Moving Target Defense using Software Defined Networking

Dr. Younghee Park~, Girish SG+,
Khushboo Desai|, Monica Ashok Kumar$, Nikhil KV*

~|$*Computer Engineering Department
+Electrical Engineering Department
~younghee.park@sjsu.edu
+girish.siddananjappagangappa@sjsu.edu #010835877 #50
|Khushboo.desai@sjsu.edu #011458044 #11
$monica.ashokkumar@sjsu.edu #011436815 #01
*nikhilvijayakumar.kengalahalli@sjsu.edu #011418069 #16
San Jose State University
San Jose, USA

Abstract— Static IP configurations has increased the chances of attacks on a network. Attackers can easily track the IP address of the destination if it remains static and thereby spoof the network information. The project, Moving Target Defense (MTD) using Software Defined Network (SDN), describes the architecture and communication protocols of OpenFlow Random Host Mutation. In this project, the network assets are hidden from the external and internal attackers. The OpenFlow Controller is programmed smartly to perform the IP Mutation technique. This technique changes the Real IP Addresses of the underlying hosts by assigning them a Virtual IP address at a high mutation rate. The Virtual IPs are extracted from the pool of unassigned IP addresses generated using pseudo random number generator ensuring high unpredictability. [1]

Keywords—Software Defined Network (SDN), OpenFlow Random Host Mutation (OF-RHM), Moving Target Defense (MTD), IP Mutation

I. INTRODUCTION

Static Configurations can be easily attacked even from a remote network. The attackers send probes using various scanning tools to random hosts in the network to identify the victims and attack them. As per the research, dynamic allocation can be carried out using protocols like DHCP or NAT. Even though NAT and DHCP provide dynamic IP address assignment scheme for hosts, they can't provide adequate security as IP mutation is not implemented frequently and can also be tracked.

In this project, we have implemented Moving Target Defense using OpenFlow Random Host Mutation Technique. We have designed the MTD Architecture containing 8 hosts and 3 switches. The important aspect of the project is providing transparency of IP Mutation to the end host. This transparency is achieved by keeping the real IP intact and mapping a virtual IP to it. The lifetime of virtual address is short keeping the mutation rate high and unpredictable. The source real IP is mapped to virtual IP via the OpenFlow controller, changed at regular intervals using timer event and sent towards the destination. Since the destination IP address of the incoming packet is virtual, the controller passes the packet through the switches until it recognizes the switch that corresponds to the destination host and thereby maps this virtual IP back to the real IP before reaching the destination host. To give a high unpredictability rate to the mutation, a random number generator is used to select the mutant IPs from the pool of unused IPs.[1]

The traditional network infrastructure does not give an efficient performance. The network components need to be added to the network architecture which is quite expensive. Also, management of the network becomes a challenge. Hence, the need for software defined networking is exponentially growing wherein the data plane and the control plane are separate from each other. The management is centralized unlike the traditional network which has a decentralized control. SDN provides openness and programmability of the network. SDN is the best solution for implementing an OpenFlow Random Host Mutation technique to prevent IP Scanning and manage the network as far as security is concerned. A lot of operational overhead can be minimized using SDN.

To run a software defined network, we need an effective controller that handles the traffic efficiently. We have programmed the network infrastructure using RYU controller. Ryu is a SDN framework for developing Networking applications. It is embedded with many software components that are associated with well-defined application interfaces used to develop Management applications in Networking. It supports various communication protocols to communicate with the network routers and switches which includes Openflow, OF-config, Netconf and many more. Ryu supports all OF versions including 1.5 and Nicira.[2]

OpenFlow protocol is a Communication protocol used to communicate between controller and networking devices which includes network switches and routers. OpenFlow protocol is considered as facilitator of Software Defined Networking as it is vendor independent. We have used OF version 1.3 in our project. OpenFlow facilitates remote controlling of layer 3 switch's forwarding table by modifying, adding and removing actions and flow rules.[3] When packets do not match with any of the flow rules in flow table, it will be sent to the controller. Then the controller decides where to forward the packet and installs flow rules on one or more switches to prevent traffic between controller and switch when a similar packet comes to the switch in the future.

II. PROBLEM DOMAIN

Network security against reconnaissance is our problem domain. Management of network by coordinating mutation across the network is the main concern. Performing real IP address to virtual IP address translation using Subnet Gateways is the core functionality.

Main drawback from this approach is that the hosts can still be reached with DNS queries. Since most of the scanners use IP addresses to collect information, this technique is 99% secure. One more challenge is the IPv4 address, the available addresses could be less to implement the mutation.

III. METHOD

The system is designed and implemented using a RYU Controller. The MTD architecture consists of eight hosts and three switches. The communication between host-1 and host-2 is as shown in figure 1. The working of the network is as follows:

1. Incoming packet to the switch-1 with SRC Addr: Real & DST Addr: Virtual
2. Packet-in message to the controller.
3. Controller checks HostAttachments if the destination address is directly connected to the current switch (1).
4. Controller receives a false message.
5. Packet-out message asking the src address to be changed to virtual since it was real.

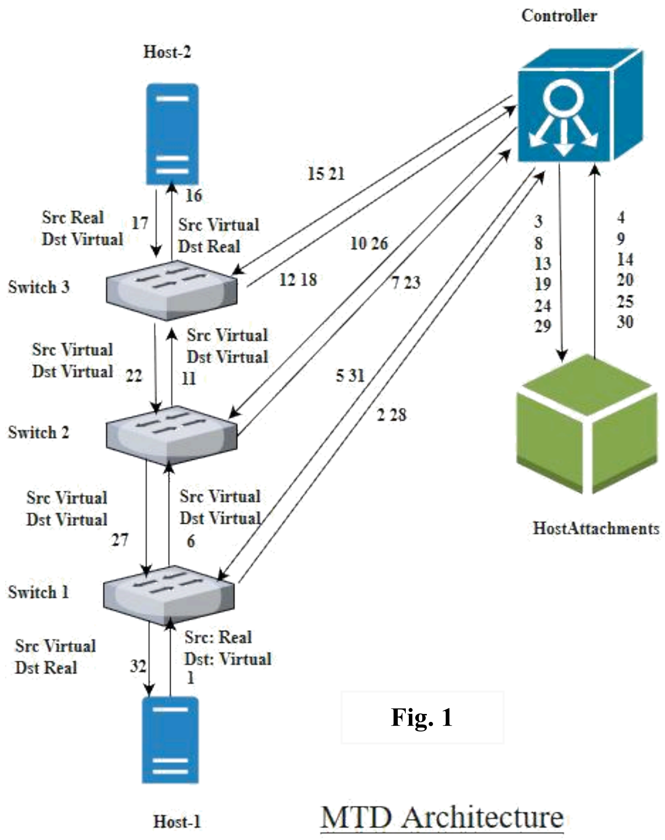


Fig. 1

6. Incoming packet to the switch-2 with SRC Addr: Virtual & DST Addr: Virtual.
7. Packet-in message to the controller.
8. Controller checks HostAttachments if the destination address is directly connected to the current switch (2).
9. Controller receives a false message.
10. Packet out message to forward the packet to next hop.
11. Incoming packet to the switch-3 with SRC Addr: Virtual & DST Addr: Virtual.
12. Packet-in message to the controller.
13. Controller checks HostAttachments if the destination address is directly connected to the current switch (3).
14. Controller receives a True message.
15. Packet out message to change the destination address to Real.
16. Packet to the destination(host-2).
17. Destination host responds with SRC Addr: Real & DST Addr: Virtual.
18. Packet in message to the controller.

19. Controller checks HostAttachments if the destination address is directly connected to the current switch (3).
20. Controller receives a false message.
21. Packet-out message asking the src address to be changed to virtual since it was real.
22. Incoming packet to the switch-2 with SRC Addr: Virtual & DST Addr: Virtual.
23. Packet-in message to the controller.
24. Controller checks HostAttachments if the destination address is directly connected to the current switch (2).
25. Controller receives a false message.
26. Packet out message to forward the packet to next hop.
27. Incoming packet to the switch-1 with SRC Addr: Virtual & DST Addr: Virtual
28. Packet-in message to the controller.
29. Controller checks HostAttachments if the destination address is directly connected to the current switch (1).
30. Controller receives a True message.
31. Packet out message to change the destination address to Real.
32. Packet reaches the destination (ie host1)

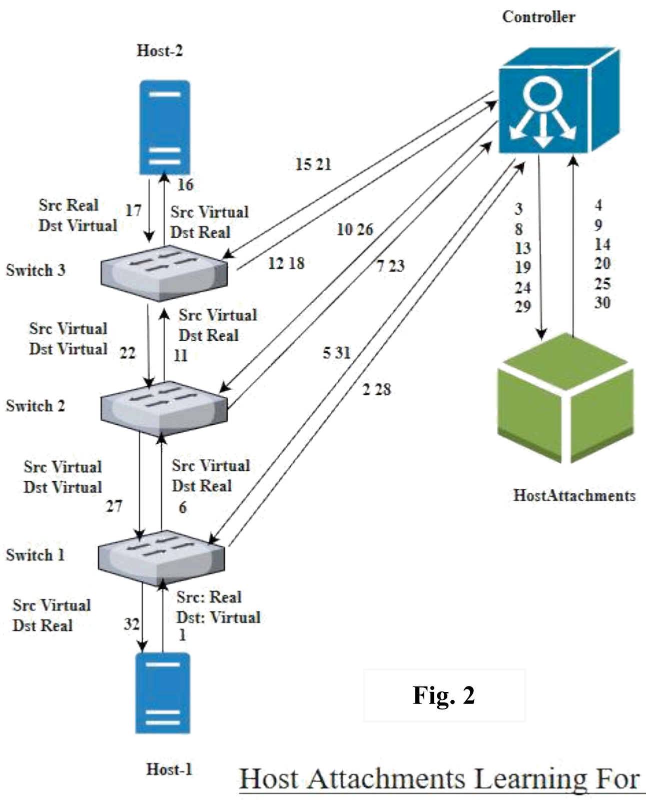


Fig. 2

Host Attachments Learning For MTD

Steps followed to implement the HostAttachments learning for MTD are as follows:

1. Incoming packet to the switch-1 with SRC Addr: Real & DST Addr: Virtual
2. Packet-in message to the controller.
   The system is made to learn as to which switch is the host attached to i.e. fill in the 'HostAttachments' dictionary that maps the real IP with the datapath ID for successful end-to-end delivery of the packet. The flow diagram for the learning MTD can be seen in figure 2
3. Controller does 2 things:
   Checks with the 'HostAttachments' if the destination address is directly connected to the current switch (1).
   Also, checks if there is a key for the src address which is real in the HostAttachements. It doesn't so it stores the src address & switch id as an entry for future.
4. Controller receives a true message about destination being directly connected to the switch. Since there are no entries for the destination address.
5. Packet-out message asking the src address to be changed to virtual since it was real & change the destination address to be real.
6. Incoming packet to the switch-2 with SRC Addr: Virtual & DST Addr: Real. Packet-in message to the controller.
7. Packet-in message to the controller.
8. Controller checks HostAttachments if the destination address is directly connected to the current switch (2).
9. Controller receives a True message since there is no entry.
10. Packet out message to forward the packet to next hop.
11. Incoming packet to the switch-3 with SRC Addr: Virtual & DST Addr: Real.
12. Packet-in message to the controller.
13. Controller checks HostAttachments if the destination address is directly connected to the current switch (3).
14. Controller receives a True message since there is no entry in the HostAttachments.
15. Packet out message forward the packet as it is.
16. Packet to the destination(host-2).

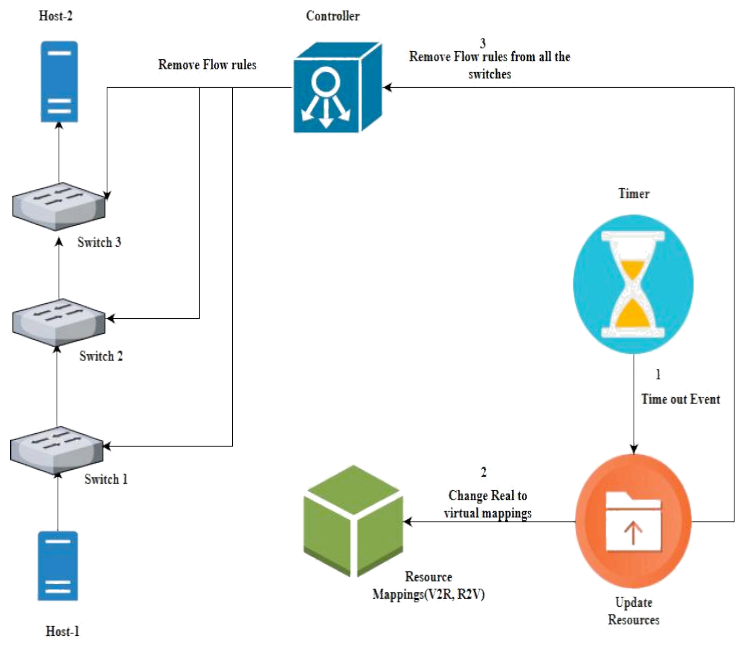
17. Destination host responds with SRC Addr: Real & DST Addr: Virtual.
18. Packet in message to the controller.
19. Controller does 2 things:

    Checks HostAttachments if the destination address is directly connected to the current switch (3). No, it is not.

    Also, checks if there is a key for the src address which is real in the HostAttachments. It doesn't so it stores the src address & switch id as an entry for future
20. Controller receives a false message.
21. Packet-out message asking the src address to be changed to virtual since it was real.
22. Incoming packet to the switch-2 with SRC Addr: Virtual & DST Addr: Virtual.
23. Packet-in message to the controller.
24. Controller checks HostAttachments if the destination address is directly connected to the current switch (2).
25. Controller receives a false message.
26. Packet out message to forward the packet to next hop.
27. Incoming packet to the switch-1 with SRC Addr: Virtual & DST Addr: Virtual
28. Packet-in message to the controller.
29. Controller checks HostAttachments if the destination address is directly connected to the current switch (1).
30. Controller receives a True message.
31. Packet out message to change the destination address to Real.
32. Packet reaches the destination (i.e. host1)

Moreover, the Real-Virtual IP address Life cycle can be seen in the figure 3 below. The life cycle depicts the important aspect of our project implemented using a Timeout Event. This event is generated by the timer every 30 secs and listened by the UpdateResources Function that performs the following two functionalities:

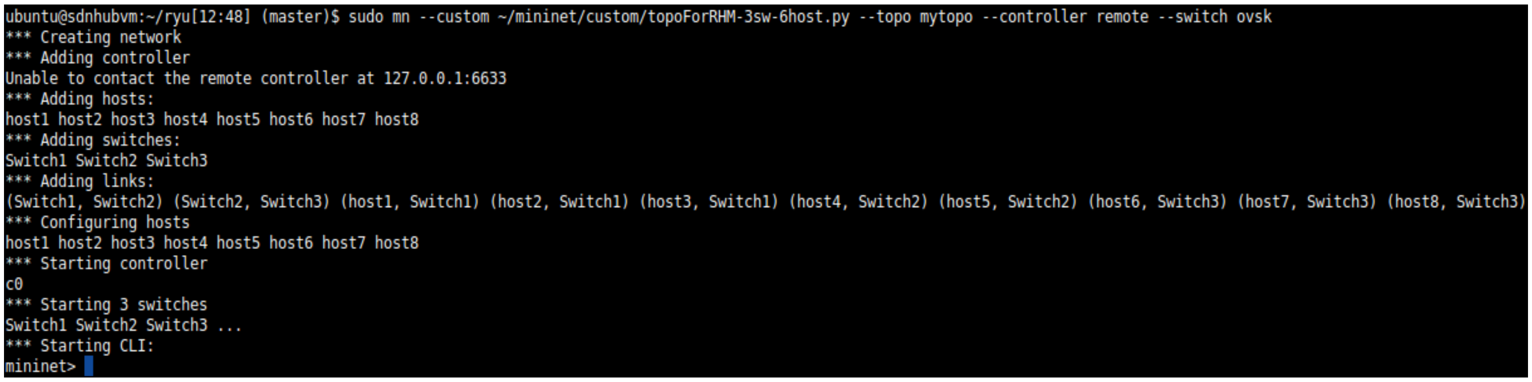
- ➢ Virtual IP addresses corresponding to the Real IP addresses are randomly updated with new IP values from the pool. This leads to change in the real-to-virtual and virtual-to-real mappings keeping the real IP intact but mutating the virtual IP exhibiting security at its best.
- ➢ Deletes all the old flow rules from all the switches for right synchronization with the updated mappings and the packet flow. Hence, after the timeout, the first incoming packet is sent to the controller for assigning the right packet flow.



Real- Virtual IP address Life cycle

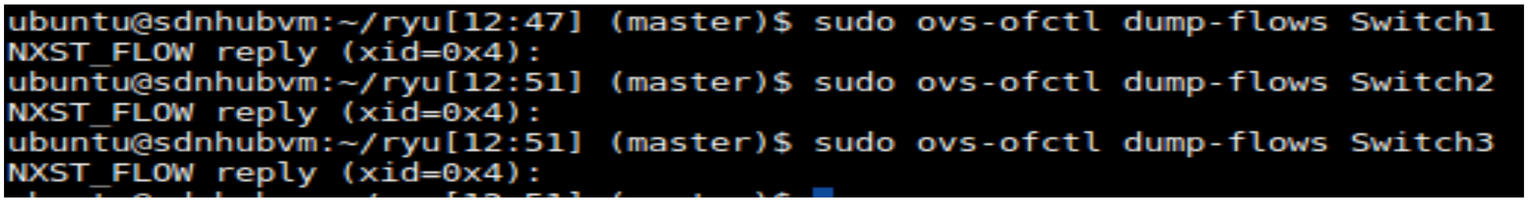
Fig. 3

IV [A]. EVALUATION CAPTURES



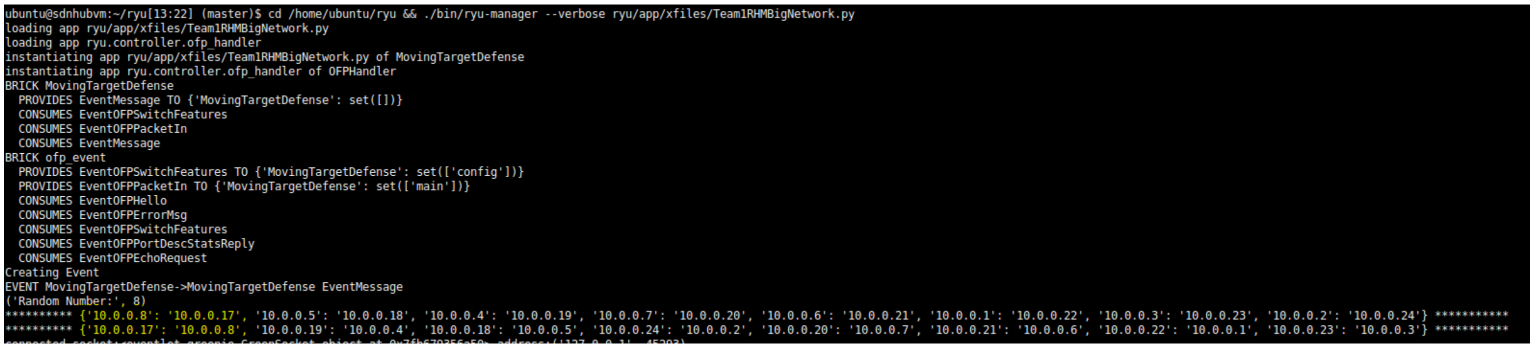
```
ubuntu@sdnhubvm:~/ryu[12:48] (master)$ sudo mn --custom ~/mininet/custom/topoForRHM-3sw-6host.py --topo mytopo --controller remote --switch ovsk
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
host1 host2 host3 host4 host5 host6 host7 host8
*** Adding switches:
Switch1 Switch2 Switch3
*** Adding links:
(Switch1, Switch2) (Switch2, Switch3) (host1, Switch1) (host2, Switch1) (host3, Switch1) (host4, Switch2) (host5, Switch2) (host6, Switch3) (host7, Switch3) (host8, Switch3)
*** Configuring hosts
host1 host2 host3 host4 host5 host6 host7 host8
*** Starting controller
c0
*** Starting 3 switches
Switch1 Switch2 Switch3 ...
*** Starting CLI:
mininet>
```

Fig. 5



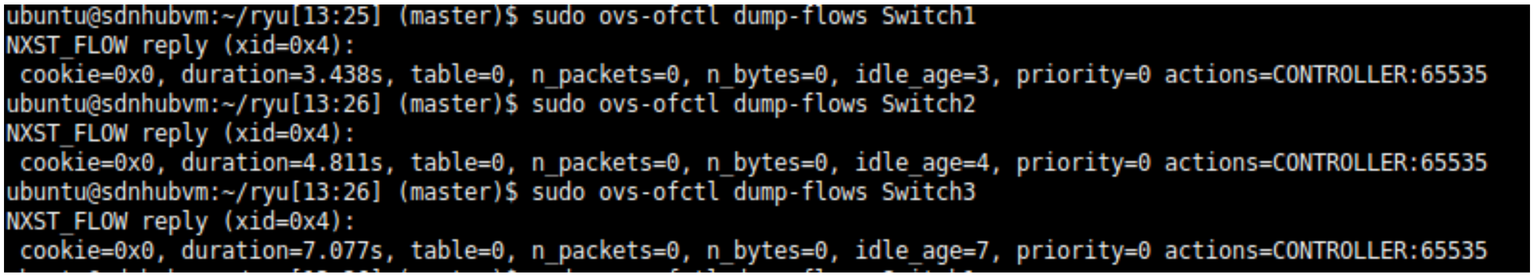
```
ubuntu@sdnhubvm:~/ryu[12:47] (master)$ sudo ovs-ofctl dump-flows Switch1
NXST_FLOW reply (xid=0x4):
ubuntu@sdnhubvm:~/ryu[12:51] (master)$ sudo ovs-ofctl dump-flows Switch2
NXST_FLOW reply (xid=0x4):
ubuntu@sdnhubvm:~/ryu[12:51] (master)$ sudo ovs-ofctl dump-flows Switch3
NXST_FLOW reply (xid=0x4):
```

Fig. 6



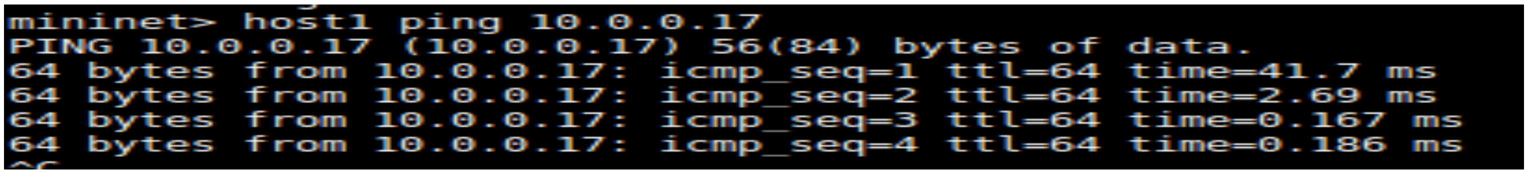
```
ubuntu@sdnhubvm:~/ryu[13:22] (master)$ cd /home/ubuntu/ryu && ./bin/ryu-manager --verbose ryu/app/xfiles/Team1RHMBigNetwork.py
loading app ryu/app/xfiles/Team1RHMBigNetwork.py
loading app ryu.controller.ofp_handler
instantiating app ryu/app/xfiles/Team1RHMBigNetwork.py of MovingTargetDefense
instantiating app ryu.controller.ofp_handler of OFPHandler
BRICK MovingTargetDefense
  PROVIDES EventMessage TO {'MovingTargetDefense': set([])}
  CONSUMES EventOFPSwitchFeatures
  CONSUMES EventOFPPacketIn
  CONSUMES EventMessage
BRICK ofp_event
  PROVIDES EventOFPSwitchFeatures TO {'MovingTargetDefense': set(['config'])}
  PROVIDES EventOFPPacketIn TO {'MovingTargetDefense': set(['main'])}
  CONSUMES EventOFPHello
  CONSUMES EventOFPErrorMsg
  CONSUMES EventOFPSwitchFeatures
  CONSUMES EventOFPPortDescStatsReply
  CONSUMES EventOFPEchoRequest
Creating Event
EVENT MovingTargetDefense->MovingTargetDefense EventMessage
('Random Number:', 8)
********** {'10.0.0.8': '10.0.0.17', '10.0.0.5': '10.0.0.18', '10.0.0.4': '10.0.0.19', '10.0.0.7': '10.0.0.20', '10.0.0.6': '10.0.0.21', '10.0.0.1': '10.0.0.22', '10.0.0.3': '10.0.0.23', '10.0.0.2': '10.0.0.24'} ***********
********** {'10.0.0.17': '10.0.0.8', '10.0.0.19': '10.0.0.4', '10.0.0.18': '10.0.0.5', '10.0.0.24': '10.0.0.2', '10.0.0.20': '10.0.0.7', '10.0.0.21': '10.0.0.6', '10.0.0.22': '10.0.0.1', '10.0.0.23': '10.0.0.3'} ***********
```

Fig. 7



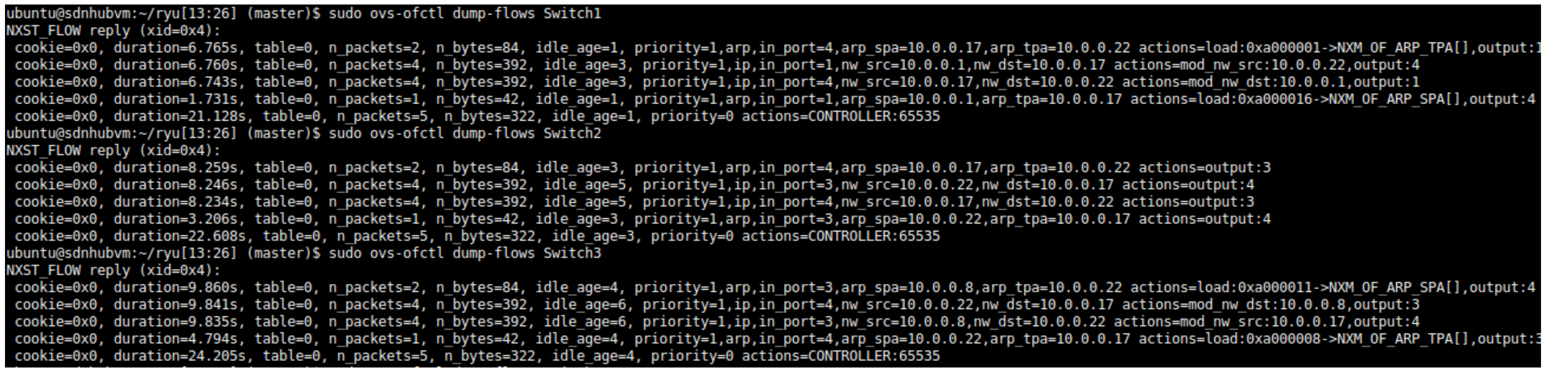
```
ubuntu@sdnhubvm:~/ryu[13:25] (master)$ sudo ovs-ofctl dump-flows Switch1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=3.438s, table=0, n_packets=0, n_bytes=0, idle_age=3, priority=0 actions=CONTROLLER:65535
ubuntu@sdnhubvm:~/ryu[13:26] (master)$ sudo ovs-ofctl dump-flows Switch2
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=4.811s, table=0, n_packets=0, n_bytes=0, idle_age=4, priority=0 actions=CONTROLLER:65535
ubuntu@sdnhubvm:~/ryu[13:26] (master)$ sudo ovs-ofctl dump-flows Switch3
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=7.077s, table=0, n_packets=0, n_bytes=0, idle_age=7, priority=0 actions=CONTROLLER:65535
```

Fig. 8



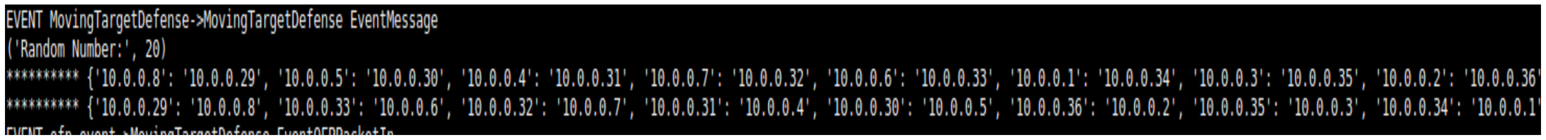
```
mininet> host1 ping 10.0.0.17
PING 10.0.0.17 (10.0.0.17) 56(84) bytes of data.
64 bytes from 10.0.0.17: icmp_seq=1 ttl=64 time=41.7 ms
64 bytes from 10.0.0.17: icmp_seq=2 ttl=64 time=2.69 ms
64 bytes from 10.0.0.17: icmp_seq=3 ttl=64 time=0.167 ms
64 bytes from 10.0.0.17: icmp_seq=4 ttl=64 time=0.186 ms
```

Fig. 9



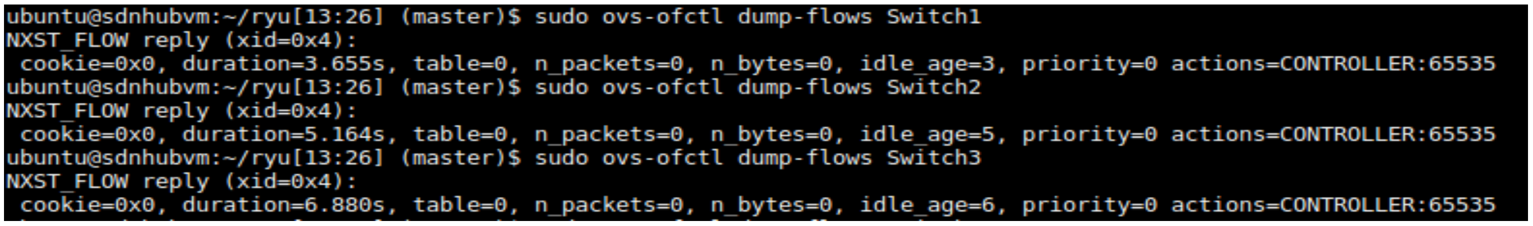
```
ubuntu@sdnhubvm:~/ryu[13:26] (master)$ sudo ovs-ofctl dump-flows Switch1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=6.765s, table=0, n_packets=2, n_bytes=84, idle_age=1, priority=1,arp,in_port=4,arp_spa=10.0.0.17,arp_tpa=10.0.0.22 actions=load:0xa000001->NXM_OF_ARP_TPA[],output:1
 cookie=0x0, duration=6.760s, table=0, n_packets=4, n_bytes=392, idle_age=3, priority=1,ip,in_port=1,nw_src=10.0.0.1,nw_dst=10.0.0.17 actions=mod_nw_src:10.0.0.22,output:4
 cookie=0x0, duration=6.743s, table=0, n_packets=4, n_bytes=392, idle_age=3, priority=1,ip,in_port=4,nw_src=10.0.0.17,nw_dst=10.0.0.22 actions=mod_nw_dst:10.0.0.1,output:1
 cookie=0x0, duration=1.731s, table=0, n_packets=1, n_bytes=42, idle_age=1, priority=1,arp,in_port=1,arp_spa=10.0.0.1,arp_tpa=10.0.0.17 actions=load:0xa000016->NXM_OF_ARP_SPA[],output:4
 cookie=0x0, duration=21.128s, table=0, n_packets=5, n_bytes=322, idle_age=1, priority=0 actions=CONTROLLER:65535
ubuntu@sdnhubvm:~/ryu[13:26] (master)$ sudo ovs-ofctl dump-flows Switch2
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=8.259s, table=0, n_packets=2, n_bytes=84, idle_age=3, priority=1,arp,in_port=4,arp_spa=10.0.0.17,arp_tpa=10.0.0.22 actions=output:3
 cookie=0x0, duration=8.246s, table=0, n_packets=4, n_bytes=392, idle_age=5, priority=1,ip,in_port=3,nw_src=10.0.0.22,nw_dst=10.0.0.17 actions=output:4
 cookie=0x0, duration=8.234s, table=0, n_packets=4, n_bytes=392, idle_age=5, priority=1,ip,in_port=4,nw_src=10.0.0.17,nw_dst=10.0.0.22 actions=output:3
 cookie=0x0, duration=3.206s, table=0, n_packets=1, n_bytes=42, idle_age=3, priority=1,arp,in_port=3,arp_spa=10.0.0.22,arp_tpa=10.0.0.17 actions=output:4
 cookie=0x0, duration=22.608s, table=0, n_packets=5, n_bytes=322, idle_age=3, priority=0 actions=CONTROLLER:65535
ubuntu@sdnhubvm:~/ryu[13:26] (master)$ sudo ovs-ofctl dump-flows Switch3
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=9.860s, table=0, n_packets=2, n_bytes=84, idle_age=4, priority=1,arp,in_port=3,arp_spa=10.0.0.8,arp_tpa=10.0.0.22 actions=load:0xa000011->NXM_OF_ARP_SPA[],output:4
 cookie=0x0, duration=9.841s, table=0, n_packets=4, n_bytes=392, idle_age=6, priority=1,ip,in_port=4,nw_src=10.0.0.22,nw_dst=10.0.0.17 actions=mod_nw_dst:10.0.0.8,output:3
 cookie=0x0, duration=9.835s, table=0, n_packets=4, n_bytes=392, idle_age=6, priority=1,ip,in_port=3,nw_src=10.0.0.8,nw_dst=10.0.0.22 actions=mod_nw_src:10.0.0.17,output:4
 cookie=0x0, duration=4.794s, table=0, n_packets=1, n_bytes=42, idle_age=4, priority=1,arp,in_port=4,arp_spa=10.0.0.22,arp_tpa=10.0.0.17 actions=load:0xa000008->NXM_OF_ARP_TPA[],output:3
 cookie=0x0, duration=24.205s, table=0, n_packets=5, n_bytes=322, idle_age=4, priority=0 actions=CONTROLLER:65535
```

Fig. 10



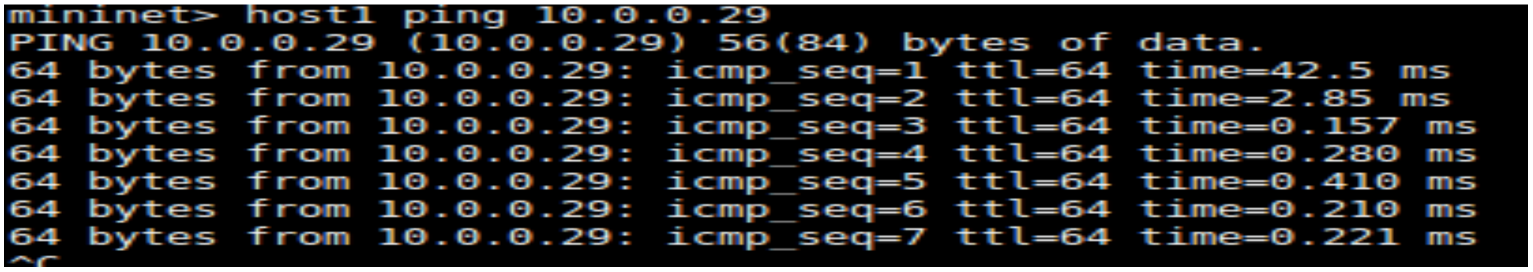
```
EVENT MovingTargetDefense->MovingTargetDefense EventMessage
('Random Number:', 20)
********** {'10.0.0.8': '10.0.0.29', '10.0.0.5': '10.0.0.30', '10.0.0.4': '10.0.0.31', '10.0.0.7': '10.0.0.32', '10.0.0.6': '10.0.0.33', '10.0.0.1': '10.0.0.34', '10.0.0.3': '10.0.0.35', '10.0.0.2': '10.0.0.36'
********** {'10.0.0.29': '10.0.0.8', '10.0.0.33': '10.0.0.6', '10.0.0.32': '10.0.0.7', '10.0.0.31': '10.0.0.4', '10.0.0.30': '10.0.0.5', '10.0.0.36': '10.0.0.2', '10.0.0.35': '10.0.0.3', '10.0.0.34': '10.0.0.1'
```

Fig. 11



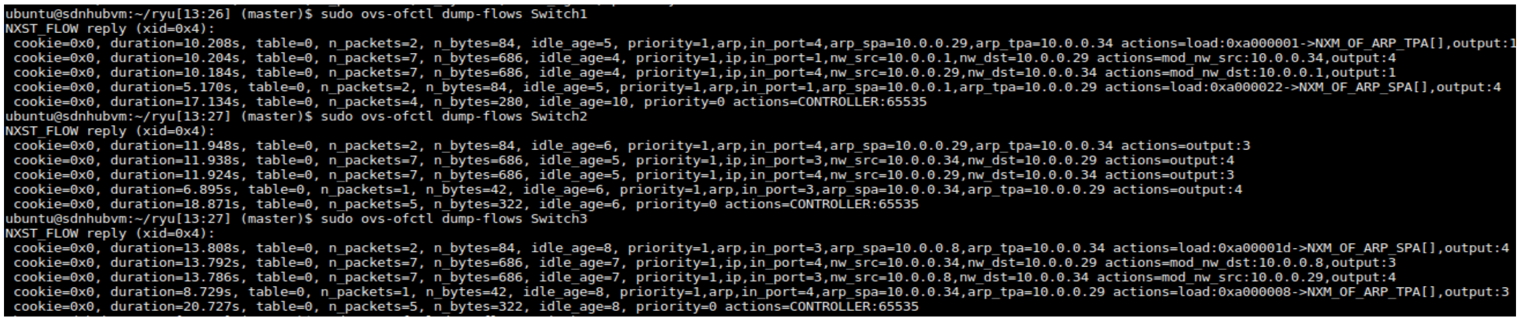
```
ubuntu@sdnhubvm:~/ryu[13:26] (master)$ sudo ovs-ofctl dump-flows Switch1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=3.655s, table=0, n_packets=0, n_bytes=0, idle_age=3, priority=0 actions=CONTROLLER:65535
ubuntu@sdnhubvm:~/ryu[13:26] (master)$ sudo ovs-ofctl dump-flows Switch2
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=5.164s, table=0, n_packets=0, n_bytes=0, idle_age=5, priority=0 actions=CONTROLLER:65535
ubuntu@sdnhubvm:~/ryu[13:26] (master)$ sudo ovs-ofctl dump-flows Switch3
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=6.880s, table=0, n_packets=0, n_bytes=0, idle_age=6, priority=0 actions=CONTROLLER:65535
```

Fig. 12



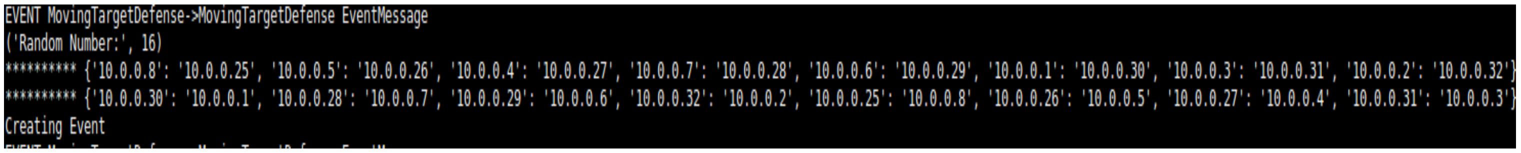
```
mininet> host1 ping 10.0.0.29
PING 10.0.0.29 (10.0.0.29) 56(84) bytes of data.
64 bytes from 10.0.0.29: icmp_seq=1 ttl=64 time=42.5 ms
64 bytes from 10.0.0.29: icmp_seq=2 ttl=64 time=2.85 ms
64 bytes from 10.0.0.29: icmp_seq=3 ttl=64 time=0.157 ms
64 bytes from 10.0.0.29: icmp_seq=4 ttl=64 time=0.280 ms
64 bytes from 10.0.0.29: icmp_seq=5 ttl=64 time=0.410 ms
64 bytes from 10.0.0.29: icmp_seq=6 ttl=64 time=0.210 ms
64 bytes from 10.0.0.29: icmp_seq=7 ttl=64 time=0.221 ms
^C
```

Fig. 13



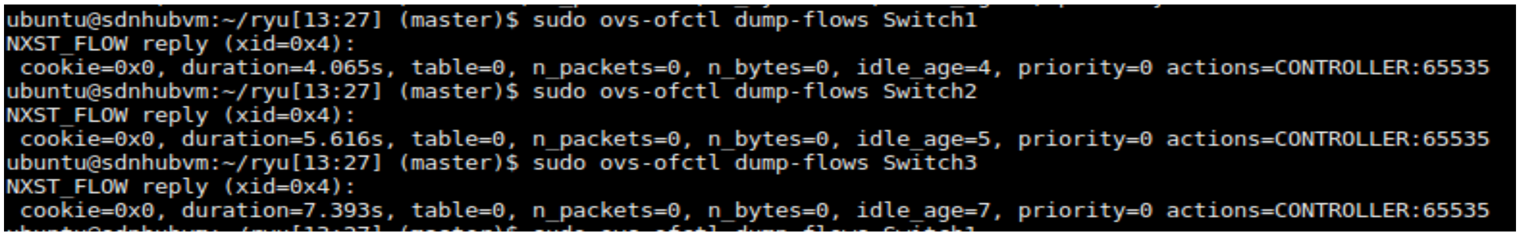
```
ubuntu@sdnhubvm:~/ryu[13:26] (master)$ sudo ovs-ofctl dump-flows Switch1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=10.208s, table=0, n_packets=2, n_bytes=84, idle_age=5, priority=1,arp,in_port=4,arp_spa=10.0.0.29,arp_tpa=10.0.0.34 actions=load:0xa000001->NXM_OF_ARP_TPA[],output:1
 cookie=0x0, duration=10.204s, table=0, n_packets=7, n_bytes=686, idle_age=4, priority=1,ip,in_port=1,nw_src=10.0.0.1,nw_dst=10.0.0.29 actions=mod_nw_src:10.0.0.34,output:4
 cookie=0x0, duration=10.184s, table=0, n_packets=7, n_bytes=686, idle_age=4, priority=1,ip,in_port=4,nw_src=10.0.0.29,nw_dst=10.0.0.34 actions=mod_nw_dst:10.0.0.1,output:1
 cookie=0x0, duration=5.170s, table=0, n_packets=2, n_bytes=84, idle_age=5, priority=1,arp,in_port=1,arp_spa=10.0.0.1,arp_tpa=10.0.0.29 actions=load:0xa000022->NXM_OF_ARP_SPA[],output:4
 cookie=0x0, duration=17.134s, table=0, n_packets=4, n_bytes=280, idle_age=10, priority=0 actions=CONTROLLER:65535
ubuntu@sdnhubvm:~/ryu[13:27] (master)$ sudo ovs-ofctl dump-flows Switch2
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=11.948s, table=0, n_packets=2, n_bytes=84, idle_age=6, priority=1,arp,in_port=4,arp_spa=10.0.0.29,arp_tpa=10.0.0.34 actions=output:3
 cookie=0x0, duration=11.938s, table=0, n_packets=7, n_bytes=686, idle_age=5, priority=1,ip,in_port=3,nw_src=10.0.0.34,nw_dst=10.0.0.29 actions=output:4
 cookie=0x0, duration=11.924s, table=0, n_packets=7, n_bytes=686, idle_age=5, priority=1,ip,in_port=4,nw_src=10.0.0.29,nw_dst=10.0.0.34 actions=output:3
 cookie=0x0, duration=6.895s, table=0, n_packets=1, n_bytes=42, idle_age=6, priority=1,arp,in_port=3,arp_spa=10.0.0.34,arp_tpa=10.0.0.29 actions=output:4
 cookie=0x0, duration=18.871s, table=0, n_packets=5, n_bytes=322, idle_age=6, priority=0 actions=CONTROLLER:65535
ubuntu@sdnhubvm:~/ryu[13:27] (master)$ sudo ovs-ofctl dump-flows Switch3
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=13.808s, table=0, n_packets=2, n_bytes=84, idle_age=8, priority=1,arp,in_port=3,arp_spa=10.0.0.8,arp_tpa=10.0.0.34 actions=load:0xa00001d->NXM_OF_ARP_SPA[],output:4
 cookie=0x0, duration=13.792s, table=0, n_packets=7, n_bytes=686, idle_age=7, priority=1,ip,in_port=4,nw_src=10.0.0.34,nw_dst=10.0.0.29 actions=mod_nw_dst:10.0.0.8,output:3
 cookie=0x0, duration=13.786s, table=0, n_packets=7, n_bytes=686, idle_age=7, priority=1,ip,in_port=3,nw_src=10.0.0.8,nw_dst=10.0.0.34 actions=mod_nw_src:10.0.0.29,output:4
 cookie=0x0, duration=8.729s, table=0, n_packets=1, n_bytes=42, idle_age=8, priority=1,arp,in_port=4,arp_spa=10.0.0.34,arp_tpa=10.0.0.29 actions=load:0xa000008->NXM_OF_ARP_TPA[],output:3
 cookie=0x0, duration=20.727s, table=0, n_packets=5, n_bytes=322, idle_age=8, priority=0 actions=CONTROLLER:65535
```

Fig. 14



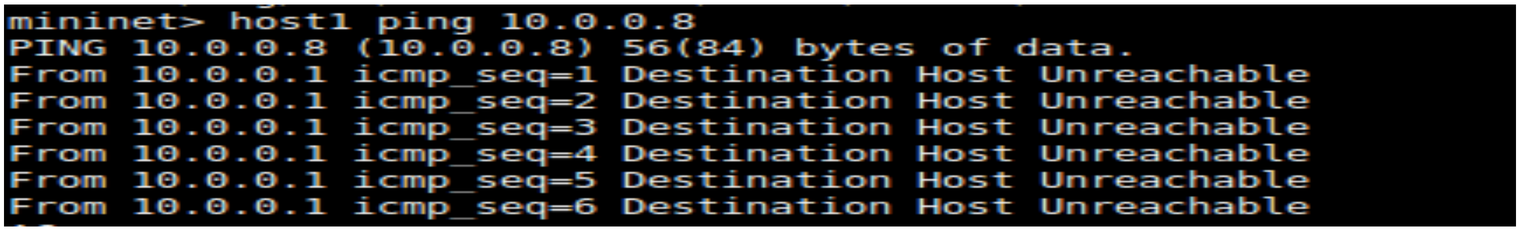
```
EVENT MovingTargetDefense->MovingTargetDefense EventMessage
('Random Number:', 16)
********** {'10.0.0.8': '10.0.0.25', '10.0.0.5': '10.0.0.26', '10.0.0.4': '10.0.0.27', '10.0.0.7': '10.0.0.28', '10.0.0.6': '10.0.0.29', '10.0.0.1': '10.0.0.30', '10.0.0.3': '10.0.0.31', '10.0.0.2': '10.0.0.32'}
********** {'10.0.0.30': '10.0.0.1', '10.0.0.28': '10.0.0.7', '10.0.0.29': '10.0.0.6', '10.0.0.32': '10.0.0.2', '10.0.0.25': '10.0.0.8', '10.0.0.26': '10.0.0.5', '10.0.0.27': '10.0.0.4', '10.0.0.31': '10.0.0.3'}
Creating Event
```

Fig. 15



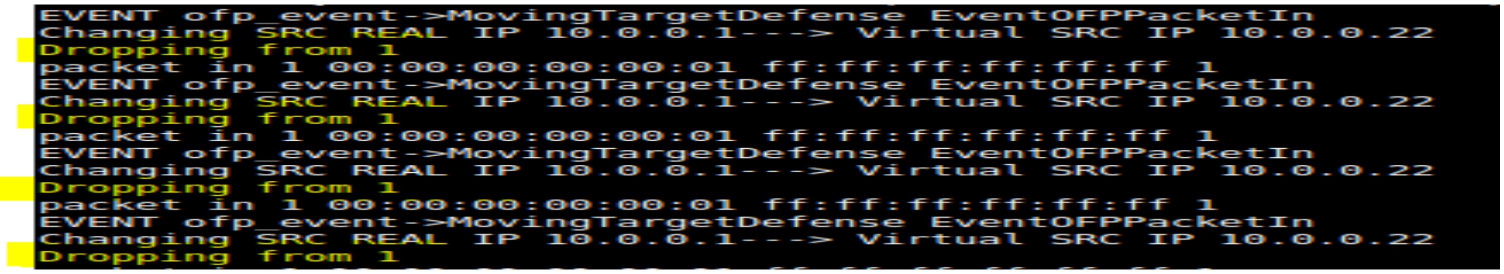
```
ubuntu@sdnhubvm:~/ryu[13:27] (master)$ sudo ovs-ofctl dump-flows Switch1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=4.065s, table=0, n_packets=0, n_bytes=0, idle_age=4, priority=0 actions=CONTROLLER:65535
ubuntu@sdnhubvm:~/ryu[13:27] (master)$ sudo ovs-ofctl dump-flows Switch2
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=5.616s, table=0, n_packets=0, n_bytes=0, idle_age=5, priority=0 actions=CONTROLLER:65535
ubuntu@sdnhubvm:~/ryu[13:27] (master)$ sudo ovs-ofctl dump-flows Switch3
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=7.393s, table=0, n_packets=0, n_bytes=0, idle_age=7, priority=0 actions=CONTROLLER:65535
```

Fig. 16



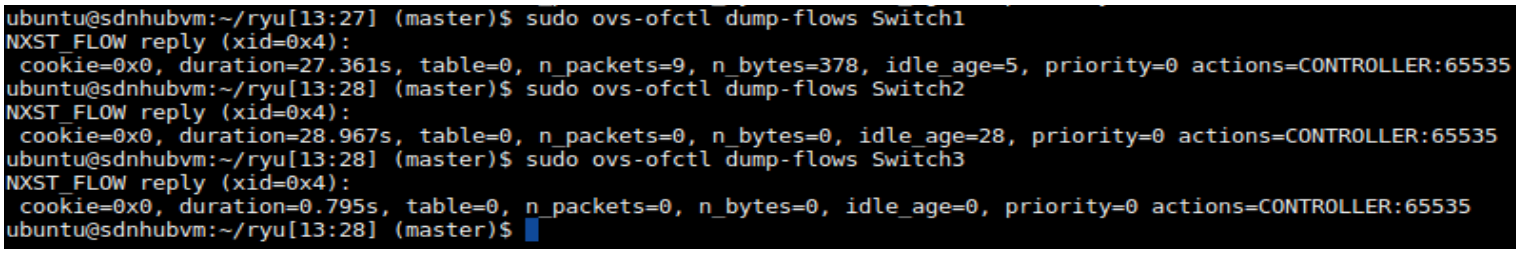
```
mininet> host1 ping 10.0.0.8
PING 10.0.0.8 (10.0.0.8) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable
From 10.0.0.1 icmp_seq=4 Destination Host Unreachable
From 10.0.0.1 icmp_seq=5 Destination Host Unreachable
From 10.0.0.1 icmp_seq=6 Destination Host Unreachable
```

Fig. 17



```
EVENT ofp_event->MovingTargetDefense EventOFPPacketIn
Changing SRC REAL IP 10.0.0.1---> Virtual SRC IP 10.0.0.22
Dropping from 1
packet in 1 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 1
EVENT ofp_event->MovingTargetDefense EventOFPPacketIn
Changing SRC REAL IP 10.0.0.1---> Virtual SRC IP 10.0.0.22
Dropping from 1
packet in 1 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 1
EVENT ofp_event->MovingTargetDefense EventOFPPacketIn
Changing SRC REAL IP 10.0.0.1---> Virtual SRC IP 10.0.0.22
Dropping from 1
packet in 1 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 1
EVENT ofp_event->MovingTargetDefense EventOFPPacketIn
Changing SRC REAL IP 10.0.0.1---> Virtual SRC IP 10.0.0.22
Dropping from 1
```

Fig. 18



```
ubuntu@sdnhubvm:~/ryu[13:27] (master)$ sudo ovs-ofctl dump-flows Switch1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=27.361s, table=0, n_packets=9, n_bytes=378, idle_age=5, priority=0 actions=CONTROLLER:65535
ubuntu@sdnhubvm:~/ryu[13:28] (master)$ sudo ovs-ofctl dump-flows Switch2
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=28.967s, table=0, n_packets=0, n_bytes=0, idle_age=28, priority=0 actions=CONTROLLER:65535
ubuntu@sdnhubvm:~/ryu[13:28] (master)$ sudo ovs-ofctl dump-flows Switch3
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=0.795s, table=0, n_packets=0, n_bytes=0, idle_age=0, priority=0 actions=CONTROLLER:65535
ubuntu@sdnhubvm:~/ryu[13:28] (master)$
```

Fig. 19

IV [B]. EVALUATION WALK-THROUGH

1. The system Evaluation and Analysis is carried out using the following topology:

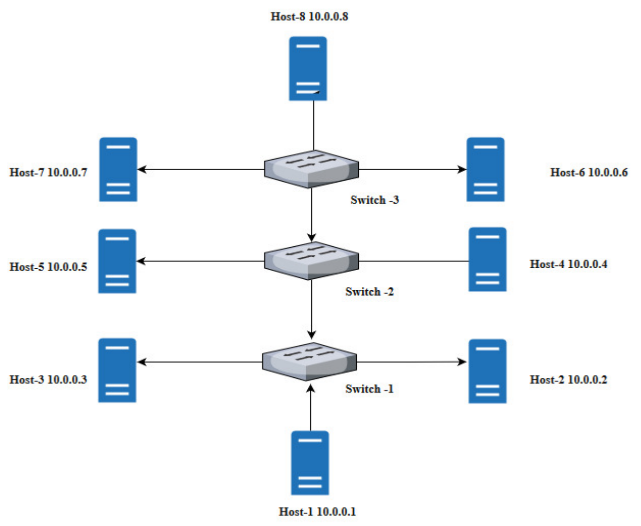


Fig. 4

2. Using the command shown in **Figure 5**, the topology as shown in **Figure 4** is created with three switches and eight hosts wherein Host 1 with Real IP: 10.0.0.1 is made to PING Host 8 with Real IP: 10.0.0.8
3. Dump flow rules in all the three switches shows no flow rules as shown in **Figure 6**.
4. On starting the controller, the table miss entry will be installed to all the switches i.e. the Default Flow Rule (action: controller) as well as the mappings are initialized. **Refer Figure 7**.
5. The Initial Real to Virtual IP address & Virtual to Real IP address mapping is as shown below.

Real IP address	Virtual IP address
10.0.0.1	10.0.0.22
10.0.0.8	10.0.0.17

Table 1

6. Now on dumping flows, the default flow rule involving actions=controller is observed as shown in **Figure 8.**
7. When Host1 initiates a ping with Host 8 having Virtual IP address 10.0.0.17, the ICMP packets flow and a successful PING is established since Destination Host is reachable. It can be observed in **Figure 9**.
8. The dump flows output can be seen in **Figure 10** depicting how source real IP is able to ping the destination virtual IP.
9. After the timer times out, a time out event is initiated that updates the Virtual-IPs thereby updating mappings & removes the old installed flow rules from all the switches. **Refer Figure 11**.
10. Hence, now host1 has Virtual IP 10.0.0.34 instead of 10.0.0.22 and Host8 has Virtual IP 10.0.0.29 instead of 10.0.0.17

Real IP address	Virtual IP address
10.0.0.1	10.0.0.34
10.0.0.8	10.0.0.29

Table 2

11. Dump flows output can be seen with default flow rule since the flows were deleted after time out event as shown in **Figure 12.**
12. Again, Host1 initiates a PING with IP address 10.0.0.29 and is successful as shown in **Figure 13.**
13. Dump flows now show the updated mappings. **Refer Figure 14**.
14. Again, a time out event initiates an update of the Real-IP mappings & removes the installed flow rules from all the switches. **Refer Figure 15.**
15. Following IP Mutation is observed:

Real IP address	Virtual IP address
10.0.0.1	10.0.0.30
10.0.0.8	10.0.0.25

Table 3

16. Dump flows output indicate that the flow rules are set to default again as shown in **Figure 16.**
17. Host1 tries to reach Host8 using real IP address 10.0.0.8; it is observed that the PING is not successful and destination host is unreachable. **Refer Figure 17.**
18. The EventOFPPacketIn makes the packet drop at switch 1 itself discouraging the Host1 to ping Host8 via Real IP. **Refer Figure 18.**
19. All packets are dropped at the first switch. No flow rules are installed. **Refer Figure 19.**

V. DISCUSSION

Ryu controller is one amongst many controllers which can be used to program the packet flow in a software defined network. The heart of random host mutation is modifying the packet's source & destination IP addresses. Actions & match objects are used effectively to install flow rules to the switches. This avoids the similar packets coming to the controller. The modification is handled by the flow rules at the switch level with the controller's interception.

OpenFlow requires certain prerequisite match fields to be specified for specific header modifications. Modification of ARP source & destination address (OXM_OF_ARP_SPA & OXM_OF_ARP_TPA) requires the specification of ETHTYPE=0x0806. [7]

Modification of IPV4 source & destination address (OXM_OF_IPV4_SRC & OXM_OF_IPV4_DST) requires the specification of ETHTYPE=0x0800. [7]
These are OpenFlow prerequisites. When not specified, missing prerequisite errors are thrown which can be analyzed with the help of Error messages from Open Flow document. [8]

VI. FUTURE SCOPE

Implementation of DNS response interception by the controller: When a host request for the IP address of the destination host, the DNS server responds with the real IP address of the destination. The Ryu controller intercepts the packet and changes the real IP address to virtual IP address before it reaches the requesting host.

Implementation of Authorized user functionality: An authorized user will be able to ping the destination host using real IP address. A REST API will be implemented to add authorized users to the list based on the IP addresses provided in JSON format.

The real IP detection within the Network can be automated.

VII. CONCLUSION

In this project, we have implemented Moving Target Defense by randomly mutating IP addresses of hosts using Software Defined Network. The main aim of MTD is to prevent the discovery of host by the probes sent by various scanning tools and worms. In this project, the real IP addresses of the hosts are hidden from both internal and external scanners to prevent attacks. The Ryu controller is used to mutate hosts IP addresses. Host's real IP address are assigned with virtual IP address at a high mutation rate from an unused resource pool of IP addresses by using pseudo random number generator.

VIII. REFERENCES

[1] J. H. Jafarian, E. Al-Shaer, and Q. Duan, "Openflow random host mutation," Proceedings of the first workshop on Hot topics in software defined networks - HotSDN '12, 2012.
[2] "Download," *Ryu SDN Framework*. [Online]. Available: https://osrg.github.io/ryu/. [Accessed: 07-May-2017].
[3] "OpenFlow", En.wikipedia.org, 2017. [Online]. Available: https://en.wikipedia.org/wiki/OpenFlow. [Accessed: 07- May- 2017].

[4] "ryu," ryu / Mailing Lists. [Online]. Available: https://sourceforge.net/p/ryu/mailman/ryu-devel/?viewmonth=201601&viewday=12. [Accessed: 07-May-2017].
[5] "The First Application¶," The First Application — Ryu 4.13 documentation. [Online]. Available: http://ryu.readthedocs.io/en/latest/writing_ryu_app.html. [Accessed: 07-May-2017].
[6] "ryu," ryu / Mailing Lists. [Online]. Available: https://sourceforge.net/p/ryu/mailman/message/32333352 . [Accessed: 07-May-2017].
[7]https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.1.pdf

APPENDIX

This application implements Moving Target Defense mechanism, in specific: ***Random Host Mutation*** technique. It is used to ***hide*** the ***real IP address*** of the host from other hosts in the network and also to hosts outside of the network. A ***timer*** has been implemented to change the real IP address to virtual IP address mapping. To have seamless connectivity the communicating hosts should use virtual IP addresses.

Class: MovingTargetDefense (Main Application Class)

<u>Member variables:</u>

- ***Datapaths:*** This is a set which consists of all the datapath ID's
- ***R2V_Mappings:*** This is a dictionary which consists of real IP address as key and virtual IP address as value.
- ***V2R Mappings:*** This is also a dictionary which consists of virtual IP address as key and real IP address as value.
- ***HostAttachments:*** This is a dictionary which consists of real IP address as key and switch DPID as value. This gives us information about a host being connected to a switch.
- ***Resources:*** This is a resource pool which consists of available virtual IP addresses. mac_2_port: This is a dictionary which stores switch DPID as key and MAC to port mapping as value. This MAC to port mapping is again another dictionary.

<u>Methods:</u>

- ***start:*** we append a new thread which calls the function TimerEventGen. This is an infinite loop which keeps generating timeout events for every 30seconds.
- ***TimerEventGen:*** This method runs as an infinite loop and every 30seconds it generates timeout events by creating an object EventMessage and sends this event to all its listeners.
- ***init:*** This is the class constructor. It is used to initialize the member variables.
- ***handleSwitchFeatures:*** Handles the switch feature events sent by the switches to the controller the first time switch sends negotiation messages to the controller. We store the switch information in the member variable 'datapaths'. We also add table miss flow entry to the switch.
- ***EmptyTable:*** This helper function empties the flow table of a search. It sends flow_mod message to the switch telling it to empty the flow table.
- ***update_resources:*** Listens to the timeout events and updates real to virtual IP address mapping from the resource pool of available IP addresses. It also removes the flow rules from all the switches and adds a table-miss entry to all the switches.
- ***isRealIPAddress:*** Returns true if real IP address is provided.
- ***isVirtualIPAddress:*** Returns true if virtual IP address is provided.
- ***isDirectContact:*** Returns true if the provided IP address of the host is directly attached to the provided switch id. If there is no IP address to switch id mappings in the 'HostAttachments' dictionary it returns true by default. Else false is returned.
- ***add_flow:*** Adds flow rules to the switch based on match, actions, timeout provided.
- ***handlePacketInEvents:*** Handles incoming packets and implements Random Host Mutation technique by changing source and destination IP addresses of the incoming packets. Details of Random Host Mutation are listed as below:
  - Extract ARP object from the packet
  - Extract ICMP object from the packet
  - Based on which object is not null, packets are handled accordingly.
  - To implement a learning MTD there is a requirement to know to which switch a host is directly connected to.
  - The first step is to build a dictionary which has 'real IP addresses' as key and 'switch DPID' as value.

- Once this dictionary is filled, the MTD algorithm is as follows:
    - If the source address is real, change it to virtual
    - If the destination address is virtual, check if the switch is directly connected to the destination host. If yes, then change the virtual IP address to real IP address. Else, forward the packet to the next hop.
    - If the destination address is real, drop the packet if real IP address key exists in the 'HostAttachments' dictionary.
- Filling the dictionary 'HostAttachments' is implemented as a learning algorithm.
    - If destination IP address is virtual, check if the destination IP address is directly connected to the switch. At this stage since destination IP address key is not present in the 'HostAttachments', it will return true. So, the destination IP address is changed to real IP address and sent.
    - The next switch sees this packet with destination IP address being real, it then checks if an entry already exists in the 'HostAttachments' for destination IP address. This is false so the packet is allowed to pass.
    - Once the packet reaches the destination and the destination responds, this acts as the first packet with source address being real for that responding host. Hence we store the IP address to switch DPID mapping.
    - Thus, we have a mapping build for both source and destination.
    - Once the host attachment is build, the successive packet flow follows MTD as depicted above.
- Extract Ethernet object from the object.
- Using MAC address to port mapping of a switch, we build a learning switch and decide the output port.

  Initially the packets are flooded. But once the 'mac_to_port' dictionary is build, the packets are sent out through specific ports. Simultaneously, flow rules are installed with actions set to IP address changes and output ports.

```
import json
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller import event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.lib.packet import icmp
from ryu.lib.packet import arp
from ryu.lib.packet import ipv4
from ryu.lib import hub
from random import randint,seed
from time import time


#Custom Event for time out
class EventMessage(event.EventBase):
    '''Create a custom event with a provided message'''
    def __init__(self, message):
        print("Creating Event")
        super(EventMessage, self).__init__()
        self.msg=message

#Main Application
class MovingTargetDefense(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
    _EVENTS = [EventMessage]

R2V_Mappings={"10.0.0.1":"","10.0.0.2":"","10.0.0.3":"","10.0.0.4":"","10
.0.0.5":"","10.0.0.6":"","10.0.0.7":"","10.0.0.8":""}
    V2R_Mappings={}
    AuthorizedEntities=['10.0.0.1']
    Resources=["10.0.0.9","10.0.0.10","10.0.0.11","10.0.0.12",
           "10.0.0.13","10.0.0.14","10.0.0.15","10.0.0.16",
           "10.0.0.17","10.0.0.18","10.0.0.19","10.0.0.20",
           "10.0.0.21","10.0.0.22","10.0.0.23","10.0.0.24",
           "10.0.0.25","10.0.0.26","10.0.0.27","10.0.0.28",
           "10.0.0.29","10.0.0.30","10.0.0.31","10.0.0.32",
           "10.0.0.33","10.0.0.34","10.0.0.35","10.0.0.36"]
    def start(self):
        '''
            Append a new thread which calls the TimerEventGen function
which generates timeout events
            every 30 seconds & sends these events to its listeners
            Reference: https://sourceforge.net/p/ryu/mailman/ryu-
devel/?viewmonth=201601&viewday=12
        '''
        super(MovingTargetDefense,self).start()
        self.threads.append(hub.spawn(self.TimerEventGen))

    def TimerEventGen(self):
        '''
            A function which generates timeout events every 30 seconds
            & sends these events to its listeners
            Reference: https://sourceforge.net/p/ryu/mailman/ryu-
devel/?viewmonth=201601&viewday=12
        '''
        while 1:
            self.send_event_to_observers(EventMessage("TIMEOUT"))
            hub.sleep(30)

    def __init__(self, *args, **kwargs):
        '''Constructor, used to initialize the member variables'''
        super(MovingTargetDefense, self).__init__(*args, **kwargs)
        self.mac_to_port = {}
        self.datapaths=set()
        self.HostAttachments={}
        self.offset_of_mappings=0

    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def handleSwitchFeatures(self, ev):
        '''
            Handles switch feature events sent by the switches to the
controller
            the first time switch sends negotiation messages.
            We store the switch info to the datapaths member variable
            & add table miss flow entry to the switches.

            #Reference: Simple_Switch
            #http://ryu.readthedocs.io/en/latest/writing_ryu_app.html
        '''
        datapath = ev.msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        self.datapaths.add(datapath);
        # install table-miss flow entry
        match = parser.OFPMatch()
        actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
                                          ofproto.OFPCML_NO_BUFFER)]
        self.add_flow(datapath, 0, match, actions)

    def EmptyTable(self,datapath):
        '''
            Empties flow table of a switch!
            Remove Flow rules from switches
            Reference:
https://sourceforge.net/p/ryu/mailman/message/32333352/
        '''
        ofProto=datapath.ofproto
        parser = datapath.ofproto_parser
        match=parser.OFPMatch()

flow_mod=datapath.ofproto_parser.OFPFlowMod(datapath,0,0,0,ofProto.OFPFC_
DELETE,0,0,1,ofProto.OFPCML_NO_BUFFER,ofProto.OFPP_ANY,ofProto.OFPG_ANY,0
,match=match,instructions=[])
```

```
        datapath.send_msg(flow_mod)

    #Listen to timeout & update the mappings
    @set_ev_cls(EventMessage)
    def update_resources(self,ev):
        '''
            Listen to the Time-out event & update the real-virtual IP
address mappings from the resources
            Also remove the flow rules from all the switches.
            & Add a default, table-miss entry to all the switches.

        '''
        '''seed function is used initialize random number generator. The
current system time is seeded to
        obtain different set of random numbers every time the function
runs.'''
        seed(time())
        pseudo_ranum = randint(0,len(self.Resources)-1) #randint returns
a random integer in the range of 0 and len(Resources)-1
        print ("Random Number:",pseudo_ranum)

        for keys in self.R2V_Mappings.keys():
            #Virtual IP address are assigned to each host from the pool
of Resources starting from (pseudo_ranum)th index
            self.R2V_Mappings[keys]=self.Resources[pseudo_ranum]
            #pseudo_ranum is updated to point to next index. If the index
is overshooted from the Resources pool, it is looped back to point to 0th
index
            pseudo_ranum=(pseudo_ranum+1)%len(self.Resources)
        self.V2R_Mappings = {v: k for k, v in self.R2V_Mappings.items()}
        print "**********", self.R2V_Mappings,"***********"
        print "**********", self.V2R_Mappings,"***********"
        '''
            Reference:
https://sourceforge.net/p/ryu/mailman/message/32333352/
            How to remove flowrules from switches
        '''
        for curSwitch in self.datapaths:
            #Remove all flow entries
            parser = curSwitch.ofproto_parser
            match=parser.OFPMatch()
            flowModMsg=self.EmptyTable(curSwitch)
            #Add default flow rule
            ofProto=curSwitch.ofproto
            actions = [parser.OFPActionOutput(ofProto.OFPP_CONTROLLER,
                                              ofProto.OFPCML_NO_BUFFER)]
            self.add_flow(curSwitch, 0, match, actions)

    def isRealIPAddress(self,ipAddr):
        '''Returns True id IP address is real'''
        if ipAddr in self.R2V_Mappings.keys():
            return True

    def isVirtualIPAddress(self,ipAddr):
        ''' Returns True if the IP address is virtual'''
        if ipAddr in self.R2V_Mappings.values():
            return True

    '''def isAuthorizedEntity(self,ipAddr):
        if ipAddr in self.AuthorizedEntities:
            return True'''

    def isDirectContact(self,datapath,ipAddr):
        '''
            Return true if the IP addr host is directky connected to the 
switch given
            Also assumes that the host is directly connected if it has no 
information in the hostAttachments Table
        '''
        if ipAddr in self.HostAttachments.keys():
            if self.HostAttachments[ipAddr]==datapath:
                return True
            else:
                return False
        return True


    def add_flow(self, datapath, priority, match, actions, 
buffer_id=None, hard_timeout=None):
        '''
            Adds flow rules to the switch 
            Reference: Simple_Switch
            http://ryu.readthedocs.io/en/latest/writing_ryu_app.html
        '''
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                             actions)]
        if buffer_id :
            if hard_timeout==None:
                mod = parser.OFPFlowMod(datapath=datapath, 
buffer_id=buffer_id,
                                    priority=priority, match=match,
                                    instructions=inst)
            else:
                mod = parser.OFPFlowMod(datapath=datapath, 
buffer_id=buffer_id,
                                    priority=priority, match=match,
                                    instructions=inst, 
hard_timeout=hard_timeout)
        else:
            if hard_timeout==None:
                mod = parser.OFPFlowMod(datapath=datapath, 
priority=priority,
                                    match=match, instructions=inst)
            else:
                mod = parser.OFPFlowMod(datapath=datapath,
priority=priority,
                                    match=match, instructions=inst,
hard_timeout=hard_timeout)
        datapath.send_msg(mod)

    #Packet Handler ICMP & ARP
    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def handlePacketInEvents(self, ev):
        '''
            Handles Incoming Packets & implements Random Host mutation
technique
            by changing src & dst IP addresses of the incoming packets.
            Some part of the code is inspired by Simple_Switch
            http://ryu.readthedocs.io/en/latest/writing_ryu_app.html
        '''
        actions=[]
        pktDrop=False


        if ev.msg.msg_len < ev.msg.total_len:
            self.logger.debug("packet truncated: only %s of %s bytes",
                              ev.msg.msg_len, ev.msg.total_len)

        msg = ev.msg
        datapath = msg.datapath
        dpid = datapath.id
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        in_port = msg.match['in_port']
        pkt = packet.Packet(msg.data)
        arp_Obj=pkt.get_protocol(arp.arp)# Extract ARP object from packet
        icmp_Obj=pkt.get_protocol(ipv4.ipv4)# Extract ICMP object packet

        if arp_Obj:
            '''Handles ARP packets'''
            src=arp_Obj.src_ip
            dst=arp_Obj.dst_ip

            '''
                To Implement a Learning MTD, there is a need to know, to
which switch, the host is directly connected to.
                So the first time an ARP packet comes in who's src
address is real, we store the IP addr-Switch DPID mapping
                into the member variable HostAttachments.
            '''
            if self.isRealIPAddress(src) and src not in
self.HostAttachments.keys():
                self.HostAttachments[src]=datapath.id

            '''
                Learning MTD implementation
                if src is real change it to virtual no matter wat.
                if dest doesn't have a mapping in my table change to real
and flood.
                    This happens only for the first time when we donot
know
                    to which switch, the destination host is directly
connected to.
                if dst is virtual check if dest is directly connected
then change it to real
                else let it pass unchanged.
            '''

            if self.isRealIPAddress(src):

match=parser.OFPMatch(eth_type=0x0806,in_port=in_port,arp_spa=src,arp_tpa
=dst)
                spa = self.R2V_Mappings[src]
                print("Changing SRC REAL IP "+src+"---> Virtual SRC IP
"+spa)
                actions.append(parser.OFPActionSetField(arp_spa=spa))

            if self.isVirtualIPAddress(dst):
                match=
parser.OFPMatch(eth_type=0x0806,in_port=in_port,arp_tpa=dst,arp_spa=src)
                if
self.isDirectContact(datapath=datapath.id,ipAddr=self.V2R_Mappings[dst]):
                    keys = self.V2R_Mappings.keys()
                    tpa = self.V2R_Mappings[dst]
                    print("Changing DST Virtual IP "+dst+"---> REAL DST
IP "+tpa)
                    actions.append(parser.OFPActionSetField(arp_tpa=tpa))

            elif self.isRealIPAddress(dst):
                '''Learn MTD From Flood'''

match=parser.OFPMatch(eth_type=0x0806,in_port=in_port,arp_spa=src,arp_tpa
=dst)
                if not
self.isDirectContact(datapath=datapath.id,ipAddr=dst):
                    pktDrop=True
                    print "Dropping from",dpid
            else:
                pktDrop=True
        elif icmp_Obj:
            '''Handles ICMP packets'''
            print("ICMP PACKET FOUND!")
            src=icmp_Obj.src
            dst=icmp_Obj.dst

            if self.isRealIPAddress(src) and src not in
self.HostAttachments.keys():
                self.HostAttachments[src]=datapath.id

            '''
                Learning MTD implementation
                if src is real change it to virtual no matter wat.
                if dest doesn't have a mapping in my table change to real
and flood.
                    This happens only for the first time when we donot
know
                    to which switch, the destination host is directly
connected to.
                if dst is virtual check if dest is directly connected
then change it to real
                else let it pass unchanged.
            '''

            if self.isRealIPAddress(src):
                match=
parser.OFPMatch(eth_type=0x0800,in_port=in_port,ipv4_src=src,ipv4_dst=dst
)
                ipSrc = self.R2V_Mappings[src]
                print("Changing SRC REAL IP "+src+"---> Virtual SRC IP
"+ipSrc)
                actions.append(parser.OFPActionSetField(ipv4_src=ipSrc))
            if self.isVirtualIPAddress(dst):
                #print self.HostAttachments
                match=
parser.OFPMatch(eth_type=0x0800,in_port=in_port,ipv4_dst=dst,ipv4_src=src
)
                if
self.isDirectContact(datapath=datapath.id,ipAddr=self.V2R_Mappings[dst]):
                    ipDst = self.V2R_Mappings[dst]
                    print("Changing DST Virtual IP "+dst+"---> Real DST
IP "+ipDst)

actions.append(parser.OFPActionSetField(ipv4_dst=ipDst))

            elif self.isRealIPAddress(dst):
                '''Learn From Flood'''

match=parser.OFPMatch(eth_type=0x0806,in_port=in_port,arp_spa=src,arp_tpa
=dst)
                if not
self.isDirectContact(datapath=datapath.id,ipAddr=dst):
                    pktDrop=True
                    print "Dropping from",dpid
            else:
                pktDrop=True

        '''Extract Ethernet Object from packet'''
        eth = pkt.get_protocols(ethernet.ethernet)[0]
        dst = eth.dst
        src = eth.src
        '''Store the incoming packet source address, switch & the port
combination to be used to learn the packet switching'''
        self.mac_to_port.setdefault(dpid, {})
        self.logger.info("packet in %s %s %s %s", dpid, src, dst,
in_port)

        '''learn a mac address to avoid FLOOD next time.'''

        self.mac_to_port[dpid][src] = in_port
        '''Learning Mac implemention to avoid flood'''
        if dst in self.mac_to_port[dpid]:
            out_port = self.mac_to_port[dpid][dst]
        else:
            out_port = ofproto.OFPP_FLOOD
        if dst in self.mac_to_port[dpid]:
            out_port = self.mac_to_port[dpid][dst]
        else:
            out_port = ofproto.OFPP_FLOOD
        '''Append the outport action to the action set'''
        if not pktDrop:
            actions.append(parser.OFPActionOutput(out_port))
        '''install a flow to avoid packet_in next time'''
        if out_port != ofproto.OFPP_FLOOD:
            '''
                verify if we have a valid buffer_id, if yes avoid to send
both flow_mod & packet_out
                Install Flow rules to avoid the packet in message for
similar packets.
            '''
            if msg.buffer_id != ofproto.OFP_NO_BUFFER:
                self.add_flow(datapath, 1, match, actions,msg.buffer_id)
                return
            else:
                self.add_flow(datapath, 1, match, actions)
        data = None
        if msg.buffer_id == ofproto.OFP_NO_BUFFER:
            data = msg.data
        '''
            Build a packet out message & send it to the switch with the
action set,
            Action set includes all the IP addres changes & out port
actions.
        '''
        out = parser.OFPPacketOut(datapath=datapath,
buffer_id=msg.buffer_id,
                                  in_port=in_port, actions=actions,
data=data)
        '''Send the packet out message to the switch'''
        datapath.send_msg(out)
```