UNIVERSITY OF ALBERTA

CAPSTONE PROJECT
DDoS Detection And Mitigation Using OpenFlow

Rahul Kumar, MINT

Supervisor, Ali Tizghadan, Telus

ABSTRACT

The solution is addressing the need of implementing DDoS mitigation at the network level without necessarily employing dedicated hardware but rather using the router platforms as efficient mitigation devices for certain clearly defined attack vectors. It also facilitates the off-ramping of the suspicious traffic to specialized security gear for more in-depth analysis and enforcement.

CONTENTS

ACRONYMS

SDN Software Defined Network
DDoS distributed denial-of-service
UDP User Datagram packet
ICMP Internet control message protocol
TCP Transmission control protocol
IP Internet protocol
MPLS Multiprotocol Label switching
DPID Datapath ID
REST Representational state transfer
API Application protocol interface
VM Virtual machine
SNMP Simple network monitoring protocol

INTRODUCTION

DDoS is an attack that scales from large to small enterprises publishing their services publically. Though nobody has been able to completely mitigate the attack without service degradation but large enterprise do deploy solution to restrict the impact in their network while segregating rogue request from legitimated ones. The service provider network is another aspect of DDoS attack, where traffic for customer can choke low bandwidth interfaces of service provider infrastructure.

Some of these enterprises are able to mitigate DDoS but still all the bandwidth is utilized by DDoS traffic. They are able to protect their servers and services from DDoS traffic but still all the legitimate users are unable to use it. In addition, DDoS attack affects service provider network as well. It is a significant issue for both the customer and provider.

In the case of service provider network, the responsibility extends to protecting its own network from over consumption and detecting affected customer. To protect their network and customers they need to detect and mitigate at their end. When these attacks occurs, the effective way to block them is to manually restrict the published server IP traffic. This method is the last resort, which requires timely response and threat detection.

SOFTWARE DEFINED NETWORK

The last decade seen great many changes in server characteristics as 'Virtualization' became prominent part of server and application [1]. The concept of server also evolved from centralized to distributed application and services. These application and services made easy for server administrator to install, update, manage and move virtual systems within and across the datacenters. On the contrary, the same level of control was required from network infrastructure to centrally understand the distributed computation and provide discrete service based on application requirement. The virtualization pushed vendors to develop new technologies and protocols to provide dynamic communication between server irrespective of network physical and logical capabilities.

Software Defined Network (SDN) a new term brought hope for evolution in the architecture. SDN holds the future for new ideas, protocols, better control & management, policing and flexible enough to support legacy components as well as making it face new challenges and error-prone [2]. Network engineers were always restricted in their approach to perform operation with limited set of tools. The implication of programming on top of physical hardware and underlying operating system the SDN has extended its arm to fine-tune network behavior with respect to application and services.

The fundamentals of SDN includes separation of control and data plane [3]. The data plane includes the forwarding logic which when receives the packet deals with it based on actions such as forward, drop, reject, replicate or perform vendor proprietary

functions. All algorithm that decides and calculate what need to be done on the incoming packet takes place in control plane. Once the decision is made in control plane it is installed in the form of flow into the forwarding table. The purpose of control plane is to maintain state of the network and synchronization with other devices on the network. In the case of SDN the control plane is lifted from the devices making them dump forwarding stations, all synchronized to a central software based controller which provides instruction allowing them to perform actions on incoming traffic flows. The software based controller provides a central view to programmer and network engineer to control forwarding behavior across network through the open interface controlled by north and south bound APIs.

OPENFLOW

OpenFlow [4] was developed as a research protocol and became part of Stanford University research network. The basic scope of this protocol is to provide a ground to new protocol that can work without the boundaries of layer 2 and layer 3 protocols, thus reshaping the communication. After it's initial result the Open Networking Foundation (ONF) promoted this protocol stack so that service provider networks can test this protocol in real environment.

The Openflow protocol follows a client server architecture where an agent is installed in the switches to connect to the software based controlled. The controller provides an openflow interface to communicate with the switches through the use of APIs. It is important to understand here that controller function is nothing without the use application of API. The open flow protocol itself is divided into two parts:

- The wireline protocol is used to define the flow exchange and action communicated to switch. This also provide pipelining for every table, in the case multiple table deployment.
- The second part uses NETCONF for the management of switches through physical ports. It also provides other services such as maintaining redundant connection for high availability and checking connection to the remote controller.

The Open flow protocol can be deployed in proactive mode where flow are installed before the incoming packet or a reactive approach where flows are installed one the data plane receives the incoming packet.

DDOS

A distributed denial-of-service (DDoS) attack is one in which a large number of compromised systems attack a single target, thereby exhausting all resources of the target system [5]. The flood of incoming messages to the target system essentially forces it to shut down, thereby denying service to legitimate users.

Attackers build networks of infected computers, known as 'botnets', by spreading malicious software through emails, websites and social media. These bots can be controlled remotely and used to launch an attack against any target without the knowledge of its owner (machine owner). Some botnets are millions of machines strong. Botnets can generate huge floods of traffic to crash a target system. These floods can be generated in multiple ways, such as sending more connection requests than a server can handle, or sending huge amounts of random data to use up the targets bandwidth. There are specialized online marketplaces to buy and sell botnets. Anyone can pay a nominal fee to attack websites or disrupt an organization's online operations.

Types of DDoS

UDP Flood

This DDoS attack leverages the User Datagram Protocol (UDP), a sessionless networking protocol. In this type of attack numerous UDP packets are flooded to random ports on remote host, causing the host to repeatedly check for the application listening at that port, and (when no application is found) reply with an ICMP Destination Unreachable packet. This process drains host resources, and can ultimately lead to inaccessibility.

ICMP Flood

It is similar to the UDP flood attack. An ICMP flood overwhelms the target resource with ICMP Echo Request packets, generally sending packets as fast as possible without waiting for replies. ICMP attack can consume both outgoing and incoming bandwidth, as the target system will often attempt to respond with ICMP Echo Reply packets, resulting a significant overall system slowdown.

SYN Flood

A "three-way handshake", which is a reference to how TCP connections work, is the basis for this form of attack. Wherein a SYN request to initiate a TCP connection with a host must be answered by a SYN-ACK response from that host, and then confirmed by an ACK response from the requester. In a SYN flood, the requester sends multiple SYN requests, but either does not respond to the host's SYN-ACK response, or sends the SYN requests from a spoofed IP address. In any case, the host system continues to wait for acknowledgement for each of the requests, binding resources until no new connections can be made, and ultimately resulting in denial of service.

Ping of Death

A ping of death attack involves the attacker sending multiple malformed or malicious pings to target system. The maximum packet length of an IP packet is 65,535 bytes. But Data Link Layer usually poses limits to the maximum frame size of 1500 bytes over an Ethernet network. So large IP packet is split across multiple IP packets known as fragments, and the target system reassembles the IP fragments into the complete packet. In a Ping of Death, following malicious manipulation of fragment content, the recipient ends up with an IP packet, which is larger than 65,535 bytes when reassembled. This can overflow memory buffers allocated for the packet, causing denial of service for legitimate packets.

Reflected Attack

A reflected attack is where an attacker creates packets with source address of target system that will be sent out to as many computers as possible. When these computers receive the packets they will reply, but the reply will be a spoofed address that actually routes to the target. All of the computers will attempt to communicate at once and this will cause the target system to be bogged down with requests until its resources are exhausted.

Peer-to-Peer Attacks

In this type of attack peer-to-peer server is exploited to route traffic to the target system instead of using botnet. So peer-to-peer clients using file-sharing hub instead send data to target system until it is overwhelmed and sent offline.

SYSTEM DESIGN APPROACH

We propose an automated solution to mitigate DoS attacks which operates at the network level via software-defined networking (SDN). DDoS traffic is detected and notified to a centralized controller, which can adaptively adjust peering routers to filter the DDoS traffic using Openflow. The centralized approach of triggering the security action from just one place (the SDN controller) will reduce the time that would be spent otherwise implementing equivalent measures on each and every router. Also, with the right policies in place on the SDN controller automatically triggering the security actions based on various conditions, manual labor could be avoided.

By operating at the network layer, we are able to stop the traffic right at the edge of our

domain, which minimizes the impact of the attack across our network.

As a proof of concept, we design and implement our solution in the Telus SDN testbed lab. We demonstrate how Openflow-enabled switches can be managed via a controller to filter DDoS traffic.

- The goal is to demonstrate that the IP/MPLS Core network SDN deployment can be leveraged as part of the network-level security mitigation;
- In the first phase, the assumption is that the security event detection is performed by a separate system:
    - e.g. for DDoS attacks the system used is the Core Arbor Peakflow deployment;
    - other systems may be used to detect the presence of malware, external feeds with threat intelligence, etc.
- In subsequent phases, we will explore leveraging SDN for the detection portion as well;
- The security event detection (DDoS attack) is performed by the system Core PeakFlow SP collecting and analysing Netflow data from routers;
- Alert trigger containing the 5-tuple (or parts of it) information is sent to the Alert Listener via syslog messages;
- SDN controller issues one of the following commands to the routers (via Openflow):
    - Drop/block (depicted in Figure 1);
    - Steer (off-ramp) towards a security device directly attached to the Openflow router defined destination (e.g. Security Pod TMS, sandbox, etc.). Figure 2 illustrates this process.

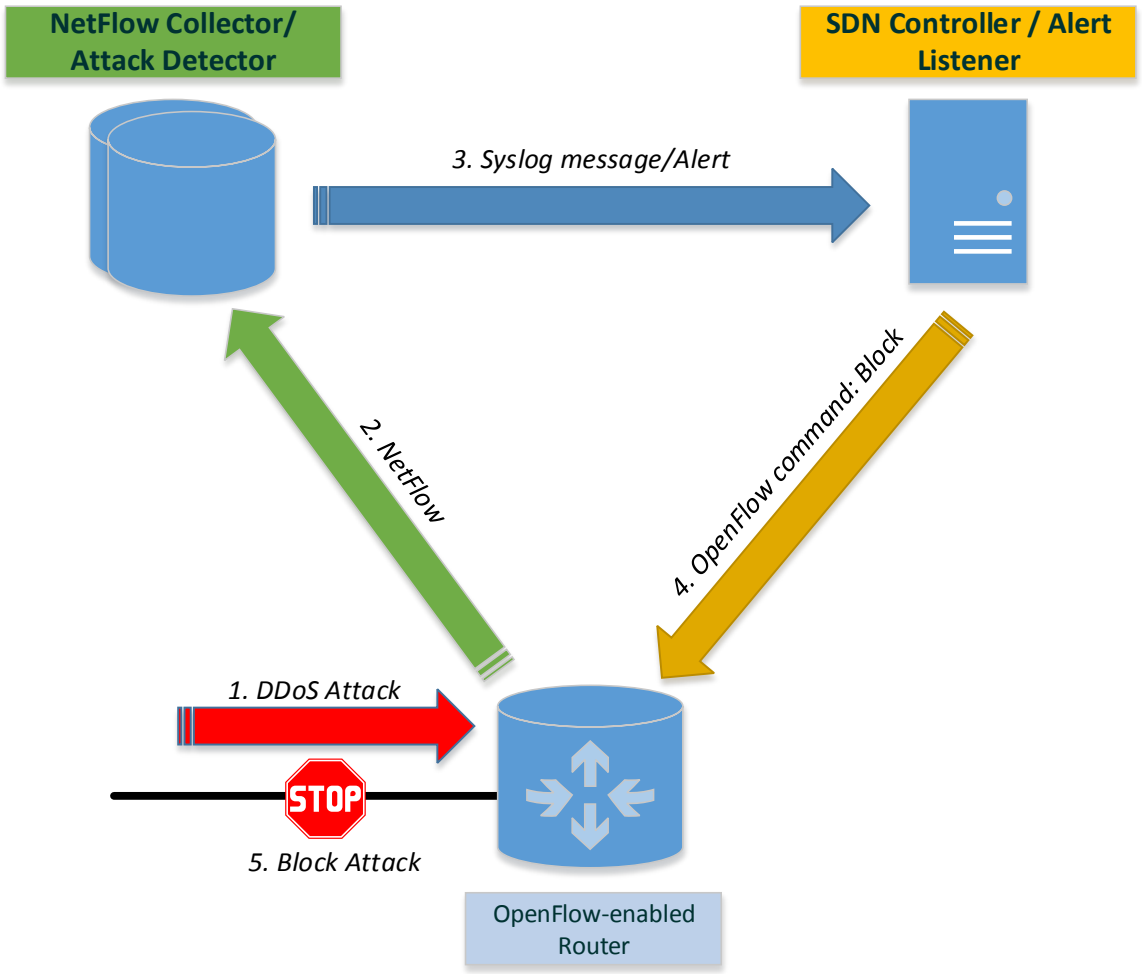

Figure 1. Block Action

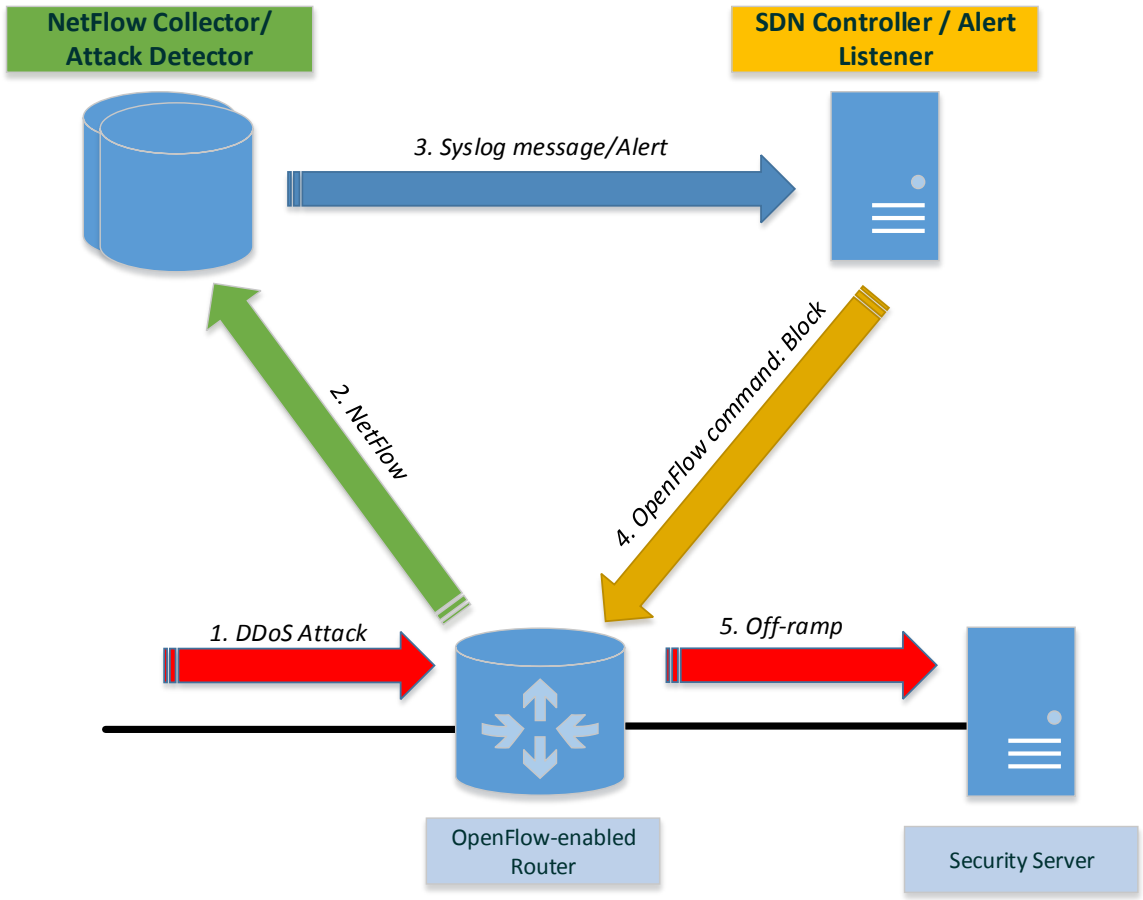

Figure 2. Off-ramp action

SYSTEM COMPONENTS

There are two applications deployed in two different virtual machines. The Alert Listener is written in python while Opendaylight is in Java. Arbor PeakFlow is a third component that is responsible for DDoS detection.

Alert Listener

The purpose of this application is to receive DoS alert syslog from Arbor Peakflow SP on a pre-defined IP address & port. It parses the alert to get information about type of DoS attack, alert id, status, source IP address, destination IP address, protocol, port and affected router name. It is not necessary that alert will contain all the information as it depends on Arbor detection system. First, it checks if parsed alert id is already in the database. If not then based on the information it generates a rule, which contains priority, unique flow id, action (DROP) and switch DPID (each affected router has unique switch DPID). Switch DPID is taken from a text file, which contains unique switch DPID for

each router. This file is created manually. Once rule is formed it uses REST API to send it to Opendaylight controller and stores the flow id along with alert id and switch DPID. Opendaylight controller stores this flow id, which is used in removing the flow later on.

In case alert id already exists in alert listener database it will check the status of alert. If status is done, then it generates a rule to delete the corresponding flow from the affected router. It sends the rule to Opendaylight controller using REST API.

Opendaylight Controller

We are using Opendaylight controller only to push/delete flows from connected openflow router (Juniper MX960). We have developed a java application (default_flow.jar), which runs on Opendaylight controller to install default flow to connected openflow router. The purpose of this default flow is to forward traffic to control plane of openflow router. As by default all the traffic is sent to controller if no matching flow is found. So in order to keep routing decision at openflow router control plane this flow is installed.

The application listens for data packets sent by connected openflow router. Openflow router registers itself with Opendaylight controller and start sending traffic, which it receives, on openflow interface. As soon as first packet is received by the application it pushes the default flow to the openflow router. As a result all traffic after that is handled by openflow router control plane.

Arbor Peakflow SP

It is used to detect DoS attack on juniper MX960. It collects statistics of openflow interface using JFLOW (Netflow) after pre-defined interval of time. The collected statistics is analyzed and compared with the pre-defined threshold value for different type of traffic by Arbor to detect DoS attack and details of attack i.e. type of DoS attack, destination IP address, port, protocol and source IP address. Once DoS attack is detected it sends syslog alert to Alert listener on pre-defined IP address and port with details of the DoS attack.

SYSTEM IMPLEMENTATION

STEP 1: Installation of default flow application in Opendaylight controller.

To install default_flow.jar application we use “install file:/path” command. After installation is finished the application receives a bundle ID, which in our case happened to be 253. The ID number will be defined in the command line output after installation. The procedure is depicted in Figure 3.

Figure 3. Default flow installation

In order to start/run the bundle we simply issue a “start” command followed by bundle ID number.

For verification, we execute “ss” command with the name of our bundle – “default”. The output (Figure 4) indicates the bundle name and its state as active.

```
osgi> ss default
"Framework is launched."


id      State       Bundle
253     ACTIVE      org.opendaylight.default_flow_0.1.0
osgi>
```

Figure 4. Starting the application

STEP 2: Configure Openflow in Juniper MX960.

A set of commands [6] needs to be executed in order to enable OpenFlow functionality in the edge router MX960.

```
{master}
sdnlab@bluemoon> show configuration protocols openflow
switch OF-SW1 {
    default-action {
        packet-in;
    }
    interfaces {
        xe-4/3/0.0;
        xe-4/1/0.0;
    }
    controller {
        protocol {
            tcp {
                port 6633;
            }
        }
        address 10.0.30.3;
    }
}
traceoptions {
    file openflow.log;
    flag all;
}
```

Figure 5. MX960 OpenFlow configuration

Default action should be “packet-in”. This means any packet/traffic on openflow interface, which doesn’t matches any flow, will be sent to controller.

In order to verify resulting configuration, “show openflow switch” command is issued.

```
{master}
sdnlab@bluemoon> show openflow switch
Switch Name:        OF-SW1
Switch ID:          0                   Switch DPID:    00:00:84:18:88:09:2f:c0
Flow mod received:  2                   Vendor received:      0
Packets sent:       542                 Packets received:     556
Echo req sent:      1                   Echo req received:    0
Echo reply sent:    0                   Echo reply received:  1
Port Status sent:   0                   Port mod received:    0
Barrier request:    0                   Barrier reply:        0
Error msg sent:     4                   Error msg received:   0
```

Figure 6. OpenFlow functionality verification

For the purpose of later communication, the Switch DPID is stored in *mapping.txt* along with router name. Controller connection status is verified by the output of “show OpenFlow controller” command, depicted in Figure 7.

```
{master}
sdnlab@bluemoon> show openflow controller
Openflowd controller information:
Controller socket: 14
Controller IP address: 10.0.30.3
Controller protocol: tcp
Controller port: 6633
Controller connection state: up
Number of connection attempt: 96166
Controller role: equal
```

Figure 7. Controller connection status

STEP 3: Configure Jflow in Juniper MX960

Jflow is a Juniper implementation of NetFlow and is essentially identical to original protocol. NetFlow protocol is used as a basis for threat detection providing valuable flow information about all source-to-destination communications. The generated statistics is collected and sent to other components of the system for analysis.

Required configuration commands are defined in Juniper documentation [7].

STEP 4: Run Alert Listener application

To run Alert listener application we use “python alert_listener.py” command on Ubuntu terminal.

```
savi@savi-vm1:~/project$ python alert_listener.py
2015-04-13 17:32:06,766 - Alert Listener - INFO - Alert Listener started at 10001
```

Figure 8. Alert Listener execution

SYSTEM WORKFLOW

After forwarding device (Juniper MX960) connects to the controller its flow table will be empty. As soon as first packet reaches an interface of the forwarding device it will be redirected to the controller (depicted in Figure 9). At this point default_flow application will receive the original packet and install a default flow in the flow table of the forwarding device, as it is depicted on the screenshot (OFPP_NORMAL = 65530).

```
{master}
sdnlab@bluemoon> show openflow flows detail

{master}
sdnlab@bluemoon> show openflow flows detail
Flow name: flow-83951616
Table ID: 1      Flow ID: 83951616
Priority: 0        Idle timeout(in sec):0        Hard timeout(in sec): 0
Match: Input port: wildcard
       Ethernet src addr: wildcard
       Ethernet dst addr: wildcard
       Input vlan id: wildcard          Input VLAN priority: wildcard
       Ether type: 0x800
       IP ToS: wildcard                 IP protocol: wildcard
       IP src addr: wildcard            IP dst addr: wildcard
       Source port: wildcard            Destination port: wildcard
Action: Output port 65530,

{master}
sdnlab@bluemoon>
```

Figure 9. MX960 Flows information

This default flow will be injected in the flow table with the lowest priority set. The purpose of this flow entry is to redirect all subsequent traffic to the control plane of the forwarding device. For every incoming packet the flow table is being checked first and if there are no other matching flows entries (with higher priority) all traffic will be handled by local control entity.

This is the original state of the forwarding plane unless it has been altered for DoS mitigation purposes.

Alert Listener is an independent software module interprets alert notifications (depicted in Figure 10) generated by security mitigation device (Arbor PeakFlow) and creates a corresponding flow messages. Each flow message contains a specific flow rule to block or divert illegitimate traffic. Alert Listener's messages are then pushed to the controller via REST API. As an abstract layer of forwarding plane controller is responsible for injection of flows into forwarding tables of network devices.

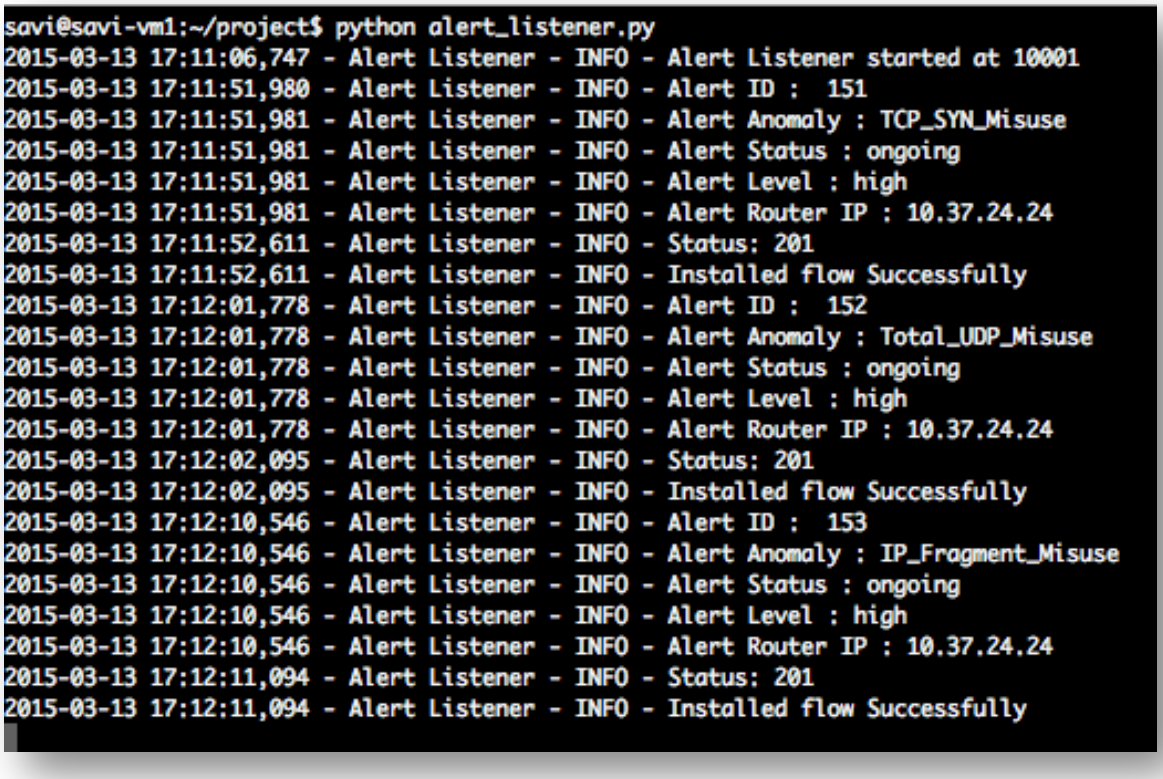

Figure 10. Alert information

All generated flows are saved in local database of Alert Listener and will remain in active status unless it receives a new notifications from Arbor PeakFlow. Every alert notification is compared to the database and may represent a new alert, status change or alert over. If the alert state changes to "*done*", Alert Listener will generate a flow

message to remove the flow from forwarding table of corresponding network device.

```
osgi> 2015-03-13 17:11:50.852 EDT [http-bio-8080-exec-1] INFO  o.o.c.u.internal.UserManager - Local Authentication Succeeded for User: "admin"
2015-03-13 17:11:50.853 EDT [http-bio-8080-exec-1] INFO  o.o.c.u.internal.UserManager - User "admin" authorized for the following role(s): [Network-Admin]
2015-03-13 17:12:00.598 EDT [http-bio-8080-exec-3] INFO  o.o.c.u.internal.UserManager - Local Authentication Succeeded for User: "admin"
2015-03-13 17:12:00.598 EDT [http-bio-8080-exec-3] INFO  o.o.c.u.internal.UserManager - User "admin" authorized for the following role(s): [Network-Admin]
2015-03-13 17:12:09.364 EDT [http-bio-8080-exec-5] INFO  o.o.c.u.internal.UserManager - Local Authentication Succeeded for User: "admin"
2015-03-13 17:12:09.365 EDT [http-bio-8080-exec-5] INFO  o.o.c.u.internal.UserManager - User "admin" authorized for the following role(s): [Network-Admin]
```

Figure 11. REST API Authentication

For security purposes REST API of the Controller has authentication policy in place and every received message have to be authenticated. The output in Figure 11 shows that "Admin" user credentials were used by Alert Listener to send a flow messages to the controller.

```
sdnlab@bluemoon> show openflow flows detail
Flow name: flow-100728832
Table ID: 1     Flow ID: 100728832
Priority: 1000    Idle timeout(in sec):0       Hard timeout(in sec): 0
Match: Input port: wildcard
       Ethernet src addr: wildcard
       Ethernet dst addr: wildcard
       Input vlan id: wildcard          Input VLAN priority: wildcard
       Ether type: 0x800
       IP ToS: wildcard                 IP protocol: 0x6
       IP src addr: wildcard            IP dst addr: 203.87.42.56/32
       Source port: wildcard            Destination port: 80
Action: Drop,

Flow name: flow-117506048
Table ID: 1     Flow ID: 117506048
Priority: 1001    Idle timeout(in sec):0       Hard timeout(in sec): 0
Match: Input port: wildcard
       Ethernet src addr: wildcard
       Ethernet dst addr: wildcard
       Input vlan id: wildcard          Input VLAN priority: wildcard
       Ether type: 0x800
       IP ToS: wildcard                 IP protocol: 0x11
       IP src addr: wildcard            IP dst addr: 75.88.66.44/32
       Source port: wildcard            Destination port: wildcard
Action: Drop,

Flow name: flow-134283264
Table ID: 1     Flow ID: 134283264
Priority: 1002    Idle timeout(in sec):0       Hard timeout(in sec): 0
Match: Input port: wildcard
       Ethernet src addr: wildcard
       Ethernet dst addr: wildcard
       Input vlan id: wildcard          Input VLAN priority: wildcard
       Ether type: 0x800
       IP ToS: wildcard                 IP protocol: wildcard
       IP src addr: wildcard            IP dst addr: 144.65.43.87/32
       Source port: wildcard            Destination port: wildcard
Action: Drop,

Flow name: flow-151060480
Table ID: 1     Flow ID: 151060480
Priority: 0       Idle timeout(in sec):0       Hard timeout(in sec): 0
Match: Input port: wildcard
       Ethernet src addr: wildcard
       Ethernet dst addr: wildcard
       Input vlan id: wildcard          Input VLAN priority: wildcard
       Ether type: 0x800
       IP ToS: wildcard                 IP protocol: wildcard
       IP src addr: wildcard            IP dst addr: wildcard
       Source port: wildcard            Destination port: wildcard
Action: Output port 65530,
```

Figure 12. MX960 Installed flows information

After alert is generated, the corresponding flow entry is installed in the forwarding table of network device (shown in Figure 12). The flow with higher priority will be first to check. In case all fields match:

- Protocol
- Destination IP Address
- Destination Port

The defined action is executed and in our case the packet is dropped. All other incoming packets are matched over a default flow and redirected to local control plane of network device, which handles these packets. In that way, for every alert, a specific flow is installed or removed depending from the instructions provided by Alert Listener Application.

LAB SETUP

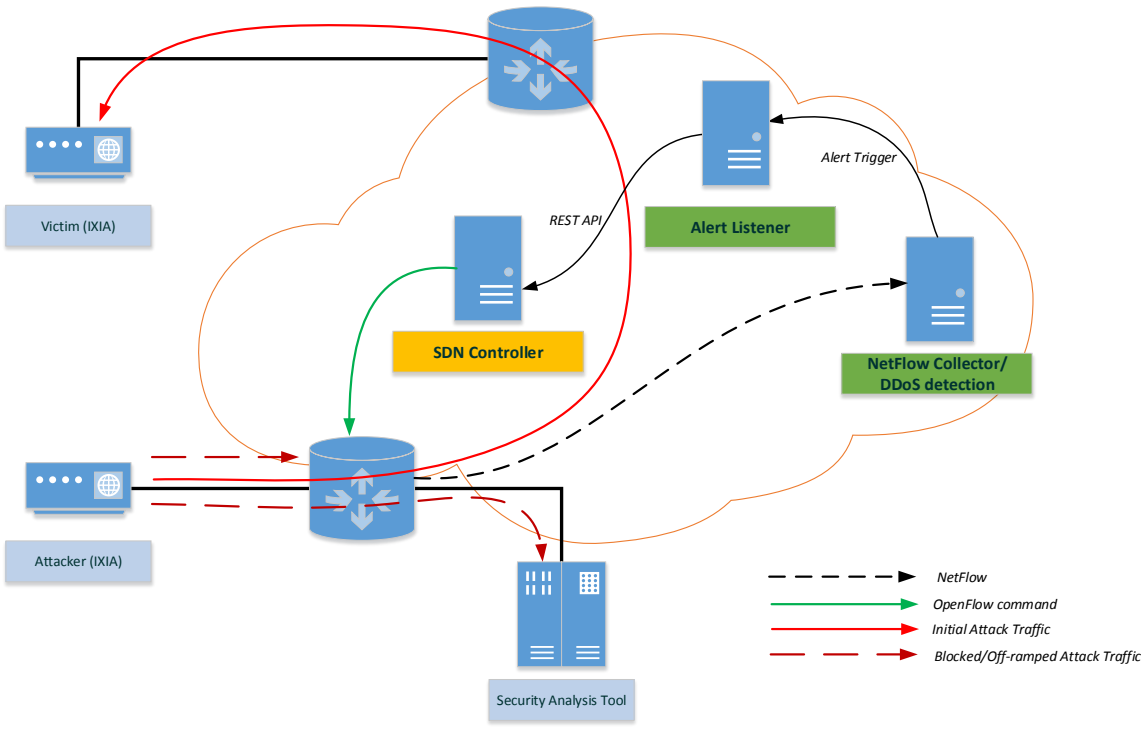

Figure 13. Lab topology

TEST CASES

Test Case 1

Objective

Connecting Juniper MX960 to Opendaylight and installing flow to forward traffic to router control plane.

Source

- Name: Ixia Attacker
- IP: 10.x.x.x
- Interface: Card 12 port 3
- Traffic: Normal
- Protocol: ICMP

Destination

- Name: Ixia Victim
- IP: 10.x.x.x
- Interface: Card 12 port 5
- Protocol: ICMP

Openflow Router

- Hostname: Bluemoon
- Model: Juniper MX960
- OS: JUNOS 13.3R1.6 with SDN software suite
- Openflow version: 1.0
- Switch DPID: 00:00:84:18:88:09:2f:c0
- Openflow Interface (Incoming): xe-4/3/0.0
- Non-openflow Interface (Outgoing): xe-4/2/3

Controller

- Name: Opendaylight

- Version: Hydrogen Base Edition

Platform: Ubuntu 12.04.3 LTS Desktop VM

- IP: 10.0.30.3
- Port: 6633
- Bundle: default_flow.jar

Traffic is sent from Ixia (Attacker) to Ixia (Victim) through MX960 Openflow interface xe-4/3/0.0. As soon as first packet reaches Openflow interface it is sent to the controller as there is no flow installed on MX960. The controller will receive the packet and install the default flow with lowest priority and action (Output port = 65530) in MX960. This default flow installed will match each incoming packet and forward the packet to the router's control plane.

In this test we have installed a default flow, which will forward each packet to router control plane. The route to Ixia (Victim) exists in the router routing table. So after the rule is installed packets from Ixia (Attacker) can reach Ixia (Victim) based on router control plane.

Verification

Connection between MX960 & Opendaylight Controller

The log on Opendaylight controller can verify MX960 connection to Opendaylight controller. The logs will show Openflow Switch DPID (depicted in Figure 14).

Figure 14. Controller logging information

Flow installation

First we run "show openflow flows detail" command on juniper MX960 to verify there are no flows installed (depicted in Figure 15). Then we will send ICMP traffic from Ixia (Attacker) to Ixia (victim). The traffic will be sent to Opendaylight controller.

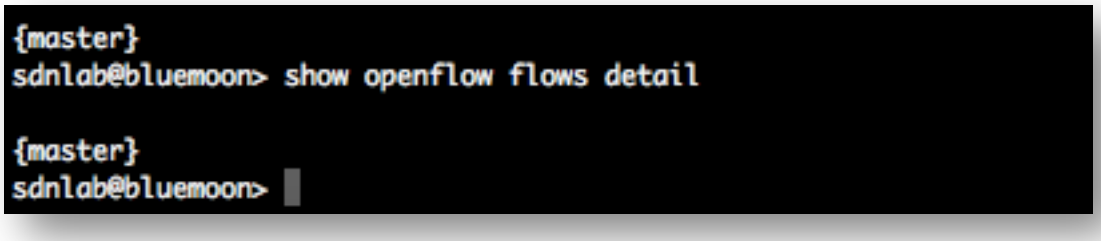

Figure 15. MX960 flow information

Now, we will deploy Default_flow.jar in Opendaylight controller. Again send ICMP traffic and run "show openflow flows detail" command on juniper MX960. A new flow is installed with all values set to wildcard (means any) except priority 0 and action output port 65530.Traffic will reach Ixia (Victim).

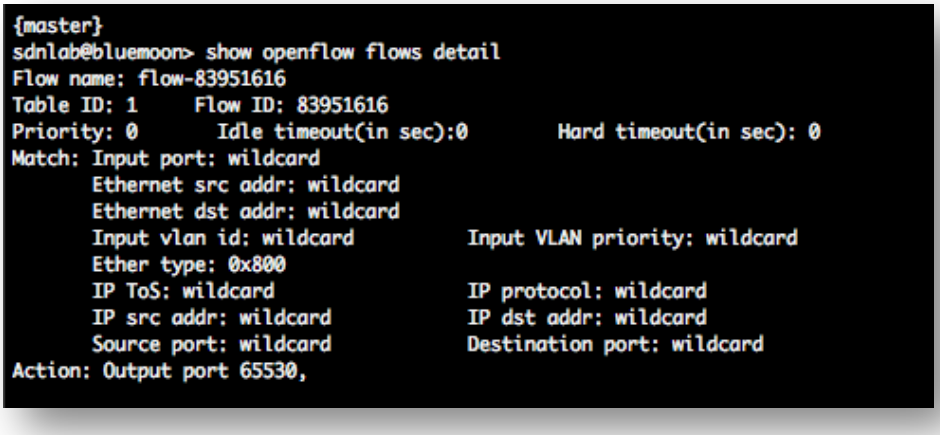

Figure 16. MX960 flow information (new flow installed)

Test Case 2

Objective

Detection of DoS attack on Juniper MX960 (Openflow interface) and mitigation by installing flow to drop DoS traffic.

Detection

- Name: Arbor Peakflow SP
- Protocol: Jflow

Alert Listener

- Name: Alert_listener
- IP: 10.0.30.2
- Port: 10000
- Platform: Ubuntu 12.04.3 LTS Desktop VM
- Programming Language: Python

Source

- Name: Ixia Attacker
- IP: 10.x.x.x
- Interface: Card 12 port 3
- Traffic: DoS
- Protocol: ICMP/TCP

Destination

- Name: Ixia Victim
- IP: 10.x.x.x
- Interface: Card 12 port 5
- Protocol: ICMP

Openflow Router

- Hostname: Bluemoon
- Model: Juniper MX960
- OS: JUNOS 13.3R1.6
- Openflow version: 1.0
- Switch DPID: 00:00:84:18:88:09:2f:c0
- Openflow Interface (Incoming): xe-4/3/0.0
- Non-openflow Interface (Outgoing): xe-4/2/3

Controller

- Name: Opendaylight
- Version: Hydrogen Base Edition
- Platform: Ubuntu 12.04.3 LTS Desktop VM
- IP: 10.0.30.3
- Port: 6633
- Bundle: Default_flow.jar

DoS traffic is sent from Ixia (Attacker) to Ixia (Victim) through MX960 Openflow interface xe-4/3/0.0. MX960 Jflow collects the flow information (Openflow interface) and sends it to Arbor system for analysis. Arbor detects DDoS attack and generates a corresponding syslog alert for Alert Listener application. Alert Listener parses the alert and generate corresponding drop flow. It sends flow to Opendaylight controller using REST API. Opendaylight controller installs the flow in MX960.

Once the drop flow is installed all the DoS traffic matching the flow will be dropped.

Verification

Detection

The log on alert _listener will verify that DoS attack is detected and it has received syslog alert containing details of the attack from Arbor peakflow SP.

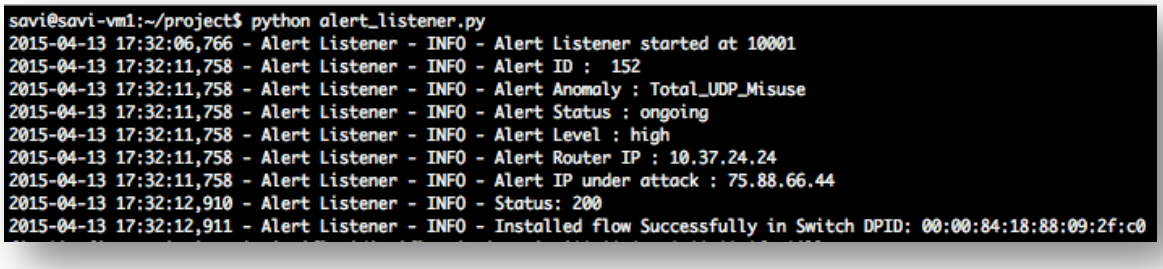

Figure 17. Alert Listener Log

Mitigation

The logs on Opendaylight controller will verify that it has received request. Secondly, it will be reflected in the output of "Show openflow flows details" command in MX960. A new flow to drop specific traffic (DoS traffic) is installed.

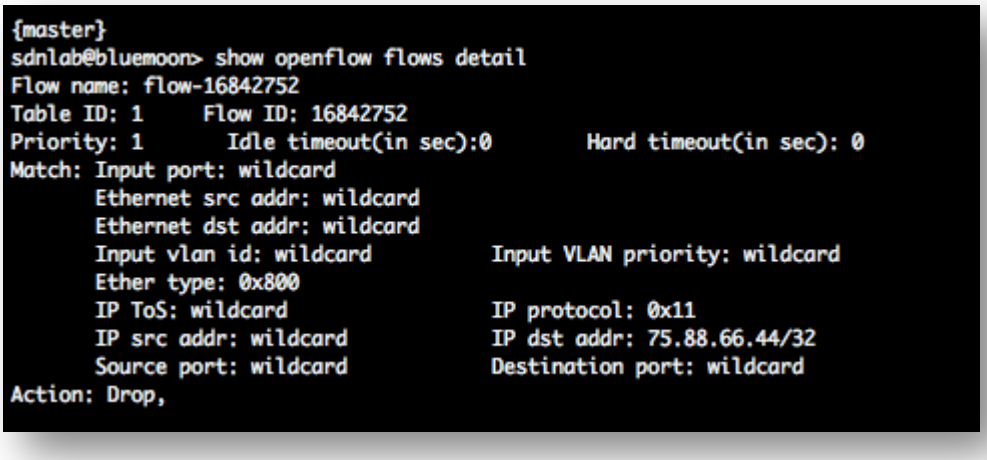

Figure 18. MX960 flow details

The DoS traffic is blocked and is not reaching Ixia (victim).

DDOS DETECTION APPROACH

Overview

In this we will be replacing the Arbor Peakflow SP with a DDoS detection application to detect DDoS attack. This application uses Entropy based approach to detect DDoS attack. Once it is detected the application will generate an alert and send it to alert listener. To optimize entropy based detection approach SNMP trigger will be used. A predefined threshold will be set on interface and monitored using SNMP. The application is divided into three modules namely Netflow collector, analysis based on Entropy and SNMP based trigger.

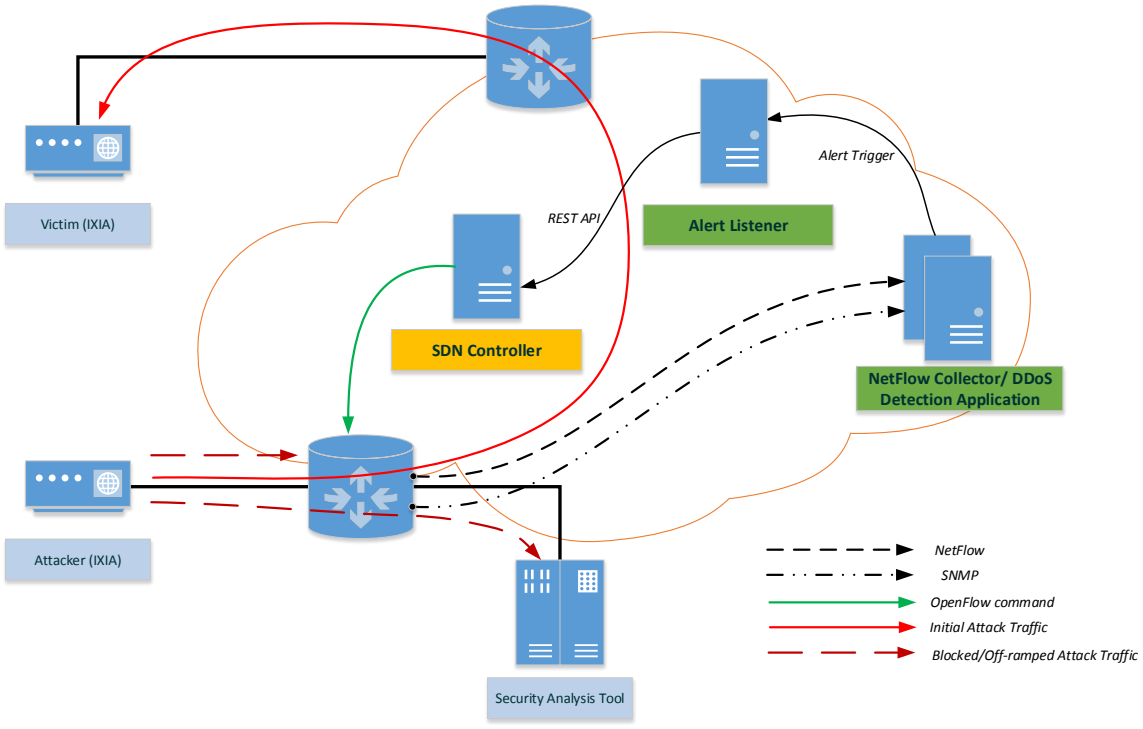

Figure 19.

APPLICATION MODULES

Netflow Collector

The main purpose of this component is to collect Netflow data and store it in database for further analysis. We are using open source application Nfdump to collect Netflow data from openflow router. Nfdump [8] uses nfcapd daemon to read the Netflow data and store data into a file. These files contain Netflow data in binary format. In this Netflow collector module we are using Nfdump command to read nfcap files and store the Netflow data in MySQL database for further analysis.

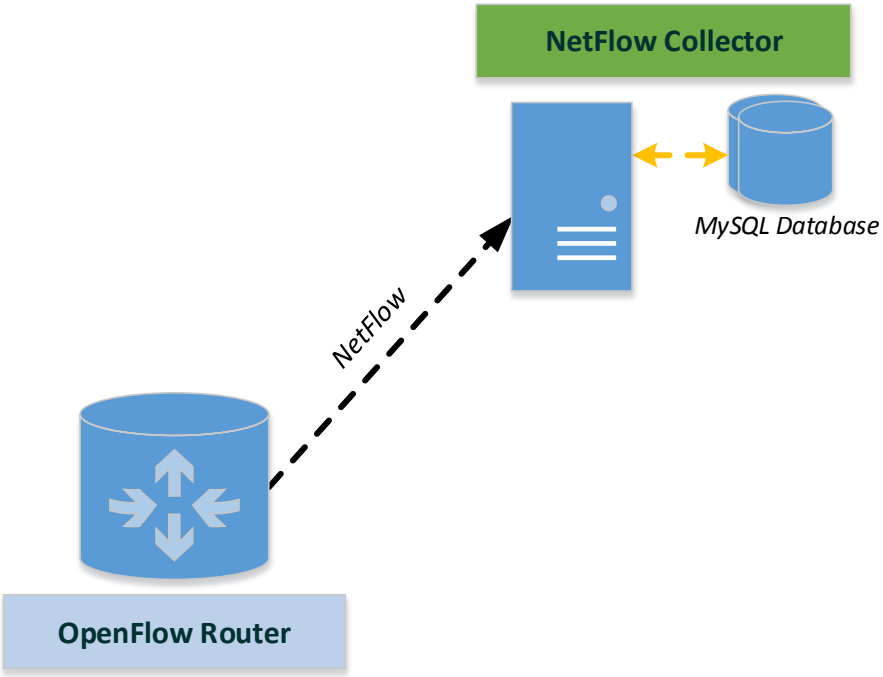

Figure 20.

Analysis Based on Entropy

The entropy-based detection mechanism is utilized to find the measurement of randomness of particular values in the network packet fields. Any packet field's value can be chosen as a basis for calculations, such as source or destination IP addresses, port numbers etc. As an example, if the network attack detector receives a flow of 100 packets, it can calculate the randomness of each individual IP address among this series of packets.

The nature of DDoS attacks is characterized by high volume of traffic and considerable increase of incomplete connection requests. All data packets are mainly constructed by attackers and have a very random source IP addresses. This distinct characteristic will be analyzed in order to detect an abnormal traffic behavior – the DDoS attack.

Based on computation of a continuous series of packets an entropy could be measured. The measured entropy value shows a level of random distribution of the source IP addresses. High value of entropy indicates a high level of source IP address' randomness. In case of low entropy, the source IP addresses randomness is low and some of the sources may prevail among others. The entropy value always varies, even during normal network operation. However, during the attack entropy will change significantly. Through the analysis of entropy value, the source IP distribution alteration can be detected giving sufficient information to mitigate the attack.

Source IP address distribution analysis serves as a detection mechanism for the system. Entropy value measurement is a base of the DDoS detection model [9] and can be improved with additional analytical algorithms. Entropy calculation formula is as follows:

$$H = -\sum_{i=1}^{n} p_i \log_2 p_i$$

Where:

H – entropy;

p_i – the appearance probability of each individual source IP address;

n – the total number of packets in the series.

This module fetches all the required data to calculate entropy from MySQL Database. It compares new entropy value with the pre-defined average range of entropy. In case new entropy value is not in range, it generates alert message (which contains Destination IP under attack, port or protocol) and sends it to alert listener.

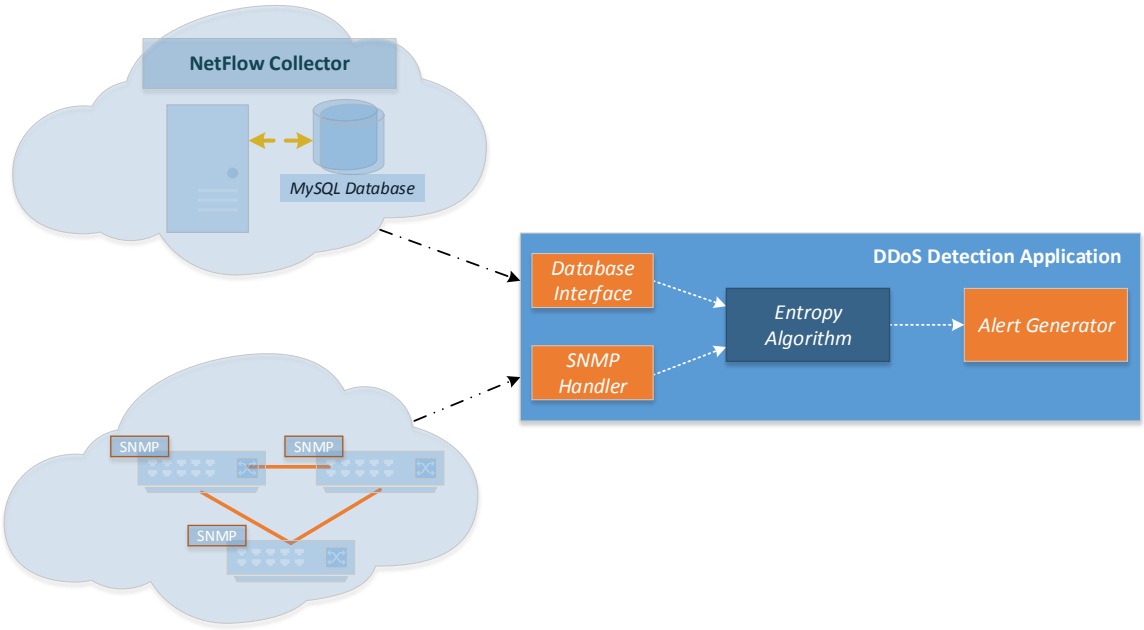

Figure 21.

SNMP Based Trigger

This module acts as SNMP handler. It collects SNMP data from openflow router to calculate input bandwidth utilization of openflow interface. A pre-defined bandwidth threshold will be set. In case the bandwidth utilization is greater than the threshold, an alert will be sent to 'Analysis' module to calculate Entropy of the incoming Netflow data.

REFERENCES

[1] R. Jain and S. Paul, "Network virtualization and software defined networking for cloud computing: a survey," 2013

[2] H. Kim and N. Feamster, "Improving network management with software defined networking," 2013.

[3] N. Feamster, J. Rexford, and E. Zegura, "The road to SDN," Dec. 2013.

[4] ONF, "Open networking foundation," 2014. https://www.opennetworking.org/

[5] Qijun Gu, Peng Liu, "Denial of Service Attacks," 2007.

[6] Juniper TechLibrary, "Configuring Support for OpenFlow on MX Series Routers," http://www.juniper.net/documentation/en_US/junos13.3/topics/task/configuration/junos-sdn-openflow-configuring-mx-series.html

[7] Juniper TechLibrary, "Configuring Inline Flow Monitoring on MX Routers," http://www.juniper.net/techpubs/en_US/junos12.1/topics/task/configuration/inline-flow-monitoring-mx80.html

[8] Nfdump , http://nfdump.sourceforge.net/

[9] L. Li, J. Zhou and N. Xiao, "DDoS Attack Detection Algorithms Based on Entropy Computing," 2007.