International Journal of Innovative Research in Engineering
Volume 3, Issue 3 (May-June 2022), PP: 37-42.
www.theijire.com
ISSN No: 2582-8746

Mitigation Policy for Unauthorized Access of Traffic in SDN

Sachin Darekar[1], Aayush Agarwal[2] , Usman Chougule[3], Abhay Donde[4]
[1,2,3,4]*Information Technology, Bharati Vidyapeeth College of Engineering, Mumbai University, India.*

How to cite this paper:
Sachin Darekar[1], Aayush Agarwal[2] , Usman Chougule[3], Abhay Donde[4], "Mitigation Policy for Unauthorized Access Of Traffic In SDN ", IJIRE-V3I03-37-42.

Abstract: *Many have recognized the requirement to reconstitute the present web work into a way additional dynamic center networks face the challenge of maintaining firewalls up thus far. it's troublesome for today's inflexible infrastructure to deal with the quick dynamical demands of the users. As a result, Software-Defined Network (SDN) was introduced around 2005 to remodel today's network to possess centralized management, fast innovation, and programmability by decoupling the management and information planes. This study focuses on developing a firewall application that runs over associate degree OpenFlow-based SDN controller to point out that almost all of the firewall functionalities area unit able to be engineered on computer code, while not the help of a zealous hardware. Among several OpenFlow controllers that exist already for the general public we've chosen Ryu written in Python for the project and to make the SDN topology, we've used VirtualBox and Mininet. during this study, we have a tendency to cowl the implementation detail of our firewall application, moreover because the experimentation result.*

Key Word: *Based firewall, Mininet emulation tool, OpenFlow protocol, Software Defined Networking (SDN), RY, Python.*

I. INTRODUCTION

Software declared Network could be a developing space of the analysis within the sector of networking. Firewalls area unit one in all the foremost necessary elements employed in networks, and new provocation has been driven by the software-defined networks in implementing firewalls. the most drawback of the firewall is its speed. The speed of the firewall provides delay; usually firewall link speeds area unit slower than the supported network interface and might cause the traffic burst from the host to be buffered till packets area unit processed. to beat these things, our project aims to resolve approaching speed-related issues by implementing duplicate instances of the firewall. By coming up with 2 topologies, single and multiple controllers, and implementing them in a very replicated atmosphere. Multiple controllers in a very network atmosphere tend to point out improved performance with the exaggerated variety of attacks, firewalls also are changing into slower and additional liable to lags and firewall explosions. With the evolution of Software-Defined Networking (SDN), it's become troublesome to implement network elements together with firewalls in ancient networks. that's why Associate in Nursing upgrade is important to adapt to new changes. code firewall elements organized from one controller is extremely helpful, creating it straightforward to line rules round the network.

Procedure methodology

A) SDN TECHNOLOGY

Software- defined networking (SDN) technology is an approach to network operation that enables programmatic structured network configuration to ameliorate network performance and monitoring making it more like pall computing than traditional network operation. SDN is supposed to deal with the very fact that the static armature of traditional networks is decentralized and sophisticated while current networks bear further inflexibility and straightforward troubleshooting. SDN attempts to polarize network intelligence in one network element by disuniting the forwarding process of network packets (data aeroplane) from the routing process (control aeroplane). The control aeroplane consists of 1 or further regulators which are considered because of the brain of the SDN network where the entire intelligence is incorporated.

1) Features of SDN

- Programmatically Configured SDN lets network directors configure, manage, secure, and optimize network coffers veritably snappily via dynamic, automated SDN programs, which they can write themselves because the programs don't depend on personal software.
- Directly programmable Network control is directly programmable because it's severed from encouraging functions. Nimble Abstracting control from forwarding lets directors stoutly acclimate network-wide business inflow meet changing requirements.
- Centrally Managed Network intelligence is (logically) consolidated in software- grounded SDN regulators that maintain a global view of the network, which appears to operations and policy machines as a single, logical switch.

- Open Norms- grounded and seller-neutral When enforced through open norms, SDN simplifies network design and operation because instructions are handed by SDN regulators rather of multiple, seller-specific bias and protocols

. B) Open Flow Protocol

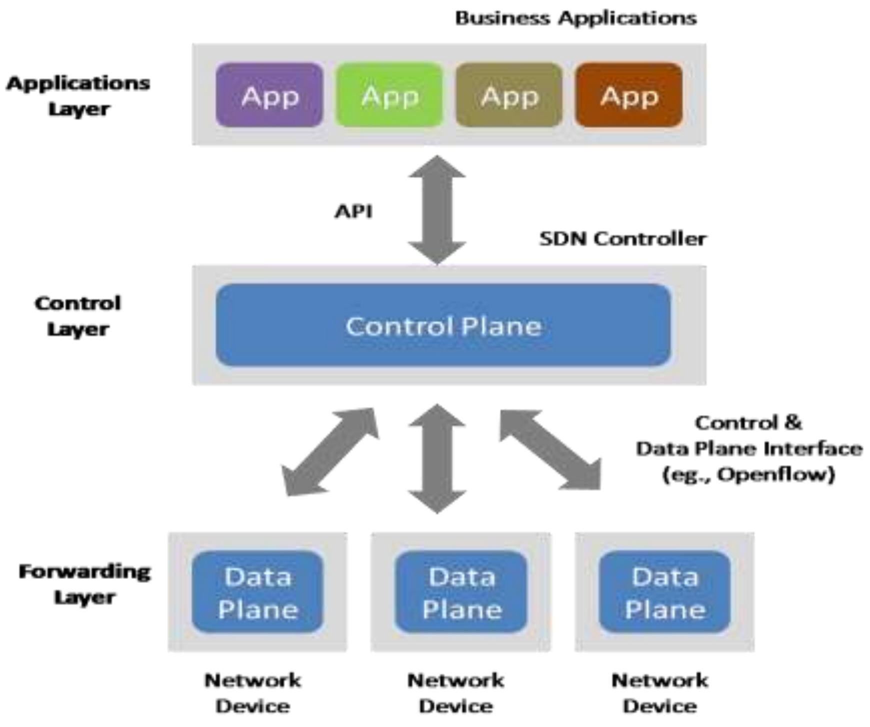
OpenFlow enables network regulators to determine the path of network packets across a network of switches. The regulators are distinct from the switches. This separation of the control from the forwarding allows for further sophisticated business operation than is doable using access control lists (ACLs) and routing protocols. Also, OpenFlow allows switches from different merchandisers frequently each with their own personal interfaces and scripting languages to be managed ever using a single, open protocol. The protocol's formulators consider OpenFlow an enabler of software- defined networking (SDN).

1) Features of OpenFlow

• Protocols other than OpenFlow are:
• Border Gateway Protocol(BGP)
• Netconf
• Extensible Messaging and Presence Protocol(XMPP)
• Open vSwitch Database Management Protocol(OVSDB)
• MPLS Transport Profile(MPLS-TP)

2) Abbreviations and Acronyms Advantages of OpenFlow over above mentioned protocols

- OpenFlow- grounded SDN creates inflexibility in how the network is used, operated, and vended. The software that governs it can be written by enterprises and service providers using ordinary software surroundings.
- OpenFlow- grounded SDN enables virtualization of the network, and thus the integration of the network with computing and storehouse. This allows the entire IT operation to be governed more sleekly with a single standpoint and toolset



C). RYU Controller

Ryu Controller is an open, software- defined networking (SDN) Controller designed to increase the dexterity of the network by making it easy to manage and acclimatize how business is handled. In general, the SDN Controller is the brain of the SDN terrain, communicating information down to the switches and routers with southbound APIs, and over to the operations and business sense with northbound APIs. The Ryu Controller is supported by NTT and is stationed in NTT cloud data centers as well.

The Ryu Controller provides software factors, with well- defined operation program interfaces (APIs), that make it easy for inventors to produce new network operation and control operations. This element approach helps associations customize deployments to meet their specific requirements; inventors can snappily and fluently modify being factors or apply their own to insure the underpinning network can meet the changing demands of their operations.

Open Flow Architecture

1) Features of RYU

Controllers available other than RYU are:

- Open Daylight
- NOX
- Beacon
- POX

2) Advantages of RYU over above mentioned controllers are:

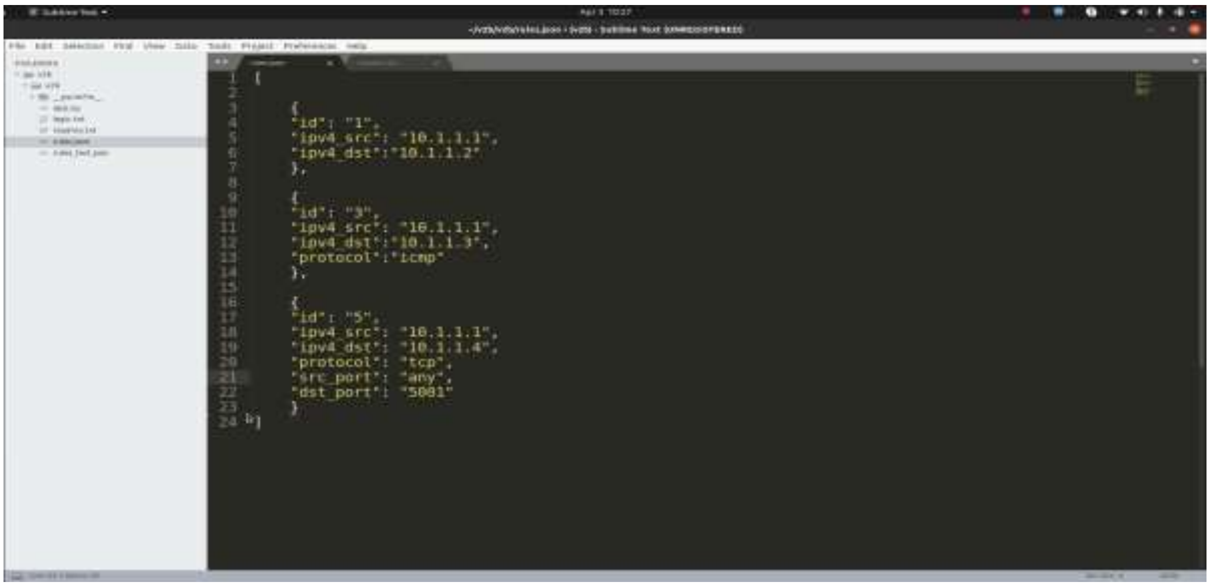
- Ryu is distributed with multiple operations similar as simple switch, router, insulation, firewall, GRE lair, topology, VLAN, etc.
- Ryu operations are single- threaded realities, which apply colorful functionalities. Ryu operations shoot asynchronous events to each other.
- Each Ryu operation has a admit line for events, which is substantially FIFO to save the order of events. In addition, each operation includes a thread for processing events from the line.
- The thread's main circle pops out events from the admit line and calls the applicable event tutor. Hence, the event tutor is called within the environment of the event-processing thread, which works in a blocking fashion, i.e., when an event tutor is given control, no farther events for the Ryu operation will be reused until control is returned.

II. IMPLEMENTATION

I) Access Control List : In this we are adding reverse access control list automatically. This means when we create the roots .json file the source and destination changes automatically and vice versa. So reverse flow for ACL is automatically added.
Rule Decoding Logic : For this the special keyword "any " is used in Ip address and port.

1. When the application start it reads the roots .json file and populates the rules in ACL Manager Object.
2. In the controller it will explicitly add flow for the rules and action is send back to controller so that when one generate the traffic the packet get backs to the controller.
3. When the packet comes to controller it will try to do the switching, forwarding and then it will forward it to the respective ports.
4. Here priority of the packets plays very important role
   - ARP _PRIORITY =100
   - ACLRULE_PRIORITY =11
   - DENY_PRIORITY=10
   - TRAFFIC_PRIORITY=50

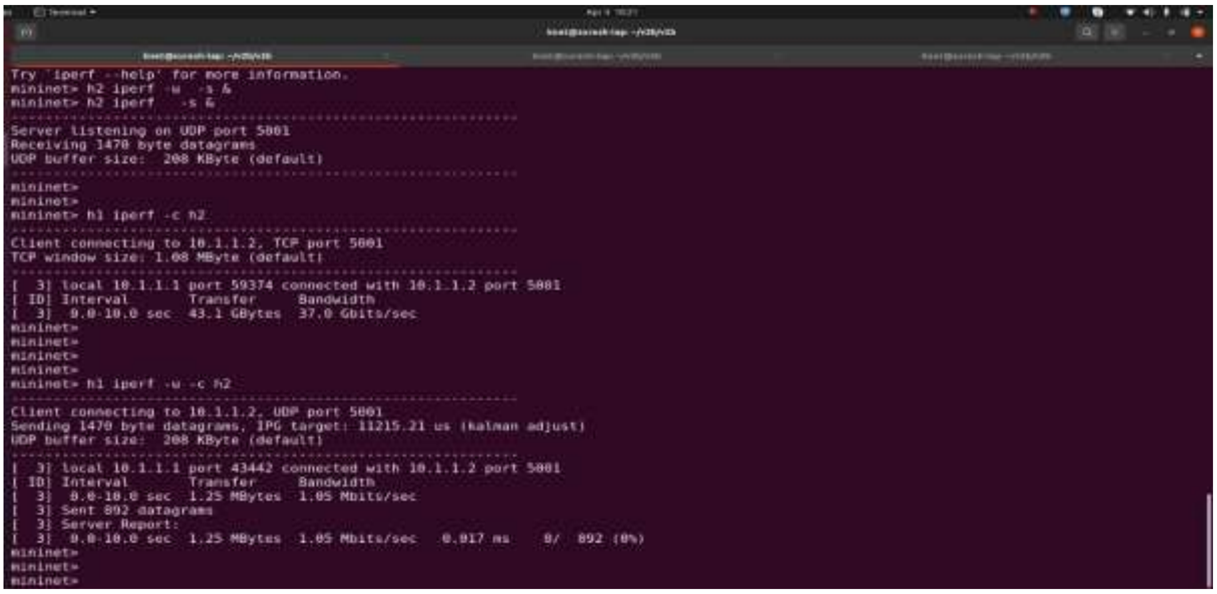
Set up the mininet and ryu controller. Access control list is the mini terminology that defines the rules for what traffic to be allowed and what traffic should not be allowed. Default behavior of access control will deny everything. To allow the access control we explicitly use scrip dstip protocol, port number. Only those traffic will be allowed in the network those are in the allow category. Any other traffic will be denied, which the basic fundamental of the access control list.
Here in this project we have defined the rules of access control in Jason. We allow the traffic flow from source to destination. We can also allow any traffic like icmp, tcp, udp. We can allow a specific traffic by mentioning them as protocol icmp, tcp,udp. For high level granularity we can include all the 5 tuples like source IP, destination IP, TCP protocol, source port and destination port. For id 5 we have given the h1 to h4 only tcp is allowed if destination port is 5000 source port can be any.



```
[

    {
    "id": "1",
    "ipv4_src": "10.1.1.1",
    "ipv4_dst":"10.1.1.2"
    },

    {
    "id": "3",
    "ipv4_src": "10.1.1.1",
    "ipv4_dst":"10.1.1.3",
    "protocol":"icmp"
    },

    {
    "id": "5",
    "ipv4_src": "10.1.1.1",
    "ipv4_dst": "10.1.1.4",
    "protocol": "tcp",
    "src_port": "any",
    "dst_port": "5001"
    }
]
```

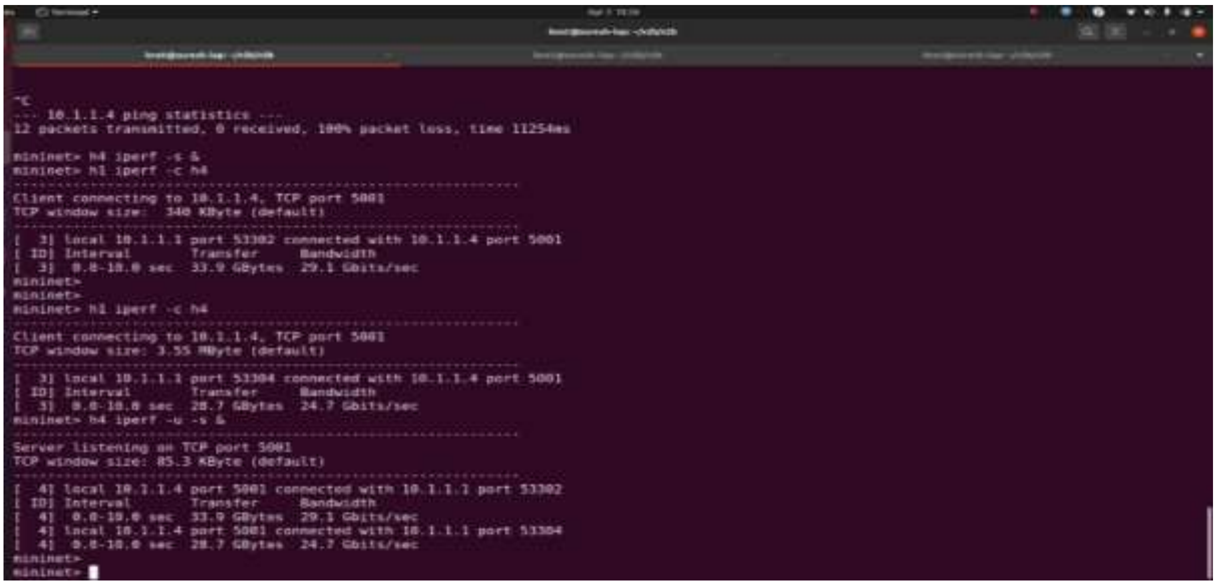
Syntax for the example “id:”5 will be:
H1 to h4
H1 (tcp, 10001, 5000) to h4
H4 tcp, 5000, 10001 to h1
Here source port ie 10001 can be anything. Where destination port is fixed ie 5000.
How to test:
1) sudo mn-controller = remote.ip=127.0.0.1-mac -1 10.1.1.0/24
   --switch=ovsk,protocols=OpenFlow13 --topo= single,6
2) ryu-manager app.py
3) Check the flows:
   sudo ovs-ofctl -0 OpenFlow13 dump-flows sl
4) Testing:
a) try pingall in mininet.
   only h1 to h2, hi to h3 will ping..
b) run the iperf server in h4
   h4 iperf -s &
   h1 iperf -c h4
   this will be passed
c) try other traffic it will be blocked

A) ping h1 to h2 with not restrictions result output.



```
Try 'iperf --help' for more information.
mininet> h2 iperf -u  -s &
mininet> h2 iperf   -s &
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
mininet>
mininet>
mininet> h1 iperf -c h2
------------------------------------------------------------
Client connecting to 10.1.1.2, TCP port 5001
TCP window size: 1.08 MByte (default)
------------------------------------------------------------
[  3] local 10.1.1.1 port 59374 connected with 10.1.1.2 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  43.1 GBytes  37.0 Gbits/sec
mininet>
mininet>
mininet>
mininet>
mininet> h1 iperf -u -c h2
------------------------------------------------------------
Client connecting to 10.1.1.2, UDP port 5001
Sending 1470 byte datagrams, IPG target: 11215.21 us (kalman adjust)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 10.1.1.1 port 43442 connected with 10.1.1.2 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec
[  3] Sent 892 datagrams
[  3] Server Report:
[  3]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec   0.017 ms    0/  892 (0%)
mininet>
mininet>
mininet>
```

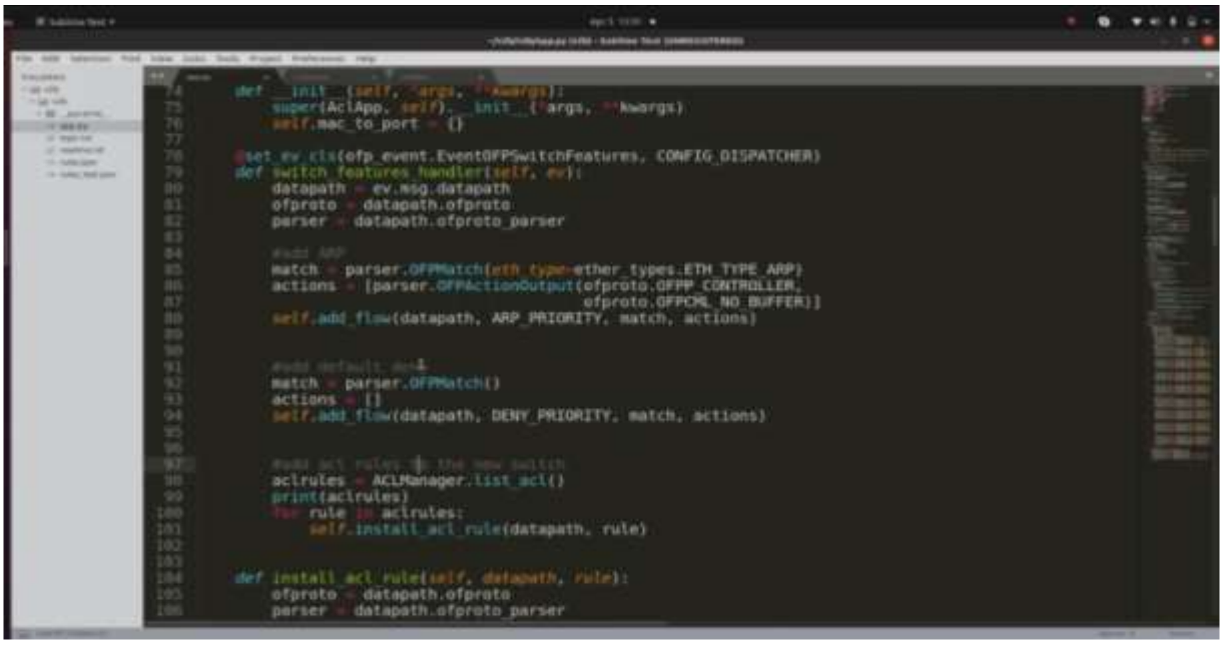
B) ping h1 to h4 with 5 tuples result output:



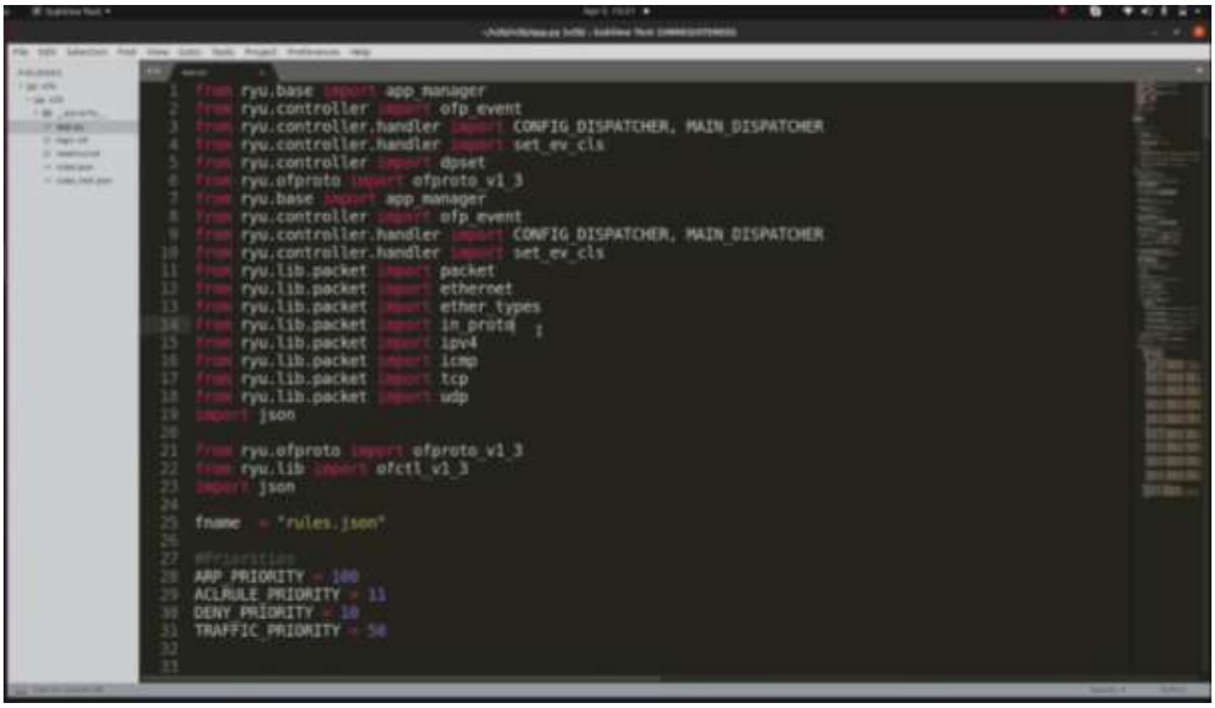
```
^C
--- 10.1.1.4 ping statistics ---
12 packets transmitted, 0 received, 100% packet loss, time 11254ms

mininet> h4 iperf -s &
mininet> h1 iperf -c h4
------------------------------------------------------------
Client connecting to 10.1.1.4, TCP port 5001
TCP window size:  340 KByte (default)
------------------------------------------------------------
[  3] local 10.1.1.1 port 53302 connected with 10.1.1.4 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  33.9 GBytes  29.1 Gbits/sec
mininet>
mininet>
mininet> h1 iperf -c h4
------------------------------------------------------------
Client connecting to 10.1.1.4, TCP port 5001
TCP window size: 3.55 MByte (default)
------------------------------------------------------------
[  3] local 10.1.1.1 port 53304 connected with 10.1.1.4 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  28.7 GBytes  24.7 Gbits/sec
mininet> h4 iperf -u -s &
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  4] local 10.1.1.4 port 5001 connected with 10.1.1.1 port 53302
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.0 sec  33.9 GBytes  29.1 Gbits/sec
[  4] local 10.1.1.4 port 5001 connected with 10.1.1.1 port 53304
[  4]  0.0-10.0 sec  28.7 GBytes  24.7 Gbits/sec
mininet>
mininet>
```

III. RESULT

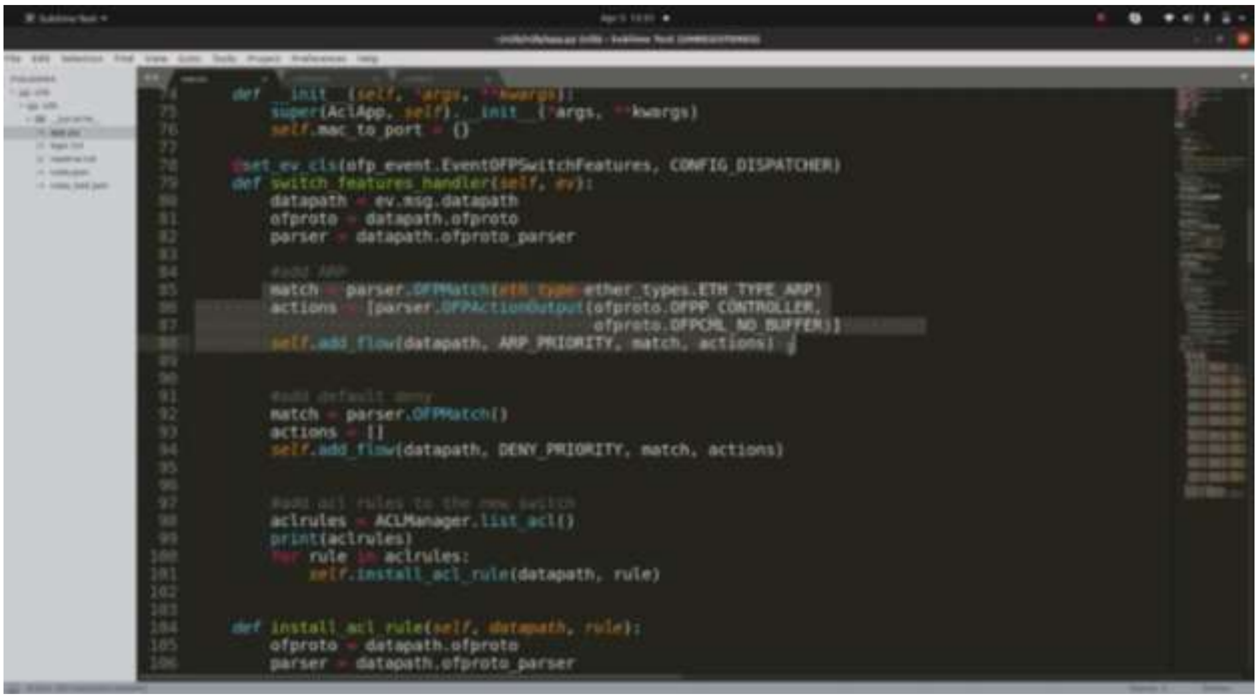
Data Traffic here output will be given to port 1 where the real forwarding happens.
When we start the manager application the trigger object is created the trigger in this is entropy this means we here do the self math to empty the dictionary. For switching purpose we are creating this data structure. For this access control class is created in this class we open the root.json file and copy all rules in self. Acl . For the first rule swapping is done between port to destination and vice versa. When this process will be done the ACL will expand. At the end of this we will have three more rules in access control manager.
In this process everything is handled through events this is how review controller works so this event get called at the time where topology starts when switch is getting registered. Here we are adding all the default rules. ARP packets are sent to controller, here whatever we define is the highest priority. Secondary thing is here we are adding deny traffic that is an empty list. If the match is empty then the action is empty so in this we are adding packet using deny priority. In rule 3 we are completing all work we have listed in ACL manager in this we are listing the objects so that all objects are gathered in ACL manager. For this rule to work we first need to create a match. The match here is a form of object . When we create a topology it tries to register with the switch all three flows will be installed and action will be sent to controller. When we initiate the ping the packet comes to packet handler if all the ACL rules are passed. The controller here is not deciding what to be allowed or what to be denied because controller already has some predefined rules which is deciding this traffic the rules are under the installer. Once packet reaches to controller there Is simple switching this switching is based on rule 4



```
    def __init__(self, *args, **kwargs):
        super(AclApp, self).__init__(*args, **kwargs)
        self.mac_to_port = {}

    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        datapath = ev.msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        #add ARP
        match = parser.OFPMatch(eth_type=ether_types.ETH_TYPE_ARP)
        actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
                                          ofproto.OFPCML_NO_BUFFER)]
        self.add_flow(datapath, ARP_PRIORITY, match, actions)


        #add default deny
        match = parser.OFPMatch()
        actions = []
        self.add_flow(datapath, DENY_PRIORITY, match, actions)


        #add acl rules to the new switch
        aclrules = ACLManager.list_acl()
        print(aclrules)
        for rule in aclrules:
            self.install_acl_rule(datapath, rule)


    def install_acl_rule(self, datapath, rule):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
```



```
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.controller import dpset
from ryu.ofproto import ofproto_v1_3
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.lib.packet import ether_types
from ryu.lib.packet import in_proto
from ryu.lib.packet import ipv4
from ryu.lib.packet import icmp
from ryu.lib.packet import tcp
from ryu.lib.packet import udp
import json

from ryu.ofproto import ofproto_v1_3
from ryu.lib import ofctl_v1_3
import json

fname  = "rules.json"

#Priorities
ARP_PRIORITY = 100
ACLRULE_PRIORITY = 11
DENY_PRIORITY = 10
TRAFFIC_PRIORITY = 50
```



```
    def __init__(self, *args, **kwargs):
        super(AclApp, self).__init__(*args, **kwargs)
        self.mac_to_port = {}

    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        datapath = ev.msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        #add ARP
        match = parser.OFPMatch(eth_type=ether_types.ETH_TYPE_ARP)
        actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
                                          ofproto.OFPCML_NO_BUFFER)]
        self.add_flow(datapath, ARP_PRIORITY, match, actions)


        #add default deny
        match = parser.OFPMatch()
        actions = []
        self.add_flow(datapath, DENY_PRIORITY, match, actions)


        #add acl rules to the new switch
        aclrules = ACLManager.list_acl()
        print(aclrules)
        for rule in aclrules:
            self.install_acl_rule(datapath, rule)


    def install_acl_rule(self, datapath, rule):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
```

IV. CONCLUSION

The scalability features of the Ryu controller by implementing two scenarios in simulation experimental environment. In this paper, the authors have provided a clear idea of how to create an experimental test with an analysis of obtained statistical results keeping the performance as the central focus. We would conclude this paper by providing additional options for the implementation of SDN in simulation and emulation environment to inspire researchers to ideate and practically implement their ideas in the form of simulations to come up with contributions pushing the technology ahead.

References

[1]. *Mohamed G. Gouda and Alex X. Liu1 "A model of stateless firewalls and its properties" 2005 International Conference on Dependable System and Networks (DSN'05), ISSN:* **2158-3927**

[2]. *Mujahid Ali, Nadir Shah, Muazzam A Khan Khattak " Dynamic ACL Policy Implementation for Software-Defined Networking," 2020 IEEE 17th International Conference on Smart Communities: Improving Quality of Life Using ICT, IoT and AI (HONET). ISSN: 1949-4092 0l*

[3]. *Rashid Amin, Nadir Shah, Babar Shah, Omar Alfandi " Auto-configuration of ACL policy in case of topology change in Hybrid SDN " IEEE Access (Volume: 4)* **ISSN:** *2169-3536*

[4]. *Serqey Morzhov, Iqor Alekseev, Mikhail Nikitinskiy, " Firewall application for Floodlight SDN controller " 2016 International Siberian Conference on Control and Communications (SIBCON), ISSN: 2380-6516.*

[5]. *Tao Feng, Jun Bi " OpenRouteFlow: Enable Legacy Router as a Software-Defined Routing Service for Hybrid SDN " 2015 24th International Conference on Computer Communication and Networks (ICCCN.* Print ISSN: *1095-2055*

[6]. *Chisong Li Zhensong Liao " An extended ACL for solving authorization conflicts "2009 Second International Symposium on Electronic Commerce and Security "ISBN:978-0-7695-3643-9"*

[7]. *Ruxandra Trandafir, Mihai Carabas, Razvan Rughinis, Nicolae Tapus " FirewallPK: Security tool for centralized Access Control List Management "2014 RoEduNet Conference 13th Edition: Networking in Education and Research Joint Event RENAM 8th Conference " ISSN: 2068-1038.*

[8]. *Kallol Krishna Karmakar, Vijay Varadharajan, Udaya Tupakula " On the Design and Implementation of a Security Architecture for Software Defined Networks " 2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), ISBN:978-1-5090-4297-5.*

[9]. *Madjed BENCHEIKH LEHOCINE Mohamed BATOUCHE " Flexibility of Manaqinq VLAN Filtering and Segmentation in SDN Networks " 2017 International Symposium on Networks, Computers and Communications (ISNCC). ISBN:978-1-5090-4260-9.*

[10]. *Ding-Fong Huang, Chien Chen Mahadevan Thanavel. " Fast Packet Classification on OpenFlow Switches Using Multiple R*-Tree Based Bitmap Intersection " NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium. ISSN: 2374-9709.*