August 22-22, 2014, Chicago, Illinois, USA

[8] Jafar Haadi Jafarian , Ehab Al-Shaer , Qi Duan, Openflow random host mutation: transparent moving target defense using software defined networking, Proceedings of the first workshop on Hot topics in software defined networks, August 13-13, 2012, Helsinki, Finland

[9] Bruce Davie, Teemu Koponen, Ben Pfaff, Justin Pettit, Anupam Chanda, Martin Casado, Kenneth Duda, Natasha Gude & Amar and Tim Petty. "A Database Approach to SDN Control Plane Design." ACM SIGCOMM Computer Communication Review Volume 47 Issue 1, January 2017.

[10] Josep Batalle, Jordi Ferrer Riera, Eduard Escalona and Joan A. Garcia-Espin. "On the Implementation of NFV over an OpenFlow Infrastructure: Routing Function Virtualization", Proc.IEEE SDN4FNS,pp.1-6,Nov.2013.

[11] M. Suh, S. H. Park, B. Lee, and S. Yang, "Building firewall over the software-defined network controller," in Advanced Communication Technology (ICACT), 2014 16th International Conference on, Feb 2014, pp. 744-748.

[12] D. Drutskoy, E. Keller, J. Rexford, "Scalable network virtualization in software-defined networks", IEEE Internet Computing, vol. 17, no. 2, pp. 20-27, 2013.

[13] N. M. K. Chowdhury and R. Boutaba, "A survey of network virtualization," Computer Networks, vol. 54, no. 5, pp. 862 – 876, 2010.

[14] sflow Application Development sflow-rt.com/writing_applications.php

[15] sflow REST API Functions sflow-rt.com/reference.php

[16] C. Buragohain and N. Medhi, "FlowTrApp: An SDN based architecture for DDoS attack detection and mitigation in data centers," *2016 3rd International Conference on Signal Processing and Integrated Networks (SPIN)*, Noida, 2016, pp. 519-524.

[17] P. H. A. Rezende, P. R. S. L. Coelho, L. F. Faina, L. Camargos and R. Pasquini, "A Platform for Monitoring Service-Level Metrics in Software Defined Networks," *2015 XXXIII Brazilian Symposium on Computer Networks and Distributed Systems*, Vitoria, 2015, pp. 19-30.

Appendix

Add code analysis through comments, the comments are in red color.

I. HEADEREXTRACT MODULE SOURCE CODE

HeaderExtract.java

```
package net.floodlightcontroller.headerextract;

import java.util.ArrayList;
import java.util.Collection;
import java.util.HashMap;
import java.util.Map;

import org.projectfloodlight.openflow.protocol.OFMessage;
import org.projectfloodlight.openflow.protocol.OFType;
import org.projectfloodlight.openflow.types.EthType;
import org.projectfloodlight.openflow.types.IpProtocol;
import org.projectfloodlight.openflow.types.TransportPort;

import net.floodlightcontroller.core.FloodlightContext;
import
net.floodlightcontroller.core.IFloodlightProviderService;
import net.floodlightcontroller.core.IOFMessageListener;
import net.floodlightcontroller.core.IOFSwitch;
import
net.floodlightcontroller.core.module.FloodlightModuleContex
t;
import
net.floodlightcontroller.core.module.FloodlightModuleExcepti
on;
import
net.floodlightcontroller.core.module.IFloodlightModule;
import
net.floodlightcontroller.core.module.IFloodlightService;
import net.floodlightcontroller.packet.Ethernet;
import net.floodlightcontroller.packet.ICMP;
import net.floodlightcontroller.packet.IPv4;
import net.floodlightcontroller.packet.TCP;
import net.floodlightcontroller.packet.UDP;
import net.floodlightcontroller.restserver.IRestApiService;
//This is the main class for HeaderExtract Module.
public class HeaderExtract implements IOFMessageListener,
IHeaderExtractService, IFloodlightModule {
   protected IFloodlightProviderService floodlightProvider;
   //ICMP src IP and counts
   protected Map<String, Integer> ICMPMap;
   //ICMP dst IP and counts
   protected Map<String, Integer> ICMPDstMap;
   //TCP or UDP dst IP, port and counts
   protected Map<String, Map<String, Integer>>
TcpUdpPortMap;
   //For providing REST API
   protected IRestApiService restApi;

   //module name
```

```
    @Override
    public String getName() {
        // TODO Auto-generated method stub
        return HeaderExtract.class.getSimpleName();
    }
    @Override
    public boolean isCallbackOrderingPrereq(OFType type,
String name) {
        // TODO Auto-generated method stub
        return false;
    }
    //Call our module before forwarding
    @Override
    public boolean isCallbackOrderingPostreq(OFType type,
String name) {
        // TODO Auto-generated method stub
        return (type.equals(OFType.PACKET_IN) &&
name.equals("forwarding"));
    }

    @Override
    public Collection<Class<? extends IFloodlightService>>
getModuleServices() {
        // TODO Auto-generated method stub
        Collection<Class<? extends IFloodlightService>> l =
new ArrayList<Class<? extends IFloodlightService>>();
        l.add(IHeaderExtractService.class);
        return l;
    }

    @Override
    public Map<Class<? extends IFloodlightService>,
IFloodlightService> getServiceImpls() {
        // TODO Auto-generated method stub
        Map<Class<? extends IFloodlightService>,
IFloodlightService> m = new HashMap<Class<? extends
IFloodlightService>, IFloodlightService>();
        m.put(IHeaderExtractService.class, this);
        return m;
    }

    @Override
    public Collection<Class<? extends IFloodlightService>>
getModuleDependencies() {
         // TODO Auto-generated method stub
        Collection<Class<? extends IFloodlightService>> l =
        new ArrayList<Class<? extends IFloodlightService>>();
        l.add(IFloodlightProviderService.class);
        l.add(IRestApiService.class);
        return l;
    }
    //Initialize member variables
    @Override
    public void init(FloodlightModuleContext context)
                                throws FloodlightModuleException
{
         // TODO Auto-generated method stub
        floodlightProvider =
context.getServiceImpl(IFloodlightProviderService.class);
        restApi = context.getServiceImpl(IRestApiService.class);
        ICMPMap = new HashMap<String, Integer>();
        ICMPDstMap = new HashMap<String, Integer>();
        TcpUdpPortMap = new HashMap<String, Map<String,
Integer>>();
    }

    //Add PACKET_IN to message listener and module REST
APIs
    @Override
    public void startUp(FloodlightModuleContext context)
                                throws FloodlightModuleException
{
        // TODO Auto-generated method stub
floodlightProvider.addOFMessageListener(OFType.PACKET
_IN, this);
        restApi.addRestletRoutable(new
HeaderExtractRoutable());
    }
    //If PACKET_IN message arrived, invoke this method
    @Override
    public net.floodlightcontroller.core.IListener.Command
receive(IOFSwitch sw, OFMessage msg, FloodlightContext
cntx) {
        // TODO Auto-generated method stub
        /* Retrieve the deserialized packet in message */
        switch (msg.getType()){
        case PACKET_IN:
            //Get network frames
            Ethernet eth =
IFloodlightProviderService.bcStore.get(cntx,
IFloodlightProviderService.CONTEXT_PI_PAYLOAD);

            /*
             * Check the ethertype of the Ethernet frame and
retrieve the appropriate payload.
             * Note the shallow equality check. EthType caches
and reuses instances for valid types.
             */
            if (eth.getEtherType() == EthType.IPv4) {
           /* We got an IPv4 packet; get the payload from
Ethernet */
            IPv4 ipv4 = (IPv4) eth.getPayload();
            /* Get Source IP address and destination IP address */
           String dstIp = ipv4.getDestinationAddress().toString();
           String sourceAddress =
ipv4.getSourceAddress().toString();
           // Check whether the IP protocol is ICMP
           if (ipv4.getProtocol() == IpProtocol.ICMP) {
              //Get payload
             ICMP icmp = (ICMP) ipv4.getPayload();
             //Check whether payload is ICMP request
             if (icmp.getIcmpType() == ICMP.ECHO_REQUEST)
{
                 //record Source IP and count
```

```
            ICMPMap.put(sourceAddress,
ICMPMap.getOrDefault(sourceAddress, 0) + 1);
             //record destination IP and count
             ICMPDstMap.put(dstIp,
ICMPDstMap.getOrDefault(dstIp, 0) + 1);
          }
        }
         //check whether packets are TCP packets?
         else if (ipv4.getProtocol() == IpProtocol.TCP) {
          /* Get a TCP packet; get the payload from IPv4 */
            TCP tcp = (TCP) ipv4.getPayload();
            /* Get destination port */
            TransportPort dstPort = tcp.getDestinationPort();
           //If port number less than 10000
            if (dstPort.getPort() < 10000) {
               HashMap<String, Integer> hmap = null;
              //Whether already has destination IP record
              if (TcpUdpPortMap.containsKey(dstIp)) {
                 //get port and count map
                 hmap = (HashMap<String, Integer>)
TcpUdpPortMap.get(dstIp);
                 //increase count by 1
                  hmap.put(dstPort.toString(),
hmap.getOrDefault(dstPort.toString(), 0) + 1);
              }
              else {
                 // create a new port and count map
                  hmap = new HashMap<String, Integer>();
                 //add a new record to port and count Map
                 hmap.put(dstPort.toString(), 1);
               }
              //put port and count map to destination IP map
              TcpUdpPortMap.put(dstIp, hmap);
          }
          //check whether packets are UDP packets
          } else if (ipv4.getProtocol() == IpProtocol.UDP) {
             /* Get  a UDP packet; get the payload from IPv4 */
              UDP udp = (UDP) ipv4.getPayload();
             //get destination port number
              TransportPort dstPort = udp.getDestinationPort();
             // If port number less than 10000
              if (dstPort.getPort() < 10000) {
                 HashMap<String, Integer> hmap = null;
                 //If already has destination IP record
                 if (TcpUdpPortMap.containsKey(dstIp)) {
                    //get port and count map
                    hmap = (HashMap<String, Integer>)
TcpUdpPortMap.get(dstIp);
                   //increase count by 1
                    hmap.put(dstPort.toString(),
hmap.getOrDefault(dstPort.toString(), 0) + 1);
                 }
                 else {
                   //Create a new port and count map
                    hmap = new HashMap<String, Integer>();
                   //put a new record – port and count
                    hmap.put(dstPort.toString(), 1);
                 }
                 //put port and count map to destination IP map
                 TcpUdpPortMap.put(dstIp, hmap);
              }
            }
          }
          default:
              break;
        }
        //continue to receive PACKET_IN msg
        return Command.CONTINUE;
        }
    //return ICMP request source IP and counts map
    @Override
    public Map<String, Integer> getResults() {
       // TODO Auto-generated method stub
        return this.ICMPMap;
    }
    //return ICMP request dst IP and counts map
    @Override
    public Map<String, Integer> getDstIPCountMap() {
       // TODO Auto-generated method stub
       return this.ICMPDstMap;
    }
    //return dst IP, port and count map
    @Override
    public Map<String, Map<String, Integer>>
getTcpUdpPortMap() {
       // TODO Auto-generated method stub
       return TcpUdpPortMap;
    }
}
```

IHeaderExtractService.java

```
package net.floodlightcontroller.headerextract;

import java.util.Map;
import
net.floodlightcontroller.core.module.IFloodlightService;
//This interface is used for Resource classes to get data in
HeaderExtract main class
public interface IHeaderExtractService extends
IFloodlightService {
   //Get ICMP src IP and count map for ICMPSrcResource
   public Map<String, Integer> getResults();
   //Get ICMP dst IP and count map for ICMPDstResource
   public Map<String, Integer> getDstIPCountMap();
   //Get dst IP, port and count for TCPUDPIPCountResource
   public Map<String, Map<String, Integer>>
getTcpUdpPortMap();
}
```

ICMPSrcResource.java

```
package net.floodlightcontroller.headerextract;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
```

```
import java.util.Map.Entry;

import org.restlet.resource.ServerResource;
import org.restlet.resource.Get;
//This class provides ICMP src IP address and count
public class ICMPSrcResource extends ServerResource {
   //Provide ICMP src IP and count data to application
   @Get("json")
   public List<ICMPIPCountPair> retrieve() {
      //Get service Instance
      IHeaderExtractService serviceInstance =
(IHeaderExtractService)getContext().getAttributes().get(IHea
derExtractService.class.getCanonicalName());
      //Create a new list for return value
      List<ICMPIPCountPair> l = new
ArrayList<ICMPIPCountPair>();
      //Get src IP address and count Map
      Map<String, Integer> hmap =
serviceInstance.getResults();
      //define a iterator to iterate src IP and count Map
      Iterator<Entry<String, Integer> > iter =
hmap.entrySet().iterator();
      while (iter.hasNext()) {
          Entry<String, Integer> entry = iter.next();
          //add a record that includes src IP and count to list
          l.add(new ICMPIPCountPair(entry.getKey(),
entry.getValue()));
      }
      return l;
   }
}
```

ICMPDstResource.java

```
package net.floodlightcontroller.headerextract;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;

import org.restlet.resource.Get;
import org.restlet.resource.ServerResource;
//This class provides ICMP dst IP address and count
public class ICMPDstResource extends ServerResource {
   //Provide ICMP dst IP and count data to application
   @Get("json")
   public List<ICMPIPCountPair> retrieve() {
      //Get service Instance
      IHeaderExtractService serviceInstance =
(IHeaderExtractService)getContext().getAttributes().get(IHea
derExtractService.class.getCanonicalName());
      //Create a new list for returning dst IP and count
      List<ICMPIPCountPair> l = new
ArrayList<ICMPIPCountPair>();
      //Get dst IP and count Map
      Map<String, Integer> hmap =
serviceInstance.getDstIPCountMap();
      //define a iterator to iterate dst IP and count Map
      Iterator<Entry<String, Integer> > iter =
hmap.entrySet().iterator();
      while (iter.hasNext()) {
          Entry<String, Integer> entry = iter.next();
          //add a new record that includes dst IP and count to
list
          l.add(new ICMPIPCountPair(entry.getKey(),
entry.getValue()));
      }

      return l;
   }
}
```

TCPUDPPortResource.java

```
package net.floodlightcontroller.headerextract;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;

import org.restlet.resource.Get;
import org.restlet.resource.ServerResource;
//This class provides IP address, tcp or udp port  and count
public class TCPUDPPortResource extends ServerResource {
   //provides dst IP, port and count data to applications
   @Get("json")
   public List<TCPUDPIPPortPair> retrieve() {
      //Get service Instance
      IHeaderExtractService serviceInstance =
(IHeaderExtractService)getContext().getAttributes().get(IHea
derExtractService.class.getCanonicalName());
      //create a new list for returning dst IP, port and count
      List<TCPUDPIPPortPair> l = new
ArrayList<TCPUDPIPPortPair>();
      //Get dst IP, port and count Map
      Map<String, Map<String, Integer>> hmap =
serviceInstance.getTcpUdpPortMap();
      //defines a iterator to iterate dst IP, port, count Map
      Iterator<Entry<String, Map<String, Integer>>> iter =
hmap.entrySet().iterator();

      while (iter.hasNext()) {
          Entry<String, Map<String, Integer>> entry =
iter.next();
          //add a record that includes dst IP, port, count to list
          l.add(new TCPUDPIPPortPair(entry.getKey(),
entry.getValue()));
      }

      return l;
   }
}
```

HeaderExtractRoutable.java

```
package net.floodlightcontroller.headerextract;
```

```
import org.restlet.Context;
import org.restlet.Restlet;
import org.restlet.routing.Router;

import net.floodlightcontroller.restserver.RestletRoutable;
//This class defines REST API URL for this module
public class HeaderExtractRoutable implements
RestletRoutable {

    @Override
    public Restlet getRestlet(Context context) {
        // TODO Auto-generated method stub
        Router router = new Router(context);
       //defines specific URL for returning src ICMP IP and
count
        router.attach("/icmpsrc/json", ICMPSrcResource.class);
       //defines specific URL for returning dst ICMP IP and
count
        router.attach("/icmpdst/json", ICMPDstResource.class);
        //defines specific URL for returning TCP or UPD dst IP
port, and count
        router.attach("/ipport/json",
TCPUDPPortResource.class);
        return router;
    }
    //define base URL for REST API
    @Override
    public String basePath() {
        // TODO Auto-generated method stub
        return "/wm/headerExtract";
    }
}
```

ICMPIPCountPair.java

```
package net.floodlightcontroller.headerextract;

import
com.fasterxml.jackson.databind.annotation.JsonSerialize;
import
net.floodlightcontroller.core.web.serializers.ICMPIPCountPair
Serializer;
//defines the class to serialize ICMP IP and count to Json
string
@JsonSerialize(using=ICMPIPCountPairSerializer.class)
public class ICMPIPCountPair {
    //ICMP src or dst IP
    private String sourceIp;
    //ICMP request count
    private int count;
    //default contructor
    public ICMPIPCountPair() {

    }
    //define a specific contructor has two parameters – IP and
count
     public ICMPIPCountPair(String ip, int count) {
         this.sourceIp = ip;
         this.count = count;
     }

    //setter and getter of two member variables – IP and count
     public String getSourceIp() {
         return sourceIp;
     }

     public void setSourceIp(String sourceIp) {
         this.sourceIp = sourceIp;
     }

     public int getCount() {
         return count;
     }

    public void setCount(int count) {
        this.count = count;
    }
}
```

ICMPIPCountPairSerializer.java

```
package net.floodlightcontroller.core.web.serializers;

import java.io.IOException;

import com.fasterxml.jackson.core.JsonGenerator;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.JsonSerializer;
import com.fasterxml.jackson.databind.SerializerProvider;
import
net.floodlightcontroller.headerextract.ICMPIPCountPair;
//This class is used to serialize ICMPIPCountPair to Json
string
public class ICMPIPCountPairSerializer extends
JsonSerializer<ICMPIPCountPair> {
    //Serialize ICMPIPCountPair list to Json string
    @Override
    public void serialize(ICMPIPCountPair p, JsonGenerator
jGen,SerializerProvider arg2) throws IOException,
JsonProcessingException {
        // TODO Auto-generated method stub
        //start to write a json object
        jGen.writeStartObject();
        //name of the json object is message.
        jGen.writeFieldName("message");
        //start to write child json object
        jGen.writeStartObject();
        //If ICMPIPCountPair object  is not null
        if (p != null) {
            //write two element - IPAddress and ICMP request
count to Json string
            jGen.writeStringField("IPAddress", p.getSourceIp());
            jGen.writeNumberField("count", p.getCount());
        }
        //write end for child json object and parent json object
        jGen.writeEndObject();
        jGen.writeEndObject();
```

```
    }
}
```

TCPUDPIPPortPair.java

```
//This class stores dst IP address, port and count
package net.floodlightcontroller.headerextract;

import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Map.Entry;
import
com.fasterxml.jackson.databind.annotation.JsonSerialize;
//defines TCPUDPIPPortPairSerializer to serialize this object
to Json string
@JsonSerialize(using=TCPUDPIPPortPairSerializer.class)
public class TCPUDPIPPortPair {
    //destination IP address
    private String ipaddress;
    //destination port number and count requested by src.
    private Map<String, Integer> portMap;

    //constructor
    TCPUDPIPPortPair(String ip, Map<String, Integer> hmap)
{
        this.ipaddress = ip;
        Iterator<Entry<String, Integer>> iter =
hmap.entrySet().iterator();
        portMap = new HashMap<String, Integer>();
        while (iter.hasNext()) {
            Map.Entry<String, Integer> entry = iter.next();
            portMap.put(entry.getKey(), entry.getValue());
        }
    }

    //setter and getter for member variables
    public String getIpaddress() {
        return ipaddress;
    }

    public void setIpaddress(String ipaddress) {
        this.ipaddress = ipaddress;
    }

    public Map<String, Integer> getPortMap() {
        return portMap;
    }

    public void setPortMap(Map<String, Integer> portMap) {
        this.portMap = portMap;
    }
}
```

TCPUDPIPPortPairSerializer.java

```
package net.floodlightcontroller.headerextract;

import java.io.IOException;
import java.util.Iterator;
import java.util.Map;
import java.util.Map.Entry;

import com.fasterxml.jackson.core.JsonGenerator;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.JsonSerializer;
import com.fasterxml.jackson.databind.SerializerProvider;
//This class is used to serialize TCPUDPIPPortPair to Json
string
public class TCPUDPIPPortPairSerializer extends
JsonSerializer<TCPUDPIPPortPair> {
    //Serialize TCPUDPIPPortPair list to Json string
    @Override
    public void serialize(TCPUDPIPPortPair p, JsonGenerator
jGen,SerializerProvider arg2) throws IOException,
    JsonProcessingException {
        // TODO Auto-generated method stub
        //Start to write Json object
        jGen.writeStartObject();
        //write Json object ipaddress and its value
        jGen.writeStringField("ipaddress", p.getIpaddress());
        //start to write Json array named “ports”
        jGen.writeArrayFieldStart("ports");
        //Get ports and count Map
        Map<String, Integer> hmap = p.getPortMap();
        Iterator<Entry<String, Integer>> iter =
hmap.entrySet().iterator();

        while (iter.hasNext()) {
            //write every requested port of dst IP and its
requested count to Json
            jGen.writeStartObject();
            Entry<String, Integer> entry = iter.next();
            //port number
            jGen.writeStringField("port", entry.getKey());
            //requested count
            jGen.writeNumberField("count", entry.getValue());
            jGen.writeEndObject();
        }
        //write end to Json
        jGen.writeEndArray();
        jGen.writeEndObject();
    }
}
```

I. HTTPFILTER MODULE SOURCE CODE

HTTPFilter.java

```
package net.floodlightcontroller.httpfilter;

import java.util.ArrayList;
import java.util.Collection;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;

import org.projectfloodlight.openflow.protocol.OFFactories;
```

```
import org.projectfloodlight.openflow.protocol.OFMessage;
import org.projectfloodlight.openflow.protocol.OFPacketIn;
import org.projectfloodlight.openflow.protocol.OFType;
import org.projectfloodlight.openflow.protocol.OFVersion;
import org.projectfloodlight.openflow.types.EthType;
import org.projectfloodlight.openflow.types.IpProtocol;
import org.projectfloodlight.openflow.types.MacAddress;
import org.projectfloodlight.openflow.types.OFPort;

import net.floodlightcontroller.core.FloodlightContext;
import
net.floodlightcontroller.core.IFloodlightProviderService;
import net.floodlightcontroller.core.IOFMessageListener;
import net.floodlightcontroller.core.IOFSwitch;
import net.floodlightcontroller.core.IListener.Command;
import
net.floodlightcontroller.core.module.FloodlightModuleContex
t;
import
net.floodlightcontroller.core.module.FloodlightModuleExcepti
on;
import
net.floodlightcontroller.core.module.IFloodlightModule;
import
net.floodlightcontroller.core.module.IFloodlightService;
import
net.floodlightcontroller.devicemanager.IDeviceService;
import net.floodlightcontroller.packet.Data;
import net.floodlightcontroller.packet.Ethernet;
import net.floodlightcontroller.packet.IPv4;
import net.floodlightcontroller.packet.TCP;
import net.floodlightcontroller.packet.UDP;
import net.floodlightcontroller.restserver.IRestApiService;
import net.floodlightcontroller.routing.IRoutingDecision;
import net.floodlightcontroller.routing.RoutingDecision;

import org.projectfloodlight.openflow.protocol.match.Match;
import
org.projectfloodlight.openflow.protocol.match.MatchField;
//This is the main class of HTTPFilter Module
public class HTTPFilter implements IOFMessageListener,
IHTTPFilterService, IFloodlightModule {
    //blocked IP address set
    protected Set<String> IPAddressSet;
    protected IFloodlightProviderService floodlightProvider;
    //For REST API of this module
    protected IRestApiService restApi;

    //Get module name
    @Override
    public String getName() {
        // TODO Auto-generated method stub
        return HTTPFilter.class.getSimpleName();
    }

    @Override
    public boolean isCallbackOrderingPrereq(OFType type,
String name) {
        // TODO Auto-generated method stub
        return false;
    }

    //run this module before switch forward packets
    @Override
    public boolean isCallbackOrderingPostreq(OFType type,
String name) {
        // TODO Auto-generated method stub
        return (type.equals(OFType.PACKET_IN) &&
name.equals("forwarding"));
    }

    @Override
    public Collection<Class<? extends IFloodlightService>>
getModuleServices() {
        // TODO Auto-generated method stub
        Collection<Class<? extends IFloodlightService>> l =
new ArrayList<Class<? extends IFloodlightService>>();
        l.add(IHTTPFilterService.class);
        return l;
    }

    @Override
    public Map<Class<? extends IFloodlightService>,
IFloodlightService> getServiceImpls() {
        // TODO Auto-generated method stub
        Map<Class<? extends IFloodlightService>,
IFloodlightService> m = new HashMap<Class<? extends
IFloodlightService>, IFloodlightService>();
        m.put(IHTTPFilterService.class, this);
        return m;
    }

    @Override
    public Collection<Class<? extends IFloodlightService>>
getModuleDependencies() {
        // TODO Auto-generated method stub
        Collection<Class<? extends IFloodlightService>> l =
        new ArrayList<Class<? extends IFloodlightService>>();
        l.add(IFloodlightProviderService.class);
        l.add(IRestApiService.class);
        return l;
    }

    //Initialize member variables
    @Override
    public void init(FloodlightModuleContext context)
                    throws FloodlightModuleException {
        // TODO Auto-generated method stub
        floodlightProvider =
context.getServiceImpl(IFloodlightProviderService.class);
        restApi = context.getServiceImpl(IRestApiService.class);
        IPAddressSet = new HashSet<String>();
    }
```

```
    //Add message Listener for PACKET_IN and REST API
    @Override
    public void startUp(FloodlightModuleContext context)
                        throws FloodlightModuleException {
        // TODO Auto-generated method stub

    floodlightProvider.addOFMessageListener(OFType.PACK
ET_IN, this);
        restApi.addRestletRoutable(new HTTPFilterRoutable());
    }

    //When PACKET_IN msg arrived, invoke this method
    @Override
    public net.floodlightcontroller.core.IListener.Command
receive(IOFSwitch sw, OFMessage msg, FloodlightContext
cntx) {
        // TODO Auto-generated method stub
        //receive network frames
        Ethernet eth =
IFloodlightProviderService.bcStore.get(cntx,
IFloodlightProviderService.CONTEXT_PI_PAYLOAD);
        //convert msg to Packetin message
        OFPacketIn pi = (OFPacketIn) msg;
        //check whether message is PACKET_IN
        switch (msg.getType()){
        case PACKET_IN:
             //check whether the packet is IPv4 packet
            if (eth.getEtherType() == EthType.IPv4) {
            /* We got an IPv4 packet; get the payload from
Ethernet */
            IPv4 ipv4 = (IPv4) eth.getPayload();
            //Get source IP address from IPv4 packet
            String sourceAddress =
ipv4.getSourceAddress().toString();
            //If the packet protocol is TCP
            if (ipv4.getProtocol() == IpProtocol.TCP) {
                //Get TCP payload from IPv4 packet
                TCP tcp = (TCP) ipv4.getPayload();
                //Get data payload from TCP segment
                Data data = (Data) tcp.getPayload();
                byte[] bytes = data.getData();
                //If TCP payload is not empty
                if (bytes.length > 0) {
                    //assign payload to a String
                    String s = new String(bytes);
                            /*If data in TCP is HTTP protocol
                     * and source IP is in block set, drop this
                     * HTTP request
                     */
                    if (s.contains("HTTP") &&
IPAddressSet.contains(sourceAddress)) {
                        //Get switch inport
                        OFPort inPort =
(pi.getVersion().compareTo(OFVersion.OF_12) < 0 ?
pi.getInPort() : pi.getMatch().get(MatchField.IN_PORT));
                        /*Prepare to create a drop action
                         * to this switch
                         */
                        IRoutingDecision decision = new
            RoutingDecision(sw.getId(), inPort,
                                IDeviceService.fcStore.get(cntx,
            IDeviceService.CONTEXT_SRC_DEVICE),
                             IRoutingDecision.RoutingAction.DROP);
                        /*Create match conditions, if the packet
match
                         *these conditions, such as inport, source IP,
                         *source Port, and etc. drop the packes
                         */
                        Match.Builder mb =
            OFFactories.getFactory(pi.getVersion()).buildMatch(
            );
            mb.setExact(MatchField.IN_PORT,inPort).setExact(
            MatchField.ETH_SRC,eth.getSourceMACAddress())
            .setExact(MatchField.ETH_DST,eth.getDestination
MACAddress()).setExact(MatchField.ETH_TYPE,
eth.getEtherType());
            mb.setExact(MatchField.IPV4_SRC,
ipv4.getSourceAddress()).setExact(MatchField.IPV4_DST,
ipv4.getDestinationAddress()).setExact(MatchField.IP_PROT
O, ipv4.getProtocol());
            mb.setExact(MatchField.TCP_SRC,
tcp.getSourcePort()).setExact(MatchField.TCP_DST,tcp.getD
estinationPort());
            Match match = mb.build();
             //set flow rules to switch
            decision.setMatch(match);
            decision.addToContext(cntx);
                    System.out.println("Http protocol
detected");
            }
          }
        }
    }
      }
      return Command.CONTINUE;
    }

    //return blocked IP set
    @Override
    public Set<String> getAllIPAddress() {
        // TODO Auto-generated method stub
        return IPAddressSet;
    }
    //Add a new IP Address to blocked IP set
    @Override
    public void addIPAddress(String IPAddress) {
        // TODO Auto-generated method stub
        IPAddressSet.add(IPAddress);
    }
    //delete a new IP Address from blocked IP set
    @Override
    public void deleteIPAddress(String IPAddress) {
        // TODO Auto-generated method stub
```

```
        IPAddressSet.remove(IPAddress);
    }
}
```

IHTTPFilterService.java

```
package net.floodlightcontroller.httpfilter;

import java.util.Set;
import
net.floodlightcontroller.core.module.IFloodlightService;

public interface IHTTPFilterService extends
IFloodlightService {
    /*Provides interface for HTTPFilterResource to manage
     *blocked IP Set
     */
    public Set<String> getAllIPAddress();
    public void addIPAddress(String IPAddress);
    public void deleteIPAddress(String IPAddress);
}
```

HTTPFilterResource.java

```
package net.floodlightcontroller.httpfilter;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Set;

import org.restlet.resource.Delete;
import org.restlet.resource.Get;
import org.restlet.resource.Post;
import org.restlet.resource.ServerResource;

import com.fasterxml.jackson.core.JsonParser;
import com.fasterxml.jackson.core.JsonToken;
import com.fasterxml.jackson.databind.MappingJsonFactory;

public class HTTPFilterResource extends ServerResource {
    //For Http Get request, return blocked IP list to applications
    @Get("json")
    public List<String> getAllIPAddress() {
        //Create a new return List
        List<String> l = new ArrayList<String>();
        //Get service Instance
        IHTTPFilterService serviceInstance =
(IHTTPFilterService)getContext().getAttributes().get(IHTTPF
ilterService.class.getCanonicalName());
        //Get Blocked IP set
        Set<String> hset = serviceInstance.getAllIPAddress();
        Iterator<String> iter = hset.iterator();
        while (iter.hasNext()) {
            //Add blocked IP to list
            l.add(iter.next());
        }
        return l;
    }
    /*For Http Post request, parse posted Json string, add a new
     *IP address to blocked IP set
     */
    @Post
    public String store(String fmJson) {
        IHTTPFilterService serviceInstance =
    (IHTTPFilterService)getContext().getAttributes().get(IHTT
PFilterService.class.getCanonicalName());
        //Create Json tools to parse Json String
        MappingJsonFactory f = new MappingJsonFactory();
        JsonParser jp;

        String ipAddress = null;
        try {
            jp = f.createParser(fmJson);
            jp.nextToken();
            //If Json String is empty, return
            if (jp.getCurrentToken() !=
JsonToken.START_OBJECT) {
                throw new IOException("Expected
START_OBJECT");
            }
            //parse every object in Json string
            while (jp.nextToken() != JsonToken.END_OBJECT)
{
                //If Json format is wrong, return
                if (jp.getCurrentToken() !=
JsonToken.FIELD_NAME) {
                    throw new IOException("Expected
FIELD_NAME");
                }
                //Get Json object key
                String n = jp.getCurrentName();
                jp.nextToken();
                //If Json object value is empty, continue
                if (jp.getText().equals("")) {
                    continue;
                }
                //If Json object key is “ip”, get the value
                if (n.equalsIgnoreCase("ip")) {
                    ipAddress = jp.getText();
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        //If ipAddress is not null, add the IP to blocked IP set
        if (ipAddress != null) {
            serviceInstance.addIPAddress(ipAddress);
        }

        String status = "200";
        return ("{\"status\" : \"" + status + "\", \"ip\" : \""+
ipAddress + "\"}");
    }
    /*For Http Delete request, parse posted Json string, new
     *delete an IP address from blocked IP set
     */
```

```
    @Delete
    public String remove(String fmJson) {
        IHTTPFilterService serviceInstance =
(IHTTPFilterService)getContext().getAttributes().get(IHTTPF
ilterService.class.getCanonicalName());
        //Create Json tools to parse Json String
        MappingJsonFactory f = new MappingJsonFactory();
        JsonParser jp;
        String ipAddress = null;
        try {
            jp = f.createParser(fmJson);
            jp.nextToken();
            //If Json String is empty, return
            if (jp.getCurrentToken() !=
JsonToken.START_OBJECT) { throw new
IOException("Expected START_OBJECT");
              }
            //parse every Json object in Json string
            while (jp.nextToken() != JsonToken.END_OBJECT) {
              //If Json format is wrong, return
              if (jp.getCurrentToken() !=
JsonToken.FIELD_NAME) {
                        throw new IOException("Expected
FIELD_NAME");
                }
              //Get Json object Key
              String n = jp.getCurrentName();
              jp.nextToken();
              //If Json object value is empty, continue
              if (jp.getText().equals("")) {
                continue;
              }
               //If Json object key is “ip”, get the value
              if (n.equalsIgnoreCase("ip")) {
                ipAddress = jp.getText();
              }
            }
        } catch (Exception e) {
              e.printStackTrace();
        }
        //If ip address is not null, delete IP address from Blocked
IP set
        if (ipAddress != null) {
            serviceInstance.deleteIPAddress(ipAddress);
        }

        String status = "200";
        return ("{\"status\" : \"" + status + "\", \"ip\" : \""+
ipAddress + "\"}");
    }
}
```

HTTPFilterRoutable.java

```
package net.floodlightcontroller.httpfilter;

import org.restlet.Context;
import org.restlet.Restlet;
import org.restlet.routing.Router;

import net.floodlightcontroller.restserver.RestletRoutable;
//This class creates REST API URLs for this module
public class HTTPFilterRoutable implements
RestletRoutable {
    //Create specific path for every Resource
    @Override
    public Restlet getRestlet(Context context) {
        // TODO Auto-generated method stub
        Router router = new Router(context);
       //configure specific REST API  path for
HTTPFilterResource
        router.attach("/ip/json", HTTPFilterResource.class);

        return router;
    }

    //Create Base Path for REST API
    @Override
    public String basePath() {
        // TODO Auto-generated method stub
        return "/wm/httpfilter";
    }
}
```

mtd_firewall.py

```
import urllib2
import urllib
import json
import time
import httplib

#PARAMETER
#your server host ip
ip_protect = '10.0.0.8
#sentinel switch '
switch_protect = "00:00:00:00:00:00:00:03"
controller_ip = '192.168.56.102'
#icmp ddos threshold
threshold = 5

icmp_pair = {}
flowSwid = {}
flowPair = {}
ip_dst = ''
ip_src = ''
flowCount =1
blockflag = False
pcnt = 0

#flow* is the flow rule for detecting, blocking and redirecting
traffic flow
flow1 = {
    'switch':str(switch_protect),
    "name":"icmp_beacon",
    "cookie":"0",
    "priority":"2",
    "in_port":"5",
    "eth_type":"0x0800",
    "ip_proto":"0x01",
    "ipv4_src":"10.0.0.1",
    "icmpv4_type":"8",
    "active":"true",
    "actions":"output=4"
}
flow3 = {
    'switch':str(switch_protect),
    "name":"udp_beacon",
    "cookie":"0",
    "priority":"2",
    "in_port":"5",
    "eth_type":"0x0800",
    "ip_proto":"0x11",
    "ipv4_src":"10.0.0.1",
    "active":"true",
    "actions":"output=4"
}
flow4 = {
    'switch':str(switch_protect),
    "name":"tcp_beacon",
    "cookie":"0",
    "priority":"2",
    "in_port":"5",
    "eth_type":"0x0800",
    "ip_proto":"0x06",
    "ipv4_src":"10.0.0.1",
    "active":"true",
    "actions":"output=4"
}
flow7 = {
    'switch':str(switch_protect),
    "name":"http_beacon",
    "cookie":"0",
    "priority":"2",
    "in_port":"5",
    "eth_type":"0x0800",
    "ip_proto":"0x06",
    "tp_dst":"80",
    "ipv4_src":"10.0.0.1",
    "active":"true",
    "actions":"output=4"
}

flow2 = {
    'switch':str(switch_protect),
    "name":"icmp_block",
    "cookie":"0",
    "priority":"2",
    "in_port":"5",
    "eth_type":"0x0800",
    "ip_proto":"0x01",
    "ipv4_src":"10.0.0.1",
    "icmpv4_type":"8",
    "active":"true",
    "actions":"output=1,set_field=eth_dst-
>86:ee:8f:9b:8d:a8,set_field=ipv4_dst->10.0.0.5"
}
flow5 = {
    'switch':str(switch_protect),
    "name":"udp_block",
    "cookie":"0",
    "priority":"2",
    "in_port":"5",
    "eth_type":"0x0800",
    "ip_proto":"0x11",
    "ipv4_src":"10.0.0.1",
    "active":"true",
    "actions":"output=1,set_field=eth_dst-
>86:ee:8f:9b:8d:a8,set_field=ipv4_dst->10.0.0.5"
}
flow6 = {
    'switch':str(switch_protect),
    "name":"tcp_block",
    "cookie":"0",
    "priority":"2",
    "in_port":"5",
    "eth_type":"0x0800",
    "ip_proto":"0x06",
```

```
    "ipv4_src":"10.0.0.1",
    "active":"true",
    "actions":"output=in_port"
}
#static flow pusher
class StaticFlowPusher(object):

    def __init__(self, server):
    self.server = server

    def get(self, data):
    ret = self.rest_call({}, 'GET')
    return json.loads(ret[2])

    def set(self, data):
    ret = self.rest_call(data, 'POST')
    return ret[0] == 200

    def remove(self, objtype, data):
    ret = self.rest_call(data, 'DELETE')
    return ret[0] == 200

    def rest_call(self, data, action):
    path = '/wm/staticflowpusher/json'
    headers = {
            'Content-type': 'application/json',
            'Accept': 'application/json',
            }
    body = json.dumps(data)
    conn = httplib.HTTPConnection(self.server, 8080)
    conn.request(action, path, body, headers)
    response = conn.getresponse()
    ret = (response.status, response.reason, response.read())
    #print ret
    conn.close()
    return ret

pusher = StaticFlowPusher(controller_ip)


#pusher.set(flow1)
#pusher.set(flow2)

#http://192.168.56.102:8080/wm/staticflowpusher/list/00:00:0
0:00:00:00:00:03/flow/json
#http://192.168.56.102:8080/wm/staticflowpusher/list/00:00:0
0:00:00:00:00:03/json
#http://192.168.56.102:8080/wm/core/staticflowpusher/list/00:
00:00:00:00:00:00:03/flow/json
#http://192.168.56.102:8080/wm/core/switch/00:00:00:00:00:
00:00:03/flow/json

#give url and retrieve json object
def get_device_url(url_json):
    #print url_json
    response = urllib2.urlopen(url_json)
    html = response.read()
    # parse response as json
    jsondata = json.loads(html)
    response.close()
    return jsondata

def push_icmp_beacon(flow):
    pusher.set(flow)
#core idea, get json data to identify the icmp pair and by
packetCount, in_port, ip_src and ip_dst
def statDaemon1(d):
    #print switch_protect
    try:
    if str(d['match']['ipv4_dst']) == ip_protect:
            fPKey = str(d['match']['ipv4_src'])+"-
"+str(d['match']['ipv4_dst'])
            if fPKey not in flowPair:
            flowPair[fPKey]=[]
            flowPair[fPKey].append(int(d['packetCount']))
            flowPair[fPKey].append(int(d['match']['in_port']))
            print d['packetCount']+','+d['match']['ipv4_dst']+"\n"
            flowPair[fPKey][0] = int(d['packetCount'])
            #print "haha"+str(data)
            flowSwid[switch_protect] = flowPair
    except:
    pass
def statDaemon(d):
    #print switch_protect
    try:
    if str(d['match']['ipv4_dst']) == ip_protect:
            fPKey = str(d['match']['ipv4_src'])+"-
"+str(d['match']['ipv4_dst'])
            if fPKey not in flowPair:
            flowPair[fPKey]=[]
            flowPair[fPKey].append(int(d['packetCount']))
            flowPair[fPKey].append(int(d['match']['in_port']))
            print d['packetCount']+"\n"
            flowPair[fPKey][0] = int(d['packetCount'])
            #print "haha"+str(data)
            flowSwid[switch_protect] = flowPair
    except:
    pass

#curl -X POST
http://192.168.1.68:8080/wm/statistics/config/enable/json
#curl -X GET
http://192.168.1.68:8080/wm/statistics/bandwidth/00:00:00:00
:00:00:00:02/3/json
#curl -X GET
http://192.168.1.68:8080/wm/core/switch/all/flow/json
#curl -X GET
http://192.168.1.68:8080/wm/core/switch/00:00:00:00:00:00:0
0:02/flow/json
#curl -X GET
http://192.168.1.68:8080/wm/core/switch/00:00:00:00:00:00:0
0:02/aggregate/json
#curl -X GET
http://192.168.1.68:8080/wm/core/switch/all/aggregate/json
```

```
#curl -X GET
http://192.168.1.68:8080/wm/core/switch/all/port/json
#curl -X GET
http://192.168.1.68:8080/wm/staticflowpusher/list/00:00:00:0
0:00:00:00:02/json

initflag = True
devicePair=[]
url_json = "http://"+controller_ip+":8080/wm/device/"
jsondata = get_device_url(url_json)
for p in jsondata:
    devicePair.append([p['mac'],p['ipv4'],0])
#print devicePair
push_icmp_beacon(flow1)
push_icmp_beacon(flow3)
#push_icmp_beacon(flow4)
push_icmp_beacon(flow7)

#check the flow with 5 sec interval. if the packetCount >
threshold, block the icmp packet with the ip_src or redirect to
our bait server
while 1:

    try:
    url_json =
"http://"+controller_ip+":8080/wm/core/switch/"+switch_prot
ect+"/flow/json"
    jsondata = get_device_url(url_json)

    for data in jsondata:
            #print data
            for d in jsondata[data]:
            #print str(d)+"\n"
            if len(d['match']) >0: statDaemon1(d)
    print flowSwid#[switch_protect]
    except:
    pass
    #url_json =
"http://"+controller_ip+":8080/wm/core/switch/"+switch_prot
ect+"/flow/json"
    #jsondata = get_device_url(url_json)

   # print jsondata
    for i in jsondata["flows"]:
    #print i['priority']
    if i['priority'] == '2':
            if int(i['packetCount']) > threshold:
            print "icmp_ddos_detect"
            pusher.set(flow2)
            pusher.set(flow5)
            pusher.set(flow6)


    time.sleep(5)
```

topo_project.py

```
#add 8 hosts and 4 switch for demo
from mininet.topo import Topo
class MyTopo( Topo ):
    "Simple topology example."
    def __init__( self ):
    "Create custom topo."
    # Initialize topology
    Topo.__init__( self )
    # Add hosts and switches
    leftHost = self.addHost( 'h3', ip='10.0.0.3' )
    rightHost = self.addHost( 'h4', ip='10.0.0.4' )
    badHost1 = self.addHost('h1', ip='10.0.0.1')
    badHost2 = self.addHost('h2', ip='10.0.0.2')
    #NODE2_IP = '192.168.56.103'

    backHost1 = self.addHost('h5', ip='10.0.0.5')
    backHost2 = self.addHost('h6', ip='10.0.0.6')

    targetHost = self.addHost('h7', ip='10.0.0.7')
    fake_listener = self.addHost('h8', ip='10.0.0.8')

    #midHost = self.addHost( 'h3' )
    #midHost2 = self.addHost('h4')
    leftSwitch = self.addSwitch( 's1' )
    rightSwitch = self.addSwitch('s2')
    backupSwitch = self.addSwitch('s3')
    #rightSwitch = self.addSwitch( 's2')
    # Add links
    self.addLink( leftSwitch, leftHost )
    self.addLink( leftSwitch, rightHost )
    self.addLink( leftSwitch, badHost1)
    self.addLink( leftSwitch, badHost2)

    self.addLink( backupSwitch, backHost1)
    self.addLink( backupSwitch, backHost2)

    self.addLink(rightSwitch, targetHost)
    self.addLink(rightSwitch, fake_listener)


    self.addLink(leftSwitch, rightSwitch)
    self.addLink(rightSwitch, backupSwitch)
    self.addLink(leftSwitch, backupSwitch)
#leftSwitch.cmd('ovs-vsctl add-port s1 s1-gre1 -- set interface
s1-gre1 type=gre options:remote_ip='+NODE2_IP)
    #self.addLink( leftSwitch, rightSwitch )
    #self.addLink( rightSwitch, midHost)
    #self.addLink(rightSwitch, midHost2)
topos = { 'mytopo': ( lambda: MyTopo() ) }
```

filter.py

```
import requests
import json
import subprocess

flows = {'keys':'ipsource,ipdestination','value':'bytes','filter':''}
threshold = {'metric':'flows','value':10000}

target = 'http://localhost:8008'

r = requests.put(target +
'/flow/flows/json',data=json.dumps(flows))

r = requests.put(target +
'/threshold/flows/json',data=json.dumps(threshold))

eventurl = target + '/events/json?maxEvents=10&timeout=6'
eventID = -1

while 1 == 1:
  r = requests.get(eventurl + '&eventID=' + str(eventID))
  if r.status_code != 200: break
  events = r.json()
  if len(events) == 0: continue

  eventID = events[0]["eventID"]
  result = []
  for i in events:
    if 'flows' == i['metric']:
      r = requests.get(target + '/metric/' + i['agent'] + '/' +
i['metric'] + '/json')
      metric = r.json()
      if len(metric) > 0:
        #print metric[0]["topKeys"][0]["key"]
       result.append(metric[0]["topKeys"][0]["key"])
  splitIP = result[0].split(",")
  acl = 'curl -X POST -d \'{\"src-ip\":\"'+splitIP[0]+'/32\",\"dst-
ip\":\"'+splitIP[1]+'/32\",\"action\":\"deny\"}\' http://127.0.0.1:
8080/wm/acl/rules/json'
  subprocess.Popen(acl, shell=True, stdout=subprocess.PIPE)
  #output = subprocess.check_output(['bash','-c', acl])
  #print (output)
```

hi.py

```
import subprocess
import requests
import json

while 1 == 1:
    cmd = raw_input('')
    if cmd == 'reset':
    resetcmd = 'curl http://127.0.0.1:8080/wm/acl/clear/json'
    subprocess.Popen(resetcmd, shell=True)
    #output = subprocess.check_output(['bash','-c', resetcmd])
    print ('all rules deleted')

    elif cmd == 'list':
    listcmd = 'curl http://127.0.0.1:8080/wm/acl/rules/json |
python -mjson.tool'
    subprocess.Popen(listcmd, shell=True)
    #output = subprocess.check_output(['bash','-c', listcmd])
    print ('restricted')

    elif cmd == 'delete rule':
    uid = raw_input('')
    removecmd = 'curl -X DELETE -d
\'{\"ruleid\":\"'+uid+'\"}\' http://127.0.0.1:8080/wm/acl/rules/js
on'
    subprocess.Popen(removecmd, shell=True)
    print ('rule deleted')
```