

**Bachelor of Science in Computer Science and Engineering**

**Detection and Prevention of ARP Spoofing using  
Dynamic Port and State Allocation.**

**Anika Akhtar Lima**

**ID: 1404058**

**July,2019**

**Department of Computer Science & Engineering  
Chittagong University of Engineering & Technology  
Chittagong-4349,Bangladesh**

# **Detection and Prevention of ARP Spoofing using Dynamic Port and State Allocation.**



This thesis is submitted in partial fulfilment of the requirement for the degree of bachelor of Science in Computer Science & Engineering.

Anika Akhtar Lima

ID:1404058

Supervised by

Dr. Asaduzzaman

Professor

Department of Computer Science & Engineering (CSE)

Chittagong University of Engineering & Technology(CUET)

**Department of Computer Science & Engineering**

**Chittagong University of Engineering & Technology**

**Chittagong-4349,Bangladesh**

The thesis titled “Detection and Prevention of ARP Spoofing using Dynamic Port and State Allocation” submitted by Roll No. 1404058 , session 2018-2019 has been accepted as satisfactory in fulfilment of the requirement for the degree of Bachelor of Science in Computer Science & Engineering (CSE) as B.Sc. Engineering to be awarded by the Chittagong University of Engineering & Technology (CUET).

## **Board of Examiners**

1.....

Chairman

Dr. Asaduzzaman

Professor

Department of Computer Science & Engineering(CSE)

Chittagong University of Engineering & Technology (CUET).

2. ....

Member

Professor Dr. Asaduzzaman

(Ex-officio)

Head

Department of Computer Science & Engineering(CSE)

Chittagong University of Engineering & Technology (CUET).

3. ....

Member

Dr. Md. Mokammel Haque

(External)

Professor

Department of Computer Science & Engineering(CSE)

Chittagong University of Engineering & Technology (CUET)

## **Statement of Originality**

It is hereby declared that the contents of this project is original and any part of it has not been submitted elsewhere for the award of any degree or diploma.

.....

**Signature of the Candidate**

**Date:**

## **Acknowledgement**

First and foremost, I would like to thank Almighty Allah for giving me the strength to finish this work. The satisfaction that accompanies the successful completion of this thesis would be incomplete without the mention of people whose ceaseless co-operation made it possible, whose constant guidance and encouragement crown all efforts with success. I am grateful to my honorable project Supervisor Professor Dr.Asaduzzaman, honorable head and professor of the Department of Computer Science and Engineering, Chittagong University of Engineering and Technology, for the guidance, inspiration and constructive suggestions which were helpful in the preparation of this project. Without his guidance it would not be possible for me to complete this task. I would also like to extend my gratitude to all of my teachers for their valuable guidance. I also convey special thanks and gratitude to Dr. Md. Mokammel Haque, professor, Department of Computer Science and Engineering, Chittagong University of Engineering and Technology for his kind advice. I would also like to extend my gratitude to all of my teachers for their valuable guidance in every step of mine during the four years of learning stage. Finally I would like to thank my friends, senior, juniors and the staffs of the department for their valuable suggestion and assistance that has helped in successful completion of the project.

## **Abstract**

Computer networking enables sharing and exchanging of data in a communication medium containing several hosts. With the growth of networking era, internet has been indulged with networking era day by day. Users can send and receive all kinds of information via internet. Providing a secure internet for dedicated users sharing a communication link has become an important issue now-a-days. Among several kinds of spoofing activities spread all over the communication system, ARP poisoning is mentionable. In this thesis work, providing security against ARP spoofing activities has been described. Moreover many more previous prevention and detection measures have been proposed for detecting ARP spoofing attacks. One of them is port security. Port security's drawback is it only allows communication among hosts those are saved on the table. if ip and mac pairings are not found on switch port then packets are ignored. Saving all the host's mac and ip pairing on switch port is troublesome. Switch is not intelligent enough for detecting spoofing activity by itself. Another method is intrusion detection system. IDS contains a drawback that is a network administrator needs to monitor the traffic 24/7 for detecting spoofing activity. Detection is done by checking whether two hosts have same ip address. For detecting and preventing ARP attacks, in this thesis, dynamic port allocation has been used. Attackers generally hold a static port value so that the states can never be updated for verification. Dynamic port allocation makes it possible to update state value every time for verifying a legitimate host. A controller(pox) has also been used for detecting spoofing activities and switch is made intelligent by the controller. Controller tells to disconnect any port when an attacker is found. Proposed method performs better than existing algorithm. Packet loss is less in ryu controller.

## Table of Contents

<b>Chapter 1:Introduction.....</b>	<b>13</b>
1.1 Address Resolution Protocol .....	13
1.2 Address Resolution Protocol (ARP) spoofing.....	15
1.3 Motivation of the work .....	16
1.4 contribution of the work .....	16
1.5 Organization of the thesis .....	17
1.6 Chapter Summary .....	17
<b>Chapter 2: Literature Review .....</b>	<b>18</b>
2.1 Background of the Problem .....	18
2.2 Present State of the Problem.....	22
2.3 Overview of Mininet .....	22
2.4 Openflow .....	24
2.5 Openflow Switch .....	24
2.6 SDN Controllers .....	25
2.7 Chapter Summary .....	26
<b>Chapter 3 :System Overview.....</b>	<b>27</b>
3.1 representation of the system model .....	27
3.2 Mininet Topology .....	29
3.2.1 Graphical Representation of NFG implementation .....	29
3.3 Algorithm .....	30
3.4 Proposed algorithm implementation .....	35
3.4.1 deficits .....	35

3.4.2 deficits abolition .....	36
3.5 Chapter Summary .....	39
<b>Chapter 4 :Implementation Details .....</b>	<b>40</b>
4.1 Existing Algorithm Implementation .....	41
4.1.1 ARP poisoning .....	41
4.1.2 DHCP .....	42
4.1.3 packet parse .....	43
4.1.4 network address translator .....	44
4.1.5 Lighttpd Web Server .....	45
4.1.6 Detection Code .....	45
4.2 Chapter Summary .....	47
<b>Chapter 5 :Experimental Result and Discussion</b>	
5.1 Experimental Setup .....	48
5.1.1 Topology for Existing Method .....	48
5.1.2 Proposed Method Topology .....	49
5.1.3 Modified Topology with a ryu controller .....	51
5.1.4 Modified Topology with a Lighttpd Web Server .....	52
5.1.5 Modified Topology with Additional Hosts .....	53
5.2 Experimental Results .....	54
5.2.1 existing Algorithm .....	54
5.2.2 Proposed Algorithm .....	64
5.3 Finding delay .....	66
5.3.1 Finding delay for Existing Algorithm .....	66
5.3.2 Finding Delay for Proposed Method .....	68



5.3.3 Finding Delay for Modified Method (Using RYU Controller) .....	68
5.3.4 Finding Delay for Modified Method (Adding Host to Topology) .....	69
5.3.5 Finding Delay for Lighttpd Server .....	70
5.4 Finding Packet Loss .....	72
5.4.1 Finding Packet Loss for Existing Method .....	73
5.4.2 Packet Loss for Proposed Method .....	74
5.4.3 Packet Loss for RYU Controller .....	75
5.4.4 Packet Loss for Additional Host .....	75
5.4.5 Packet Loss for Lighttpd Web Server .....	76
5.5 Link Loss Rate .....	77
5.5.1 Link Loss Rate for Existing Algorithm.....	78
5.5.2 Link Loss Rate for Proposed Algorithm .....	79
5.5.3 Link Loss Rate for RYU Controller .....	79
5.5.4 Link Loss Rate for Additional Hosts .....	80
5.5.5 Link Loss Rate for Lighttpd Web Server .....	81
5.6 Prevention of ARP Spoofing .....	81
5.7 Comparison .....	83
5.7.1 Comparison of Packet Loss.....	83
5.7.2 Comparison of Delay .....	85
5.7.3 Comparison of Link Loss Rate .....	87
5.8 Detection Comparison .....	90
5.9 Chapter Summary .....	90
<b>Chapter 6: Conclusion and Future Recommendation .....</b>	<b>92</b>
6.1 conclusion .....	92

6.2 Future Improvements .....	92
-------------------------------	----

## List of Figures

### Chapter 1

Fig1.1: ARP request and Reply observation .....	14
Fig 1.2: ARP spoofing .....	15

### Chapter 2

Fig 2.1 port Security .....	19
Fig 2.2:Intrusion Detection System .....	20
Fig2.3: A Simple Mininet Topology .....	23
Fig 2.4: Openflow Switch .....	25
Fig 2.5: SDN controller (Control plane logic) .....	26

### Chapter 4

Fig 4.1: ARP Poisoning Method .....	42
Fig 4.2: Dhcp server .....	44

### Chapter 5

Fig 5.1: Existing Topology .....	49
Fig5.2: Topology for Proposed Method .....	50
Fig5.3: Modified Topology for RYU Controller .....	51
Fig5.4: Modified Topology with Lighttpd Web Server .....	52
Fig5.5: Modified Topology with Additional Host .....	53
Fig5.6: packet Parse on mininet .....	54
Fig5.7: setting dhcp server on mininet .....	55
Fig5.8: setting dhcp server on mininet .....	55
Fig5.9: Setting Dhcp server on mininet .....	56

Fig5.10: Setting Dhcp Server on mininet .....	56
Fig5.11: ConFiguring POX Controller on mininet .....	57
Fig5.12: ConFiguring POX Controller on mininet .....	57
Fig5.13: ConFiguring Openflowswitch on mininet .....	58
Fig5.14: Running Detection Code on Controller .....	58
Fig5.15: Running Detection Code on Controller .....	59
Fig5.16: Initializing Wireshark tool .....	59
Fig5.17: Detecting ARP poisoning .....	63
Fig5.18: Detecting Duplicate use of Ip .....	63
Fig5.19: Detecting ARP spoofing .....	64
Fig5.20: Detecting ARP spoofing .....	65
Fig5.21: Detecting ARP spoofing .....	65
Fig5.22: Detecting ARP spoofing .....	65
Fig5.23: Detecting ARP spoofing .....	66
Fig5.24: Finding Delay for Existing Method .....	66
Fig5.25: Finding Delay for Proposed Method .....	67
Fig5.26: Finding Delay for Proposed Method(RYU Controller) .....	68
Fig5.27: Finding Delay for Proposed Method(Added Hosts) .....	69
Fig5.28: Setting Lighttpd Server .....	70
Fig5.29: Finding Delay for Proposed Method(Lighttpd Server) .....	70
Fig5.30 : Packet loss for existing method .....	72
Fig5.31 : Packet loss for proposed method .....	74
Fig 5.32: Packet Loss for Ryu Controller .....	74
Fig5.33 : Packet Loss for Additional Host .....	75

Fig 5.34: Packet Loss for Lighttpd web Srever .....	76
Fig5.35 : Finding Link Loss Rate .....	76
Fig5.36 : Link Loss for Existing Method .....	77
Fig5.37 : Link Loss for Proposed Method .....	78
Fig5.38 : Link Loss for RYU Controller .....	79
Fig 5.39: Link Loss for Additional Hosts .....	80
Fig5.40 : Link Loss for Lighttpd Web Server .....	80
Fig5.41: D. table showing ports .....	81
Fig5.42: D. table showing disconnected ports .....	82
Fig 5.43: Packet Loss Comparison .....	82
Fig5.44: Delay Comparison .....	83
Fig5.45: Link Loss Rate Comparison .....	86
Fig 5.46: Detection percentage comparison .....	90

## List of Flow Charts

Fig 3.1:A Flow Chart Representing Existing Method.....	30
Fig 3.2: A Flow Chart Representing Existing Method.....	32
Fig 3.3:A Flow Chart Representing Existing Method.....	33
Fig 3.4:A Flow Chart Representing Proposed Method.....	37
Fig 3.5:A Flow Chart Representing Proposed Method.....	38

## **1.Introduction:**

Address resolution protocol (ARP) request is a kind of request when a host wants to learn the MAC address of a destination host. That means IP address is known to the host but MAC address is not known. So a request is provided with the source IP and MAC address of the host and the destination IP address is known but MAC address is not known. Thus an ARP request appears. But the problem is spoofing activities may occur at any time. As the source IP is known, but destination IP is not known, so anyone can claim the destination IP as its own IP. ARP attacks occur when spoofing activities are performed. That means when the attacker has changed its own IP with the desired destination IP, then the attack occurs. ARP spoofing activities can be minimized by taking some ARP preventing steps. Several ARP preventing steps are existing. They are known as dynamic ARP inspection, intrusion detection system, port security. But the existing solutions are not efficient. In some cases they lack a lot. Software based preventing methods for ARP attacks are more efficient than the above processes. So software based prevention measures are to be taken. In this case software defined networking plays an important role. They provide an accurate and efficient method in a faster way. So day by day software defined networking is getting popular. Step by step verification of preventing method for ARP spoofing is a more precious way for ensuring security. Hence the measures followed for removing ARP spoofing activities need to be software based. In this paper we are going to propose algorithms for preventing ARP spoofing activities. It will work in two ways. Here we will use dynamic port and state allocation. Thus by checking and updating the port and state values each time, these algorithms work. These algorithms will provide a much secure way for preventing ARP spoofing activities. Reduction of ARP spoofing activities are now a big concern.

### **1.1 Address Resolution Protocol**

Link layer address can be obtained using a kind of communication protocol typically known as address resolution protocol[1]. The process includes both internet address as well as a mac address. Internet address is the ip4 address. ARP has two different special features. One of them is request and the other one is reply. Once a host sends a request to the other host and if the destined host wants to reply to that request, then a reply is sent to the host. Then a response is sent to the host which is considered as reply. When two hosts lie in different network, then they use a gateway for reaching one another. But after sending an ARP request, the other need to decide whether he wants to reply or not. If the host agrees to reply, then a mapping is saved

in the ARP cache. Then after matching with that entry, future communications are performed. Generally a host who will receive an ARP packet, will update its table with senders ip and mac mappings. Thus ARP works. The updation process continues for a time being. If the entries were saved a long time ago, no recent updates were not available, then the entries are dropped from the table. A new entry is added to the table, a new entry is given more priority than the previous ones. Thus new entries are added to the table. Former entries are deleted from the table if they seem too old. In a cache there exists two kinds of entries. One is static cache entry and other one is dynamic cache entry[2]. Static entry has no option for editing the existing cache. While a dynamic cache entry contains the option for flushing ip , mac mappings and creating a new ip and Mac pairings. The entries are kept on the table until a new request arrives for the receiving host.

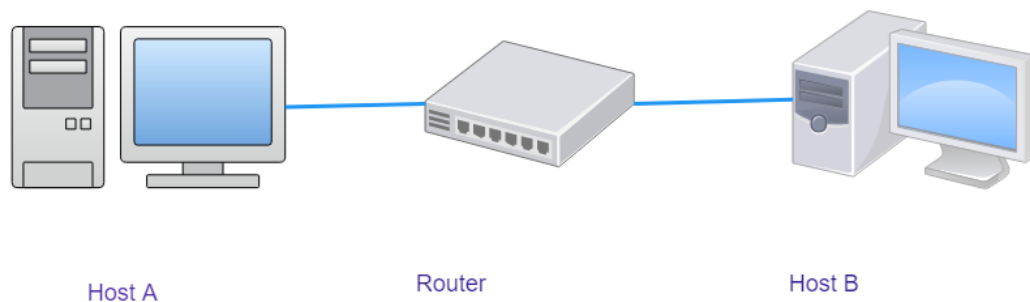


Fig1.1: ARP Request and Reply Observation

Here host A wants to send an ARP packet to host B.

A router is in the middle of two devices. Here the request will reach to the router at first, then the router will ask for host B's mac address.

The request is broadcast for seeking host B's ARP request.

As soon as the router learns host B's mac address, then an ARP reply is sent to host A through the router.

## 1.2 Address Resolution Protocol (ARP) Spoofing:

ARP spoofing is a kind of attack where an attacker replies instead of a legitimate host. In local area network an ARP request is sent for finding the host whose mac is not known but the ip address of that host is known. Attacker changes the ip address with the requested ip and then the he replies for the request which was actually not meant for him. But the mac address cannot be changed. Mac address of spoofer remains same as it was before. Then the target thinks that reply has come from a legitimate host and a connection is established though the ARP reply was falsified [2]. For example if the attacker spoofs the ip of a web server then the target sends his user name and password to the attacker though he does not know what is actually happening. Thus the attacker gets the log in credentials of target and attacker is also able to intercept packets of target. Even the attacker can change the important messages of a target. This is known as mitm (man in the middle attack). This is generally seen in local area network(LAN).

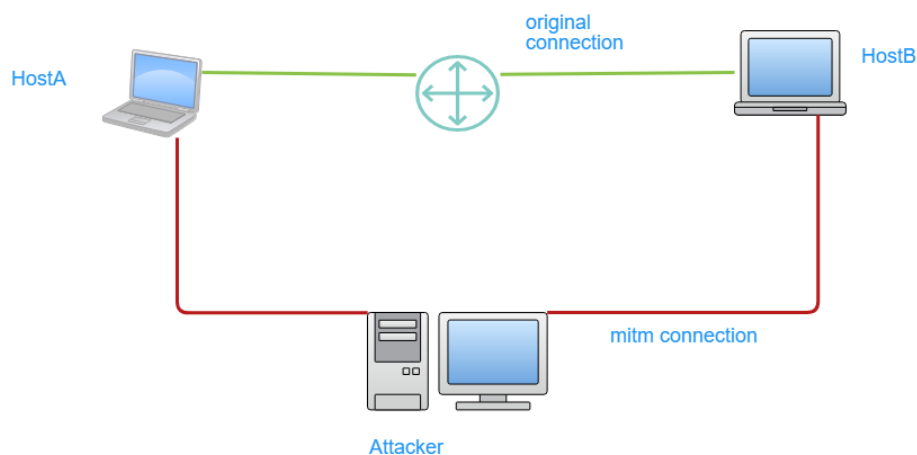


Fig 1.2: ARP spoofing

Host A sends an ARP request for host B. But attacker replies for ARP request by changing its ip address according to the receiver host of that ARP request. Then sender gets the reply and thinks that it is the proper reply and then adds the ip mac mapping to the table, then the attacker starts intercepting the packets sent by the another host. Thus a successful attack takes place and the attacker may even learn the personal information sent by host A. Even attacker may change the messages sent for the host B. Thus spoofing occurs. Attacker tells the target that the gateway

ip is his own ip , then all the packet of the target passes through the gateway. The target thinks that he is sending the messages to the receiver through the gateway. But in real, nothing is happening like that. The attacker gets the packets sent by host A. This is how man in the middle attack works[3]. Thus victims ARP cache is poisoned. To prevent this attack one needs to find the falsified ip mac mappings and then he will be able to prevent that mac from sending more packets to the target. For preventing and detecting ARP attacks different measures have been followed.

### **1.3 Motivation of the Work**

The motivation for the thesis was to ensure the delivery of secure ARP packet between the sender and receiver. Below is a list of points that specify the motivation for the thesis work:

1. Assignment of a static port allocation does not provide the state update facility. If the port is static then the packets need to be ignored for ensuring a secure communication. In this thesis work, port is kept as dynamic so that states can be updated in each steps that lead to a verification step.
2. Again when networking topologies have been introduced, it becomes really difficult to configure a network with physical devices. To configure a networking topology with physical devices is expensive and at the same dealing with physical devices needs extra caution. We are using mininet , which creates topologies virtually, so this is much and more effective than dealing with physical devices. Attacker may try a several times to attack the system, after being dropped from the flow table, attacker tries to guess a correct state value by providing a state value again and again, so we are solving thus issue. The followed method has been described below.
3. Ensuring security to network has been an important issue need now-a-days. In this thesis we are dealing with a secure ARP protocol.

### **1.4 Contribution of the Work**

The main objective is to present a detection procedure for ARP spoofing activities which is simple, easy for the client to use and also represents a faster detection of malicious ARP replies. To do so we need to optimize some metrics for presenting a better solution against spoofing activities. The following measures have been followed for detecting ARP spoofing activities.



1. We are dealing with assigning state values every time of verification. Now if the port is static, assignment of state values are not possible, so we need to ensure that port is not defined as static. Again assignment of state values are taken as random, as constant use of values may be guessed. This will lead to a disaster, so we will assign random values to the port.

2. Now a spoofer may try several times with a guess state value, as he wants to attack the system. Meanwhile if the attacker tries again and again, then after trying for a long time attacker may gain access to the network. Hence we are providing a limited number of trial request for any host. If the request from a host exceeds that fixed time then the ip will not be allowed to request again.

## **1.5 Organization of the Thesis**

The remainder of the report is organized as follows. In the next chapter, an overview of our project related terminologies have been presented. Also it contains a brief discussion on previous works that is already implemented with their limitations. Chapter 3 describes the working procedure of our proposed system. In chapter 4, we have illustrated our implementation of the project in details. Chapter 5 focuses on experimental results of the proposed system. The system concludes with a summary of research contributions of our work in chapter 6.

## **1.6 Chapter Summary**

In this section we discussed the core idea of address resolution protocol and how an ARP request generates from a host. We also learned how an ARP reply is sent to the host. Initially the request and reply was defined in brief. Then it has been discussed how attack in ARP occurs. This thesis contains the brief attacking process of a ARP packet. Then in the motivation section, we discussed the necessity of ARP attack detection and prevention. In this part we learnt how detection process will be performed and the whole implementation of these process has been discussed in another chapter. After that we discussed about contribution of the work. We learnt from this part that the attacks are normally seen ip's having static port allocation. So we have decided to ignore packet that are coming from the static port allocation. We made it dynamic so that one's state can be updated step by step. After updating states we will then reach at the verification step, at verified state, another value will be assigned.

## **Chapter 2**

### **Literature Review**

Ensuring a secure end to end communication requires every steps to be verified so that there exists no vulnerabilities that an attacker may use for destroying a network's privacy. Here we proposed a method for detecting these kind of activities so that an attacker is not left with a single drawback that may help him to attack the system. Thus several cases that may lead to an attack have been studied and then different solutions have been proposed. According to the proposal solutions have been implemented. As the system works by passing different steps one by one and when all of the states have been passed through, a verification step has been reached. After reaching the verification state, a finalized entry has been added to the cache. Then the packet is considered as coming from a legitimate host. Then a secure communication has been established between the two hosts and then packets have been passed through the link from one host to another.

#### **2.1 Background of the Problem**

Network security is a growing concern. And hackers continuously test the limits of tools available to network operators. Several mechanisms are followed. One of them is dynamic ARP inspection. Here is a brief description of dynamic ARP inspection. Intrusion detection system, port security, dynamic ARP-inspection etc. are various types of anti-spoofing skills introduced by developers. These are not an accurate or perfect solution. So these solutions need more and more guidance for restricting the attacker from committing spoofing terminologies. Port security has the problem that if the host is not connected with switch port then the host will not be able to establish a connection with those hosts. Intrusion detection system mostly depend on the network administrator. If the network administrator is not available for preventing a source from unauthorised access. So manual effort is necessary here. Actually manual effort is not always an efficient solution. Previous problems work for only one solution. But previous processes are not applicable for all the solutions. So these things need to be combined in such a way that the solution works for all the situations and all types of spoofing activities can be prevented. Here are the brief description of the previous works and the drawbacks that the previous works contain have been discussed here. Here are a description of the previous works and the reason why the current solution is better have been discussed here. There are also a description of benefits of the current algorithm. The reason behind choosing

these algorithm and why the algorithm performs better than the previous algorithms have been discussed her

**Port Security:** In a switch port, all connection's MAC address is remembered. So which device is allowed to connect and which device is not allowed to connect is previously known. Hence the switch port remembers the MAC address of the device. Now if a client tries to log into the network, switch port will disable that port for a while[3]. Thus port security is ensured. But it has number of drawbacks. Providing MAC addresses to specific devices manually is a big challenge. Few devices can be provided with this facility. So any other device that is not assigned to the port, may become an important host to communicate. But as switch port does not have any identity of that host, communication is not possible. This may turn to a major drawback of the whole system. However shut down, restrict and port are several solution commonly used in port security though these are not proper prevention requirements and these drawbacks were improved by introducing other prevention methods.

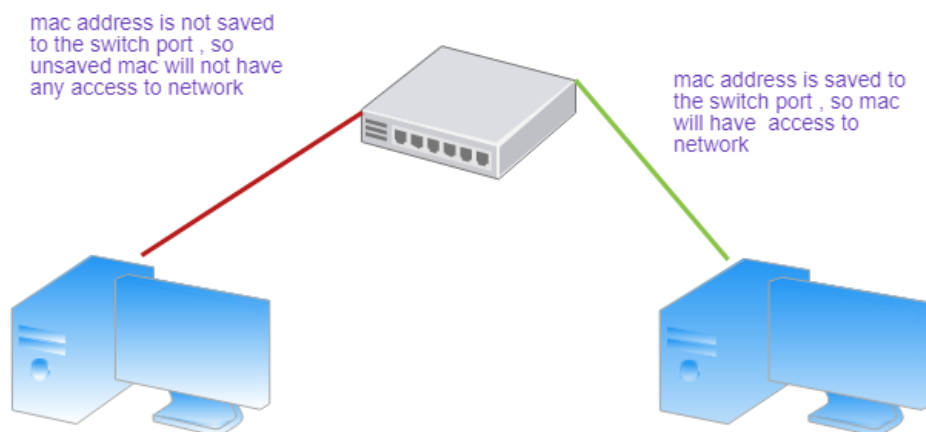


Fig 2.1 port Security

**Drawback :** Only those hosts are able to communicate through a switch port whose IP:MAC mappings are saved into the table. That is the number of host able to communicate with each & other are limited in number. Large number of host management requires a huge database. Doing Again it is quite tough to allocate all of the host's configuration into the switch port.

**Intrusion Detection System:** Let us consider an example. Suppose a user is restricted from using facebook. So whenever he/she tries to log into facebook, firewall get notification about the unauthorised access of the user. So the user is restricted from using that site that he/she has

been prohibited from using. Firewall restricts unauthorised access gain of the user. Now the firewall won't approve the request of the user at all. That means the request is denied from forwarding. Now if the user may be cunning enough and he/she may use proxy server for gaining unauthorised access. But if an intrusion detection system is installed into the network then the user can be prohibited from using that site because for browsing that site the user needs to use the gateway of the network where he/she has been at the current state. A network administrator is needed because intrusion detection system creates an alarm for the network administrator for concerning him/her that an unauthorised site is requested from the user[1].

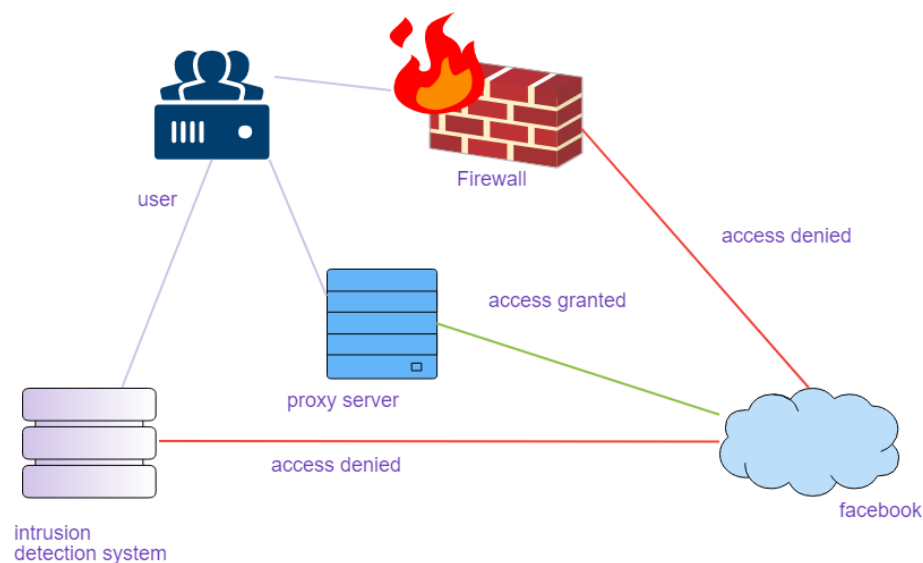


Fig 2.2: Intrusion Detection System

### Drawback:

The task of network administrator is to look over the network 24/7. But this task can not be performed manually by the network administrator.

Now the charming innovation is using software defined networking for preventing and detecting ARP poisoning. Software defined network is a virtualized networking where data plane and control plane have been separated from the similar plane. Data plane introduces the network traffic. Control plane introduces the routing tables for each routers. Number of hardware is reduced as SDN has been introduced. Furthermore SDN introduces us with open flow switch. Open flow switch is a kind of switch that makes switch chip sets to be controlled using a software. That means controlling switches is possible using a software. That is a huge revolutionary contribution from SDN to networking. Thus ARP poisoning attacks are detected

and prevented using this methods. Virtualization of networking enables to configure necessary networking tools that were previously done manually. Now measurements that need to performed were done by network administrator for preventing and detecting ARP attack. But now-a-days SDN provides with virtualization of networking that can be configured easily. This procedure is saving much more time[4]. Instead, software defined networking, a new paradigm that decouples the control plane from the switch's data plane to provide security against ARP spoofing attacks. Network flow guard is a modular application that augments an SDN controller to detect and prevent ARP-replies from unauthorized hosts. This method requires no topology changes, authentication services, cryptographic keys, or significant network operator support, instead NFG monitors dynamic host configuration protocol (DHCP) Offers, requests, and acknowledgements from valid DHCP server to construct a dynamic table consisting of MAC: IP: port: fixed: state associations for each device on a given LAN. Doing so leverage the capabilities of SDN to prevent ARP poisoning attempts as they occur. NFG's key contribution is that it provides ARP security while requiring little network operator invention, no additional equipment or host software, and no changes to the network's current topology or protocols[5].

### **Problems of Existing Method:**

The puzzling problems about the existing method is that the update of state values are fixed there. Using a fixed value for state update provides a risk for the users that one may guess the correct state value and then he may attack the network. Thus using a constant update value may lead to a problem.

The another problem is that we are checking whether dhcp offer or acknowledgement is true or false. This checking is based on the request arrival. When a request arrives we need to check the value for ensuring an offer or an acknowledgement. After that if we find that the request is from a suspicious ip, then we are going to ignore the packet. Here if the attacker tries several times, then he will be succeeded to guess a correct state value. So what we can do is to set a limited number of time and this time indicates how many time an ip can request for offer or acknowledgement. Finally , if request from an ip arrives exceeding that limited number of times, then we will ignore the packet. Otherwise attacker will send more and more requests by guessing a correct state value and the network will be spoofed.

## **2.2 Present State of the Problem**

We discussed about the drawbacks of the existing algorithm in the background of the problem. In this section we are going to add solutions to the previously mentioned problems. For solving the problems mentioned above, we need to set a counter like variable that can count how many time a request has arrived. Next if a request arrives more than the counter's value ; then we are going to ignore the request. Thus one deficits of the existing algorithm can be solved.

In this part, we are going to discuss about the second problem which was described in the previous section. The problem is, if the state value is almost same for all the request then one time the attacker will somehow manage to guess the state value. So we should consider such methods that will prevent the attacker from guessing a correct value for state. In this case state can be updated as we are using dynamic port allocation. Moreover, to solve the above issue, we are going to assign the state with random values. When a request arrives we will be updating its state value with random values. As random values are difficult to guess, so security is assigned to the ARP protocol. The main issue is that , we are using dynamic port[18] allocation. It supports updating state values for ensuring security. The mentioned two problems are threats for the existing method. We are solving these two problems by following the methods mentioned in this section.

## **2.3 Overview of Mininet**

Different types of software emulators are available now-a-days. Mininet is one of them. Mininet is used for presenting a large network on a single machine. Mininet uses openflow switch. Openflow switch details is presented on the article below.

Network topologies can also be simulated using this emulator. It allows the user to quickly create a topology within a single computer.

Software Defined Networking needs an environment and a proper environment is provided by mininet. SDN development is allowed on any laptop using mininet[6]. Sometimes complex topologies need to be tested , for testing a physical network is not a good option. In this case we may use mininet.

Host, switches and controller is the minimum necessary things for creating a topology on mininet. We are able to define the topology as our requirement. We may occupy as many host as we want. Again we may occupy as many switches as we want. Thus different combinations provide a different result.

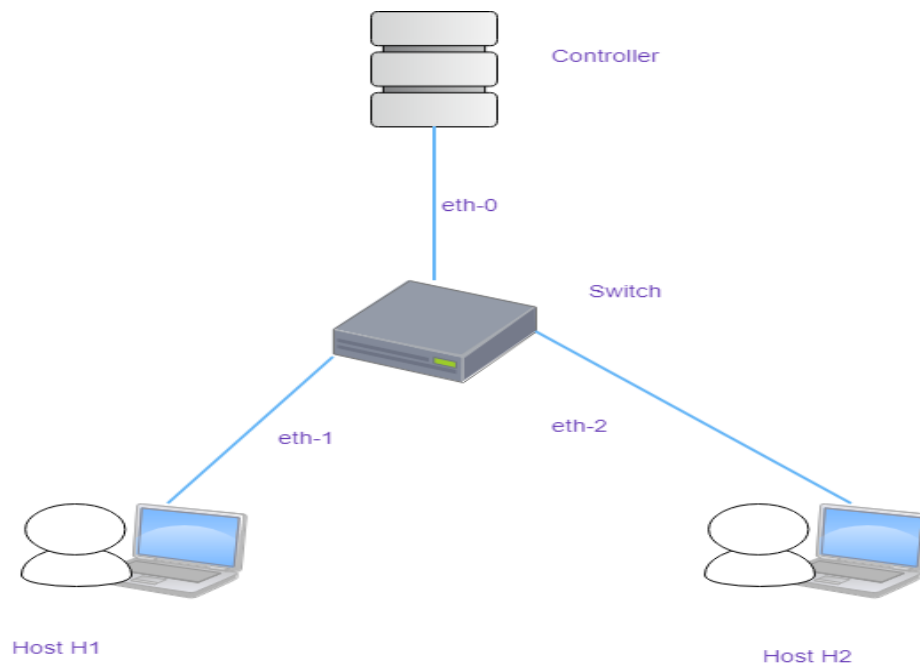


Fig 2.3: A Simple Mininet Topology

The main advantage is it provides an opportunity for creating a testbed that has no expense. Moreover, networking topologies need to be experimented and an extensible topology is desired to be developed. So mininet is introduced for serving all of the mentioned purposes.

Performance evaluation, testing, deployment etc. needs minimal changes to the network topology. Dealing with physical devices may not be very effective to serve this purpose. Hence mininet is used for finding and comparing experimental results.

Mininet has a list of commands which are used for developing a simulation. Here Python API has been used which may be extensible and adaptable as per requirement.

Connectivity among the hosts are also shown using mininet[7]. If the ping is successful then the rate is shown as 100% otherwise it is 0%.

Linear topology, Single topology any kind of topology can be adopted in mininet. Tree topology can also be implemented.

Creating a virtual testbed has minimized expense and a lot more things. Applications of SDN can be built using mininet easily. Basically to develop and experiment with openflow requires networking devices which extremely expensive , to avoid such circumstances we may choose mininet as working environment[8].

## **2.4 Openflow**

Today's networking industry is being fascinated by software defined networking. Agility has been added to the network by software defined networking. Openflow protocol is the core of the SDN technology . Openflow adds flexibility and scalability to the network.

SDN requires a programmable network protocol as communication between controller and switches needs to be maintained by that protocol. Openflow is that desired protocol which is used for the above purposes. Programming of networking devices is separated from underlying hardware using openflow protocol. openflow makes that network adaptable so that changes in network can be quickly accepted.

Openflow offers a centralized and programmable network.

## **2.5 Openflow Switch**

Openflow protocol is enabled in openflow switch. An openflow channel is used for communicating with the controller. It contains one or more flow table or group table for the purpose of packet forwarding. The openflow switch is managed by the controller. flows are needed to be managed securely so that any threats can be avoided, so controller basically monitor the floes on switches. Conversation between switches and controllers are performed using an openflow protocol[9].

In this thesis , we are dealing with dynamic port allocation, this dynamic demands are fulfilled using an openflow switch.

Packet forwarding and routing are done on the same device in a conventional switch. The data plane and control plane are decoupled from each other on SDN. Now switch handles the data plane but SDN implements the control plane. Routing decisions are taken by SDN controllers. A communication is needed to be established among controller and switch, this is done using an openflow protocol[10].



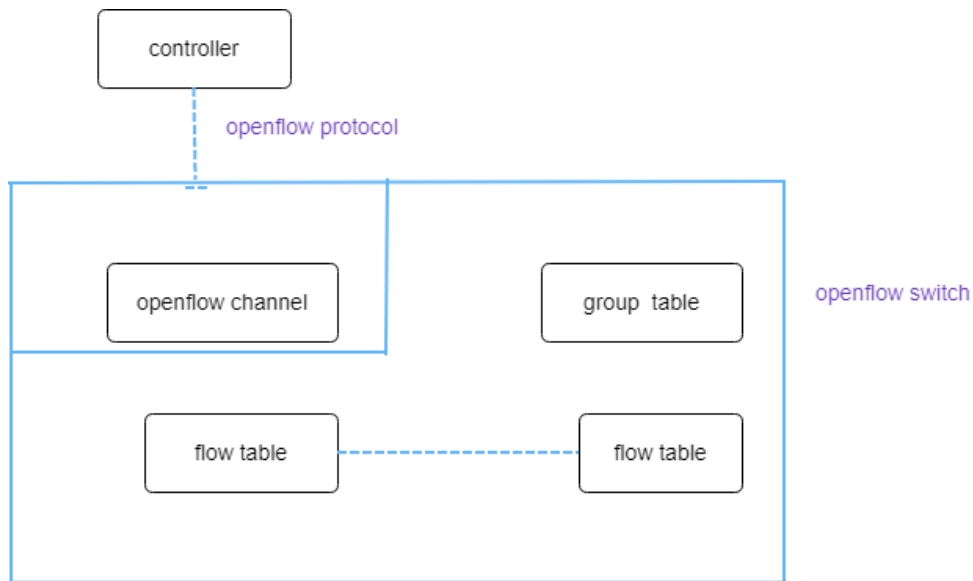


Fig 2.4: Openflow Switch

## 2.6 SDN Controllers

There are a lot of variation among SDN controllers. Flow control is managed by SDN controllers. in this thesis we implemented pox controller and we also implemented ryu controller. Pox is one of the mininet network simulators. Updating flow tables is the core task of this controllers.

Switches and hosts are basically controlled by the SDN controllers. A framework is provided by pox for communicating with openflow switches. New pox components can be created and then a more complex SDN controller will be found. Users are allowed to write their own applications. For running a pox controller, pox components need to be specified according to the necessities.

When a switch receives packets from hosts that are not present on the flow table ; every time a traffic has been generated to the controller. Thus controller comes to know about every flows. Even controller can handle the updating process of flow tables.

Pox is a python based controller. Other controllers are also available, they are named as Nox, Ryu, Open Day Light etc.

Programmability and separation are both are the most attractive and effective part of SDN controller. Here pox controller can be designed as per requirements. Like this, it is written in python and by programming according to needs, one is able to change the previously

programmed controller and may design a new one. It keeps two different interfaces called northbound interface and southbound interface.

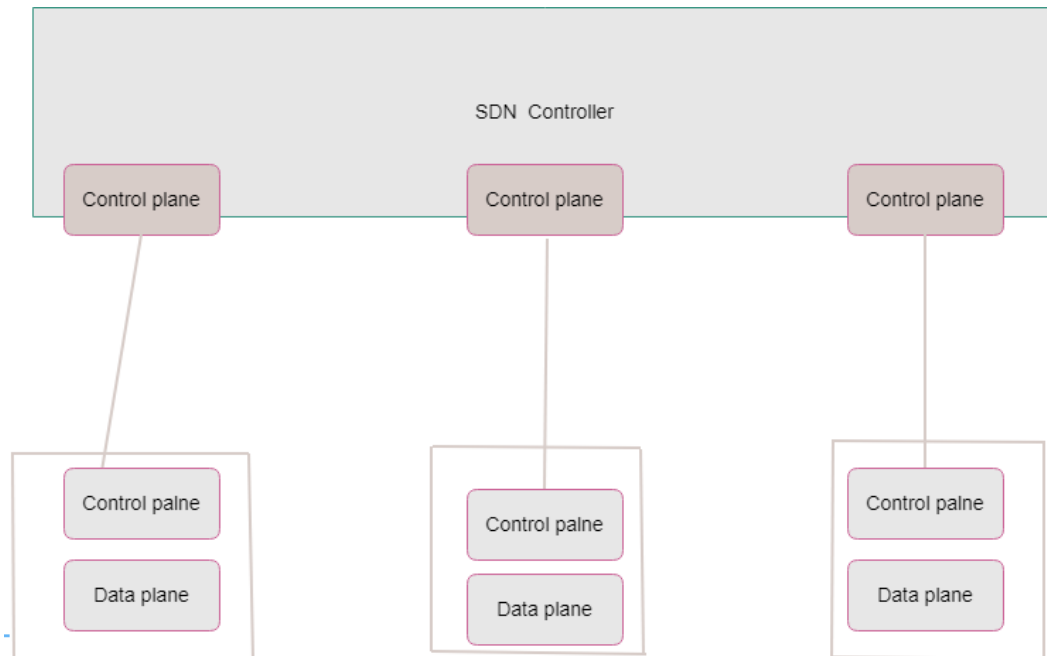


Fig 2.5: SDN controller (Control plane logic)

## 2.7 Chapter Summary

From the very beginning , networking security has become an important issue for keeping the system's integrity a secret. For doing so there have been a lot of measures. The measures have been followed so that the poisonous activities can be prevented. Here we discussed about such two measures. They are intrusion detection system and port security[11]. These two methods have been discussed with their drawbacks. The current world needs more authentic and faster recognition of the detection measure. Continuous monitoring may not be possible for human beings. Software defined networking may solve this issue.

## Chapter 3

### Method Overview

In this thesis , a solution against address resolution protocol attack has been presented. Currently different solutions are available. But all the solutions are not effective considering different scales. So to find an effective and faster solution , we have given priority to check the port. Again different states have been assigned with random values and finally a verification state has been found. Moreover, the assigned values are random as there is a mere possibility that attacker will guess the state value then he will be reached to the verified state. This is a straight threat to the networking system.

### 3.1 Representation of the System Model

Our implementation procedure followed a mininet utilization. A realstick virtual network is needed for implementation and mininet basically creates that opportunity. Openflow can be experimented or developed easily using mininet as a virtual platform. We can work with devices that are needed for our researches. We can work with switches, controllers and hosts and all other networking devices along with links.

The characteristics considered in mininet network topology is described below:

- Large network needs a huge resources for running a testbed. But using mininet we can implement a large network with limited resources.
- As hardware testbeds are very expensive though they are faster and shared. But to avoid expenses initially, mininet is the best testbed alternative to hardware.
- Accuracy of performance can also be measured using a mininet testbed. Thus an accurate result does not need any kind of expenses.
- Mininet contains Minimal, single, tree, reversed or linear topologies which are default topologies.

- For research works, it may be needed to work with some custom topologies. Mininet holds the feature of creating custom topologies. we can add as many hosts, switches or other devices as much as we wish.
- Lots of numbers , classes, functions are found in mininet.
- There are a list of basic commands which are used for configuring mininet.
- Mininet topologies need a base class.
- Ones network can be started or stopped using start() and stop() command.
- Connectivity among hosts can also be checked . any connection or link can be dumped if the user wants to.
- Topology needs an application of appropriate parameter.
- An expiration of flow entries are available. That means after a certain time period, flow entries will be added again. Previous entries are denied. A new addition of entries are needed.
- Reconfiguration of a real system is not only difficult but also painful. So one can use a virtual machine installed with mininet for configuring a system again and again. It's free of cost and other technical issues have been omitted here.
- Very large scale network which is hard to implement on real system , can be implemented using mininet.

## 3.2 Mininet Topology

In this thesis for suggesting an adaptable solution for current network includes a NFGC (NFG coupler) module. The NFG coupler couples both a mac learning switch and NFGA (NFG ARP).

Rules are generated by the NFGA and NFG coupler allows to generate these rules. These rules are pushed through the northbound interface of POX controller along with the correspondence of switch. POX controller generates and interprets these instructions and then pass them via southbound interface of the openflow switch.

NFGC have been modified using pyretic, which is a modular programming language. Through this part, we may tell the controller to handle almost all the packets. After identifying destination ip, source ip, destination mac, source mac the controller tells the switch port to disconnect a port, or maintain connectivity with a host. Controller has some basic rules that contain all the necessary conditions for checking whether an attack exists or not. Controller handles the switches flow table. The flow table entries are updated using the commands that will be passed from the controller via its southbound interface[12].

### 3.2.1 Graphical Representation of NFG Implementation

Here is a representation of NFG module. Pyretic queries are used by NFG. Different types of controllers exist. Among them pox controller, ryu controller and frenetic controller , ODL open day light controllers are commonly known. There also exists another controller named NOX controller. But NOX controller is specified using c++ language. Python based controllers are ryu and pox. These controllers are used for different purposes. For implementing a simple topology, frenetic controllers are the best. Every controllers have different performances based on the throughput and latency[27]. Controller is the core part of the network. When a poisoning activity is defined, controller tells the switch port to disconnect the port. A dynamic table is used for showing which ports have been disconnected. The ports are shown separately from other table entries. After detecting an active attack, a prevention is also needed. Otherwise only detecting a poisonous activity does not ensure a network's security. So , the poisonous entry has been removed from the table and then all the entries are updated as before. Also this thesis contains dynamic port allocation. Dynamic port allocation means that the ports can be updated using different states[28]. Table has different entries. And this entries can be modified at any time. Entries that are old can be replaced with a new one. That means entries are updatable at

any times. If any saved entry is not updated for a long time then this entry will be removed from the table and a new entry will be added to the table. Thus entries are updated at every arrival of a new entry request.

### 3.3 Algorithm

Algorithm has several steps.

Dhcp packet arrives from the server to the host. This is known as dhcp offer. After that specified host sends a reply as dhcp acknowledgement to the server. So we have two arrival request initially, this is known as dhcp offer and dhcp acknowledgement.

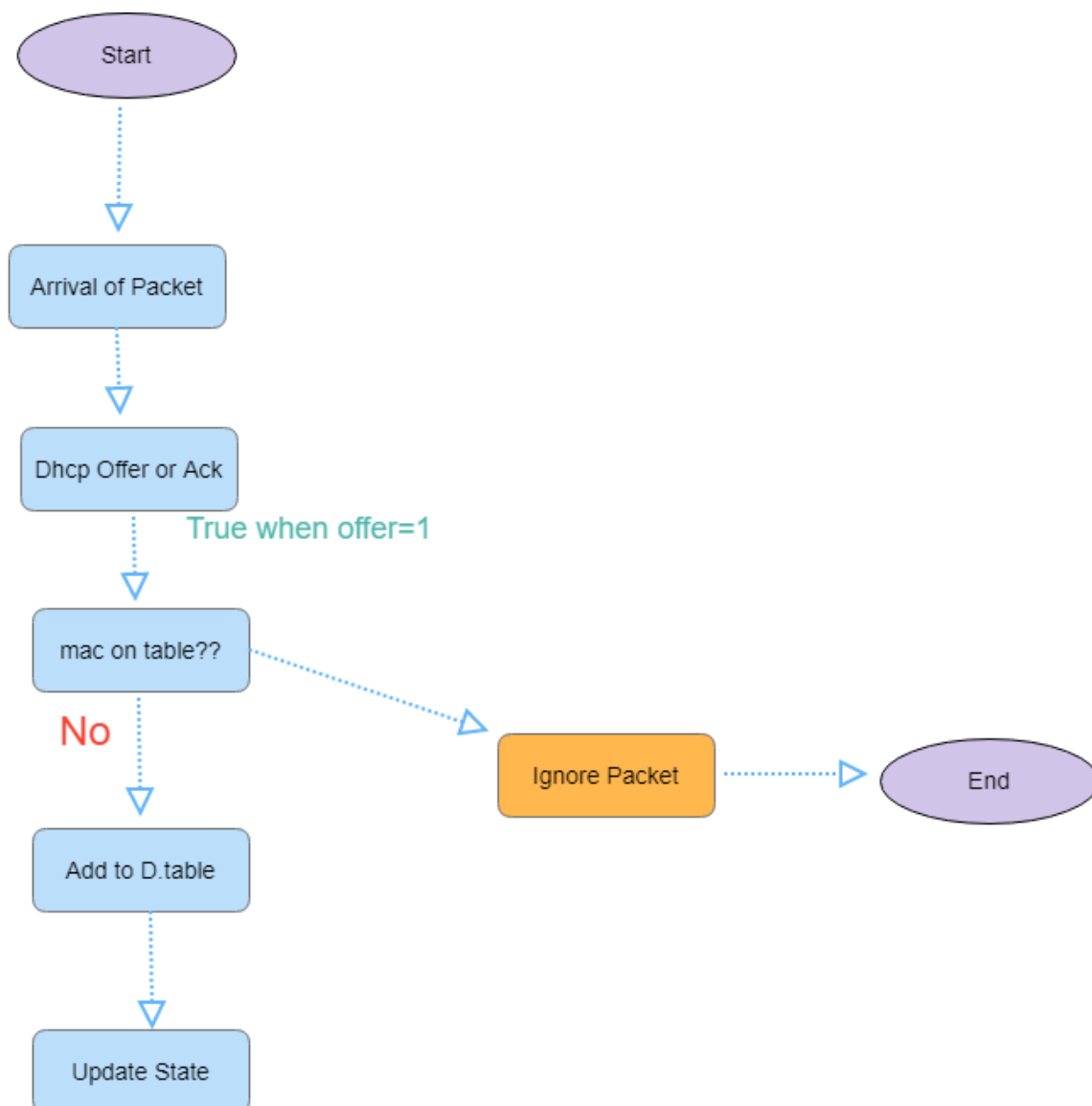


Fig 3.1:A Flow Chart Representing Existing Method

It means that a dhcp packet needs to be arrived. Then it is checked whether it is an offer or acknowledgement. The true value indicates that the entry has been already inserted on the table. So the request is not a new one. The request is an old one. Therefore it is neither an offer or an acknowledgement. So finally the value is found as false. As the request can not be recognized as offer or acknowledgement, the value indicates a false value. That means the request has arrived before. After that the mac is checked on the table. If the mac is not found on the table then packet is ignored. If the mac is found on the table then another processes are followed.

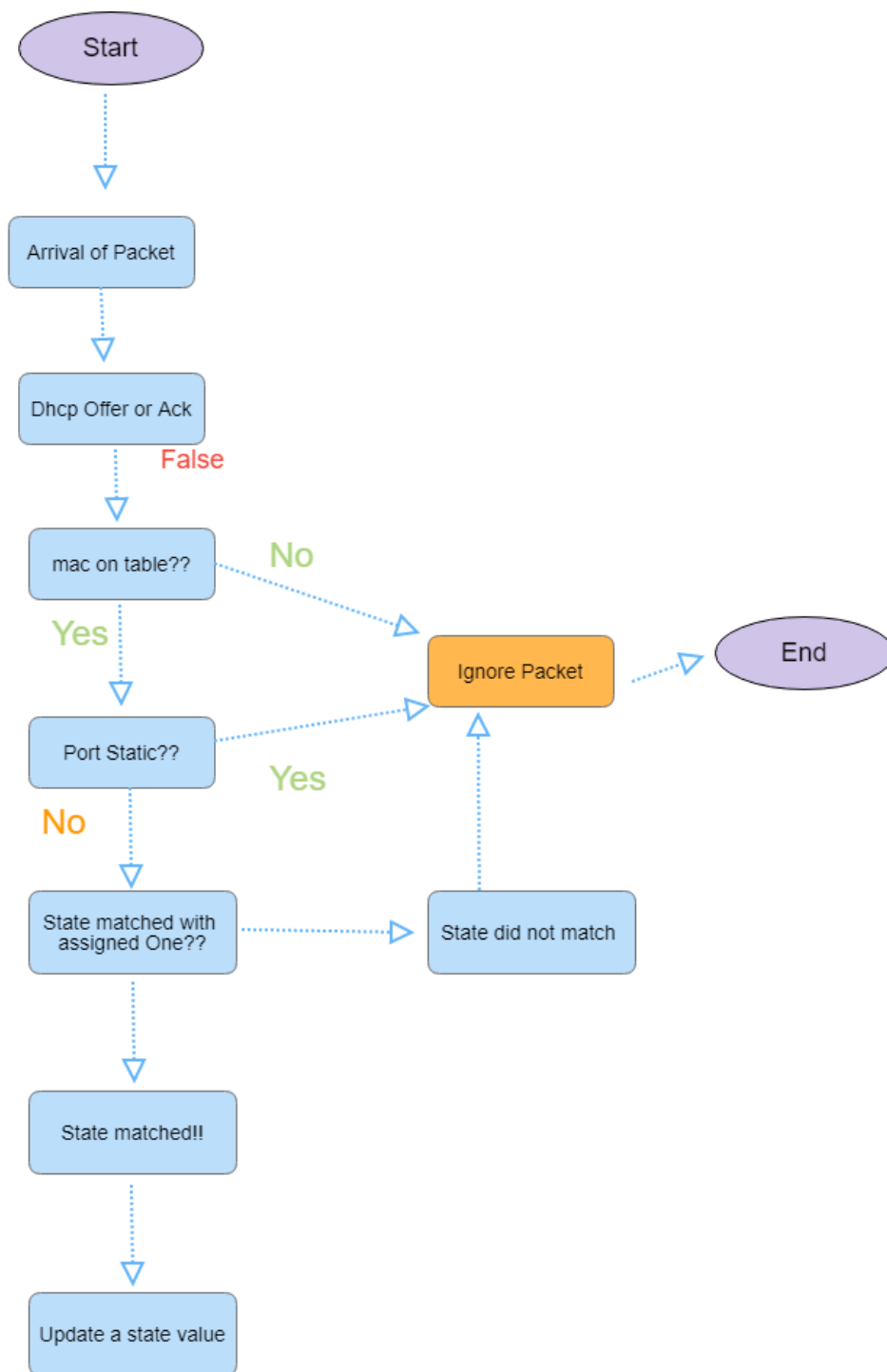


Fig 3.2:A Flow Chart Representing Existing Method

The thesis work involves an dynamic port allocation. So port which is static is ignored instantly[29].



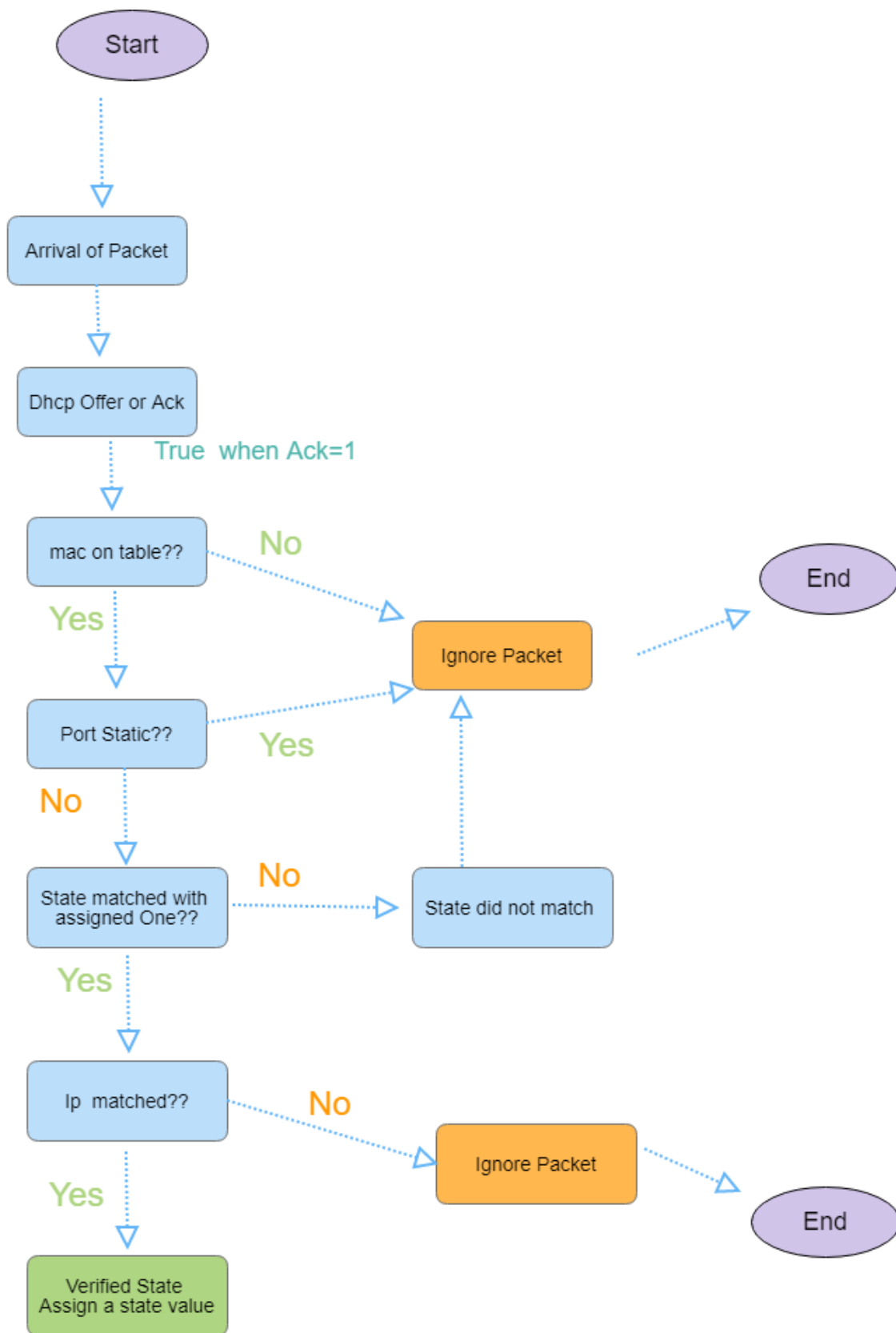


Fig 3.3:A Flow Chart Representing Existing Method

These are basically intermediate states which are used for verifying state values and finding out whether a verification state is found or not. The state values are used for matching with the assigned one and the actual value. Port uses fixed assigned as true or false for finding out whether a port is static or not.

In the above states, we discussed whether an old entry has a new packet arrival. This is done for ensuring maximum security. As the old entries may attack the network, if no verification steps are considered, so we are keeping an option for verifying a legitimate request although the new request has arrived from an existing entry. This basically minimizes possibilities of an existing entry being an attacker. Thus this portion of the algorithm optimizes the detection procedure. This is done so that an entry which is on the table do not send a new offer or acknowledgement. Thus an existing entry's request will also be verified step by step and then finally the packet will be ignored or accepted.

This is done when a new request arrives. Then an offer is sent by the dhcp server and then an acknowledgement is sent to the host. This actually indicates that there is no existing entry on the table. There was no previous request from the mac before. For handling a new request, a new entry was made to the table and then the entry is dropped if it fails in step by step verification.

Static port allocation is not supported, in every step before proceeding it is checked whether the port is static or not. If the port is static then the packet is ignored.

The above algorithm works in a number of steps. Initially the request is assigned with a state value which is zero. It may be a random value. After that the offer and acknowledgement value is found false, then an intermediate state value is assigned, before that it has been checked whether the state value is zero (assigned when offer, acknowledgement is true).

Thus the assignment works. After the intermediate assignment, the offer and acknowledgement is false each time. Then it is checked if the requested ip is the same as the currently processing ip; if matches then a final state assignment is done, it is known as verified state.

At the very beginning the value returns with a true value and after that the value returns with a false one. Finally, after verification state is reached, it can be easily predicted that whether an attacking activities were present. If present then the entry is removed from the table. If not

present, then the entry is added to the table. In this attachment, we have shown a table with the prove of prevention method that includes the disconnected ports.

Basically the controller detects the poisoning acts and then tells the switch to drop the packet and to disconnect the port in association with the poisoning ip address. Thus the ip is prevented from poisoning an ARP cache.

The procedure does not need any network operator to monitor the running activities all day and night. The network administrator needs to block any poisonous ip if it seems to be suspicious. But this process is so troublesome. It not possible for a person to monitor the network all the time and prevent the network from poisonous ips.

### **3.4 Proposed Algorithm Implementation**

The above algorithm is implemented with different modifications that maintain the solutions for above algorithm.

#### **3.4.1 Deficits**

Initially we implemented the proposed method that was presented on the paper. The procedure contains some deficits stated below.

One of them is , in every step we are assigning a state value. Initially , when the request arrives for the first time the state value is assigned as zero. After that when the offer and acknowledgement value is given as a false value, the state value is assigned again and this time value is one. The second time offer and acknowledgement value is treated as false, because the request has arrived for the second time. However, the state values can be changed by the attacker, attacker may guess the right state value and then the entry on table will be added. The attacker will be treated as legitimate host. As the previous method uses one and zero as state's value , so this can be easily guessed by the attacker. If the attacker can guess the right state value then it will be a poisonous activity. So, in summary state value should be used or declared or assigned in such a way that no one ever can guess the right state value. That's why user need to use a random assignment of state values. This ensures that no two consecutive or any other request does not have the same state value. Here the state values need to be initialized with randomized values. Random values are used as there are risks for guessing the assigned state values.

There exists another risk that if one tries to attack a network, he may try with several state values, thinking that he may succeed one time by guessing a right state value. So steps should be taken in such a way that one may send only one request at a time. After a certain time period, it will not be possible for any host to send a request. So request from any specific ip need to be saved and then the saved value will be compared with the minimal accepted request times. If the time exceeds, then we need to prohibit from that ip from sending any kind of ARP request. There should be a counter for counting the number of requests arrived and then the counted request will be checked.

### **3.4.2 Deficits Abolition**

In our proposed method, we will set a counter for performing the number of request arrived. Then the counted number will be compared to a minimum set value that indicates that if the number of time a request has arrived crosses the minimum set time value then we will send the switch port a message to ignore the packet. This whole task will be performed through a controller. Thus the controller handles the minimum number of time a request occurs. There is a chance that the attacker may change the state value again and again and have a lots of attempt for poisoning the cache.

While providing solutions to the above problem, we are assigning a minimal request time counter. Another problem stated is, if the state value is assigned with the values like one or zero or two etc, then it will be easy for the attacker to poison the system that's way we need to assign state with such values that can not be imagined easily.

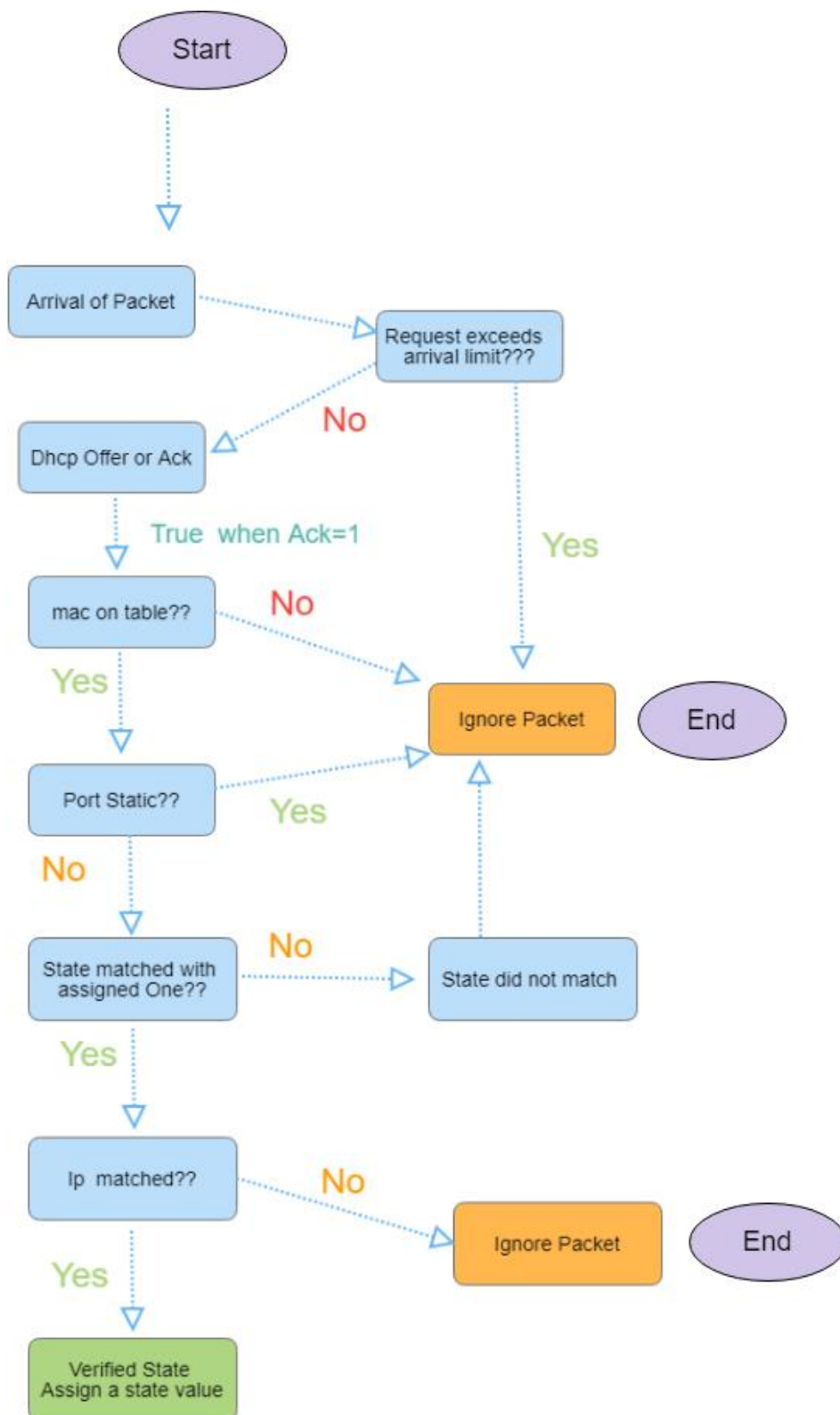


Fig 3.4:A Flow Chart Representing Proposed Method

For serving this purpose, we are going to assign random values. That random values will be matched with the currently saved state values.

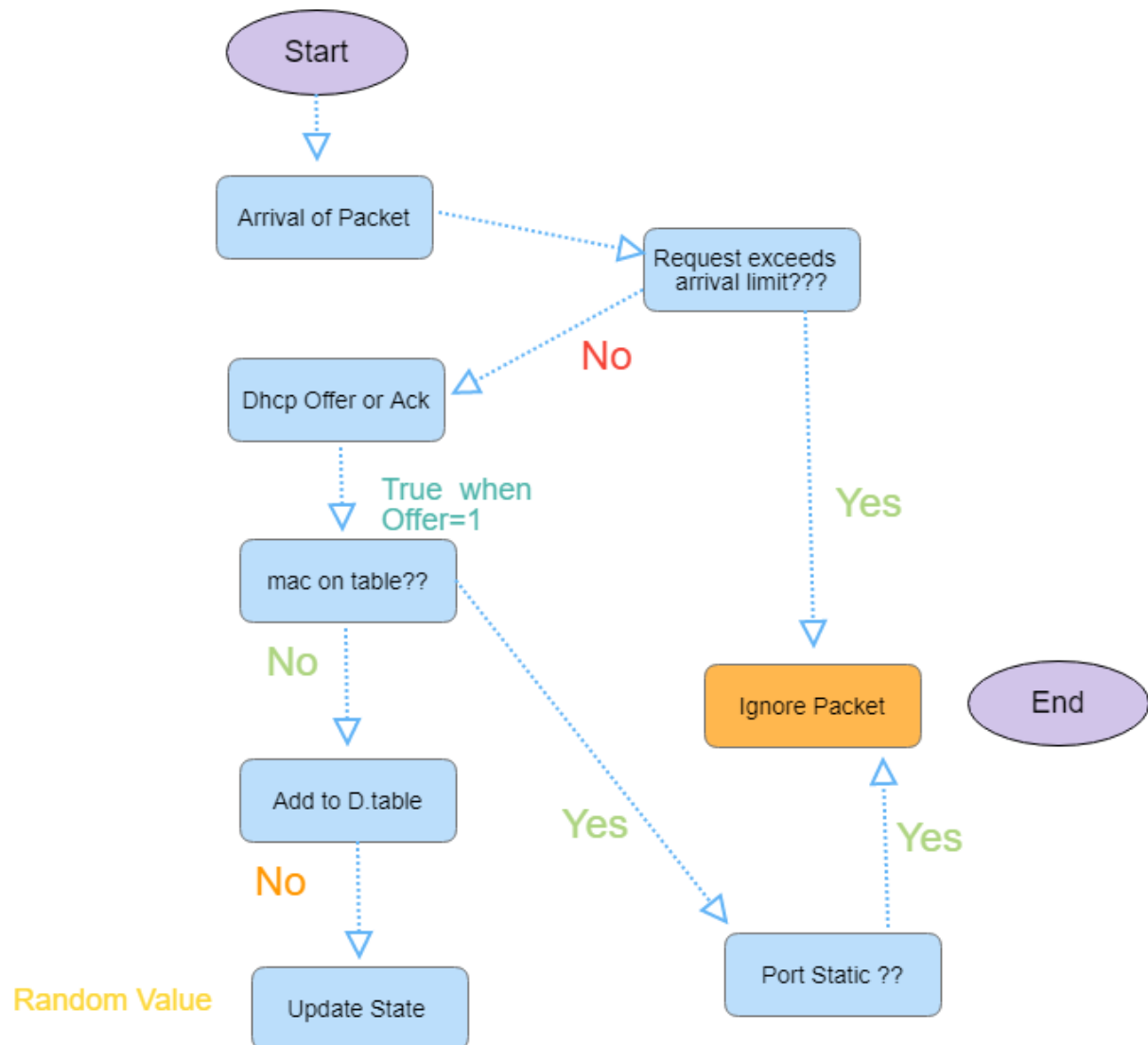


Fig 3.5:A Flow Chart Representing Proposed Method

If matches are found then states will be continued for verification. If state's value does not match with the assigned one, then it indicates a poisoning tasks. The port will be disconnected instantly. Thus the processes work with assigned random values.

### **3.5 Chapter Summary**

In this section we covered a representation of the system model. A graphical overview has also been presented here. Here the steps of the algorithm has been discussed briefly. Again the proposed method has also been discussed. We also provided a brief of every algorithm steps. All the steps were discussed with their necessary significance. We also discussed the deficits of the existing algorithm and then the abolition of the deficits using the proposed method have also been discussed.

## Chapter 4

### Implementation Details

In our implementation details we are presenting the different ways for finding the solution of the proposed algorithm and the previously mentioned algorithm.

To implement the system we are setting an environment comprising a list of devices. The devices are an openflow switch, controller (POX), dhcp server, hosts, network address translator and a lighttpd web server. Different links have been created for implementing the devices at once. Now this devices need to be configured according to requirements. Here are a representation of the devices configuration.

#### 1.Hardware Specification

- Personal Computer
- Core I5 processor
- 2.50 GZ clock speed

#### 2.Software Specification

- ❖ Operating system
  - Ubuntu 14.04 LTS
  - Kali Linux
- ❖ Programming language used:
  - Pyretic
  - Python

#### 3.Tools

- ❖ Wireshark



## ❖ XARP

### 4.Platform

- mininet

## 4.1 Existing Algorithm Implementation

### 4.1.1 ARP Poisoning

while writing a code for poisoning ARP we need to provide the ip address of the gateway and the target. The process works as described below. Initially the attacker tells the target that he is the gateway. Unfortunately the target does not recognize the actual gateway and he sends packets to the attacker thinking that the ip of the attacker is the gateway ip. Hence the attacker can intercept the packets sent by the target ip. Then he may change all the contents lying below the packet or he may know the secret messages sent by the target ip. Python uses below classification:

```
target_ip  
  
gateway_ip  
  
conf_iface
```

here conf\_iface takes the interface as input.

python has another class called get\_mac. ARP request is constructed here.

function : sr

A layer three packet is sent or received using sr function.

Restore\_network: As the attacker also sends packets using a gateway, we need to set the actual gateway for transmitting packets to the target. That is why network which was poisoned before need to be restored. This is done so that the actual gateway works for the attacker.

ARP\_poison: This uses gateway ip, gateway mac, target ip and target mac for ARP poisoning. While source is the target or the host from which a packet is sent to the destination and the gateway is used as destination. For attacking, a false gateway ip is used.

When the script has started, then ip forwarding has been started.

Poison\_thread: Poison threading is started using `poison_thread.start()`. Again one the used arguments are gateway mac, gateway ip, target mac and target ip.

Get\_mac(gateway\_ip): Shows the gateway ip, also compares gateway ip after the reverse networking operation.

Sniff traffic: This is done by using target ip as the ip host. This is basically known as `sniff_filter`. Network traffic has been started and then `packet_count` counts the packets.

Sys.exit(0): This stops the network and then the network capturing is also stopped until all the ips has been processed for detection measure.

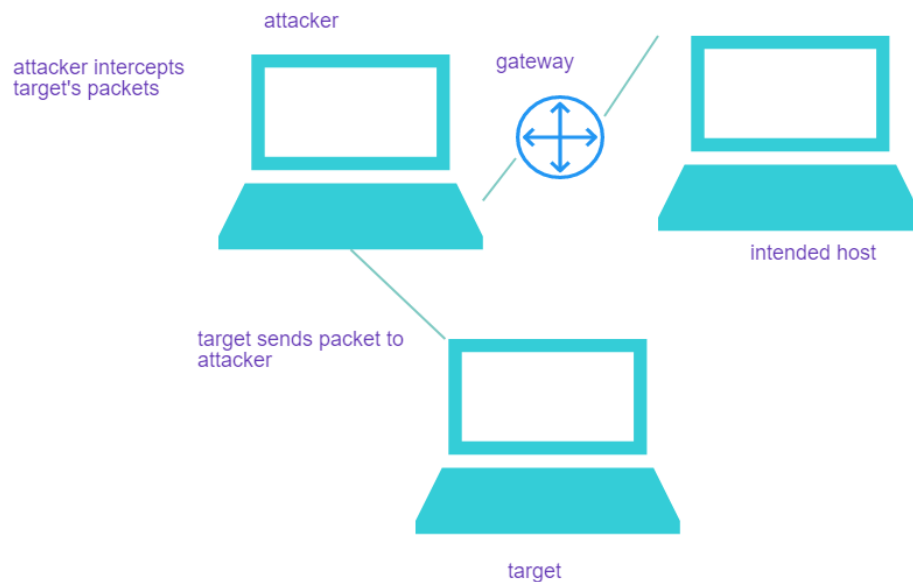


Fig 4.1: ARP Poisoning Method

#### 4.1.2 DHCP :

A dns template has been used for defining dns zone. When a domain is registered, then a dns zone is defined.

dhcp option 60 : It defines a class id.

for example , option 1 defines domain name and option 2 defines domain server. Here 60 indicates class id. Dhcp server is used for assigning ip. Ip addresses for gateway, target and attacker has been assigned with the help of dhcp server[13].

hostconFig

linkconFig

Host links are specified using necessary parameters. The above two functions present those parameters for specifying links. Host links are defined here.

dhcp conFiguration parameter

This function creates a conFiguration file. The parameter used is called conFig. Dnstemplete is also a parameter used for this purpose. Option router and option dns is also used. These two things mainly contain natip and dns. After the conFiguration, dhcp server is activated once and then deactivated once.

Host, switch and dhcp is connected through links. They create a new network. here victim requests for joining to the mininet topology. But here dhcp is connected with the switch through a slower link. There exists some delay in this example. For example the delay may be 500 ms. Evil is connected with the switch with a faster link. Here evil provides a false dhcp response. evil hosts a malicious dhcp server and a dns server. For example host h1 sends request to both dhcp server and to the evil. As evil is connected to the switch with a faster link, so dhcp response is provided from the evil to host h1. Evil here uses the dns ip as his own ip address. Thus victim is connected to the web server. Then the ARP poisoning gets activated[14].

### **4.1.3 Packet Parse :**

Parsing generally means resolving into components or analysing into components. A packet's round time is set. the clocks ticks only fifty or hundred time per second. importing these is needed for openflow, topology and switches and also versions.

ev.message

this is used for packet data structure. ev,message is an object that represents a packet in data structure.

msg.dp

msg.dp represents a datapath. this is also an object.

`dp.ofproto`

`dp.ofproto` represents an openflow protocol.

`dp.ofproto_parser`

`dp.ofproto_parser` is an object that represents an openflow protocol and negotiate the ryu.

Ethernet is used when bit rate and link address is longer. ARP maps ip address to hardware address.

```
ether = pkt['eth'] = pkt['pkt'].get_protocols(ethernet.ethernet)[0]
```

Ethernet packet is initialized. Ether is the name and any other name can be initialized instead of ether. After we have initialized packet, we will define other parameters. Parameters will be declared in this section. Every packet needs a fix amount of parameters. Sometimes parameters can be 10 for a packet and sometimes parameters can be 15 or even sometimes can be 11. We have declared packets for Udp, tcp, icmp, ipv4, ipv6 etc.

#### **4.1.4 Network Address Translator**

We are using a rogue host here that is a lighttpd web server. When a device wants to connect to the internet it needs a public ip address. A private ip can not be used for connecting to the internet. Nat translates this public ip to private ip address. That means a host enters to its gateway by its private ip but when it leaves the gateway it owns a public ip[15]. Then that private ip gets access to internet.

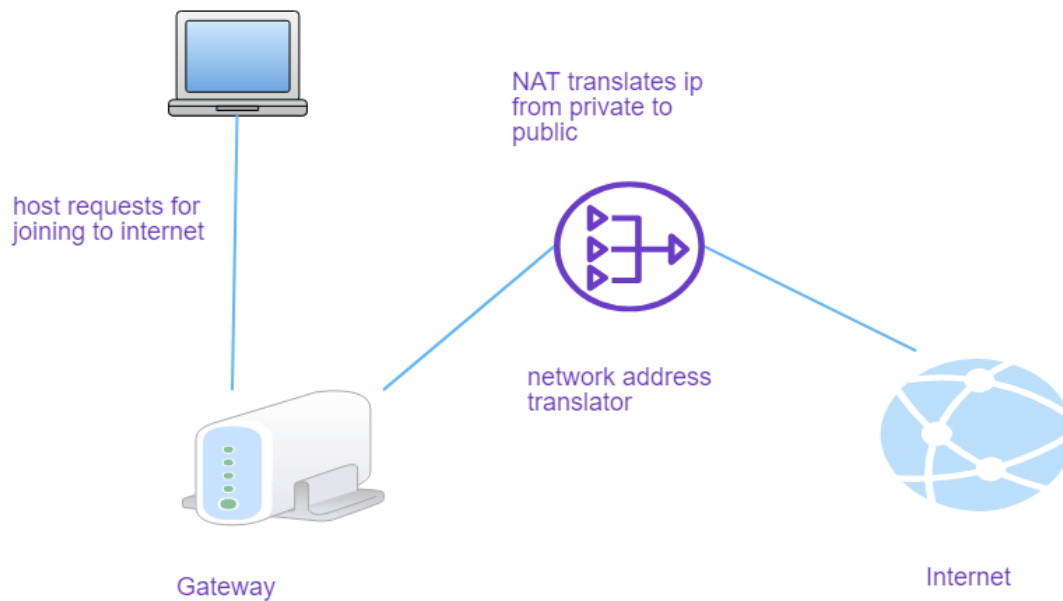


Fig 4.2: Dhcp Server

When a public ip is made private , for receiving packets from that ip , or when an ARP request arrives or when ARP reply is provided then a proxy ARP is needed. Router interface needs to save all the information about that public ip, but when we use a proxy ARP ; it saves all the information about public ip. Proxy ARP helps to provide response to the ARP requests. Proxy ARP is typically configured to a networking device where network address translator is present. Then it can accept all packets from the translated ip address.

#### 4.1.5 Lighttpd Web Server

A lighttpd web server is added to the topology for finding packet loss, delay and link loss rate. Server is used for proving that spoofing activity is running on mininet.

Target ip uses its username and password for logging into the web server. Then the attacker gets the user\_id and password using mininet commands.

#### 4.1.6 Detection Code

At the very beginning we need to set a minimum number of trials.

physhark

subprocess

blockip

blockIP1

physhark is used for capturing packets. Subprocesses are being called so that we can reject a request which have been generated from a blocked ip address. Blockip and blockIP1 is defined to check whether a request has arrived from the blocked ip address.

len(offers)

A minimal number of trials have been established. If the counter value exceeds the minimal number of trials then we will not accept that request.

DHCPOffer:

This is a class. Here it is checked whether it is dhcp offer or not.

DHCPDiscover

It is checked whether it is dhcp acknowledgement , if so then the value returns a true value.

If both of them returns a false value , then mac is checked on the table.

Port\_range

Staticmethod

It is checked using the above two functions whether a port is static or not. If the value is false then checks the assigned state value by using the below terminologies.

s.bind(("x", x))

s.getsockname()

If the value matches then state is assigned another value and it is 1. If state is not assigned with 0 then the packet is ignored. When acknowledgement and offer returns a true value then mac is checked on the table using the below queries:

m getmac -v -d -i enp11s4

This is used for finding the mac on table, if mac is on table then it is checked whether port is static. If port is static then packet is ignored. Else ip address is matched , then state is checked. then a verified state is gained.

phyawall

make\_rule

If ip is not found on table then mac is added to the table. As it indicates that the request has arrived for the first time and then it is required to add a temporary entry to the table. After verifying all the states if we find that the added entry is poisonous then we will remove the entry from table. for checking static ports we use the queries below:

fixed= true

fixed=false

If the port is static then the fixed has true value and if the port is not static then the port has a false value. So we need to check the fixed containing true values, then we will drop those packets. From the dynamic table these kind of entries are dropped.

Controller runs the detection procedure by itself and detects poisonous entries[21]. If the entries are found fishy then controller tells the switch to drop a port connection. This is used as a prevention measure.

handl\_port\_stat

This class tells the switch to disconnect the ports. The entries are dropped from dynamic table. After cleaning up and disconnecting ports the controller goes down. The self method contains the parameters such as block, snit, allow, i\_iface etc.

Add\_rule is defined in phyawall class. Here entires are popped up if they seem to be poisonous.

## 4.2 Chapter Summary

In this section we briefly considered all the devices and provided a brief description of them containing how they works, why they have been used in this thesis work. We also provided a basic topology for performing the detection procedure. Here all the devices working procedure have been discussed . Then the detection process was described briefly here. As we are using pyretic for detection procedure we used some terminologies for providing a basic idea about

how the detection measure works. Again we provided some list of Figures for understanding the topology clearly.

## **Chapter 5**

### **Experimental Result and Discussion**

In this section, we will discuss the results obtained from the proposed algorithm and the existing algorithm. Here we will compare the packet loss , link loss, delay modifying the proposed method. Existing method uses the pox controller. We will be using ryu controller for getting comparison with the above mentioned parameters. Again number of hosts were added on the proposed method for comparing results. A rogue server which was named as lighttpd server, was added to the existing topology. Then the packet loss, delay, link loss rate etc. was measured. In this attachment, we are providing a list of comparisons showing how packet loss, delay and link loss rate differs with the changes.

#### **5.1 Experimental Setup**



In this section we will be discussing about the topologies we have used in this thesis work. We will also discuss about the topologies and then we will show flow charts for describing the whole procedure. below is a list of topologies and flow charts for representing the overall processes of the thesis work.

### 5.1.1 Topology for Existing Method

In this section we are going to describe the existing topology and hence we are going to describe how the detection and prevention is performed in this part of the thesis.

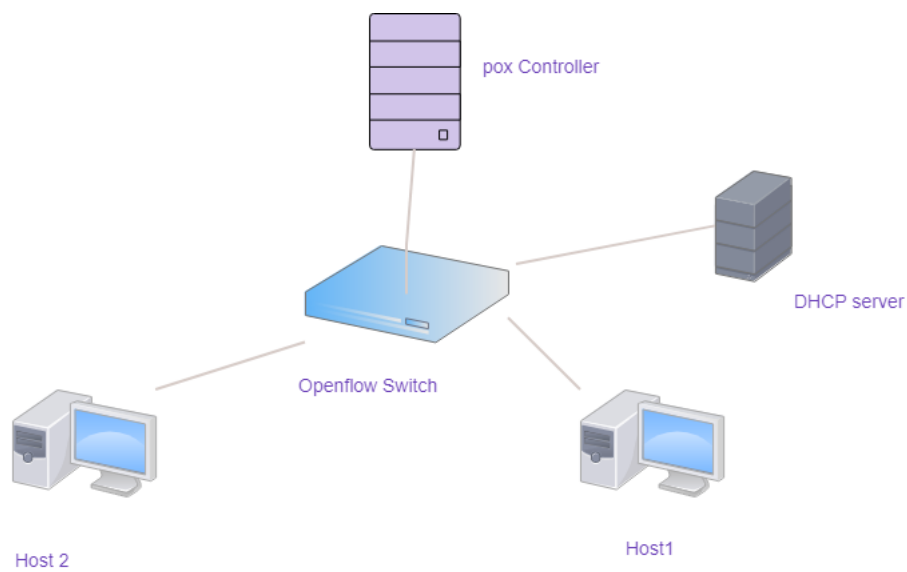


Fig 5.1: Existing Topology

The detection processes are shown using a flow chart which is provided in the following part.

In this section, we have shown how detection process is continued in mininet. For showing the whole process we also showed what process is performed on which device. We have presented the whole process with the algorithmic steps and we also mentioned the devices here.

### 5.1.2 Proposed Method Topology

In this portion, we are presenting the proposed topology and the proposed method. The topology shown below uses a pox controller, dhcp server, openflow switch, hosts. Here the two hosts are attacker and the spoofer.

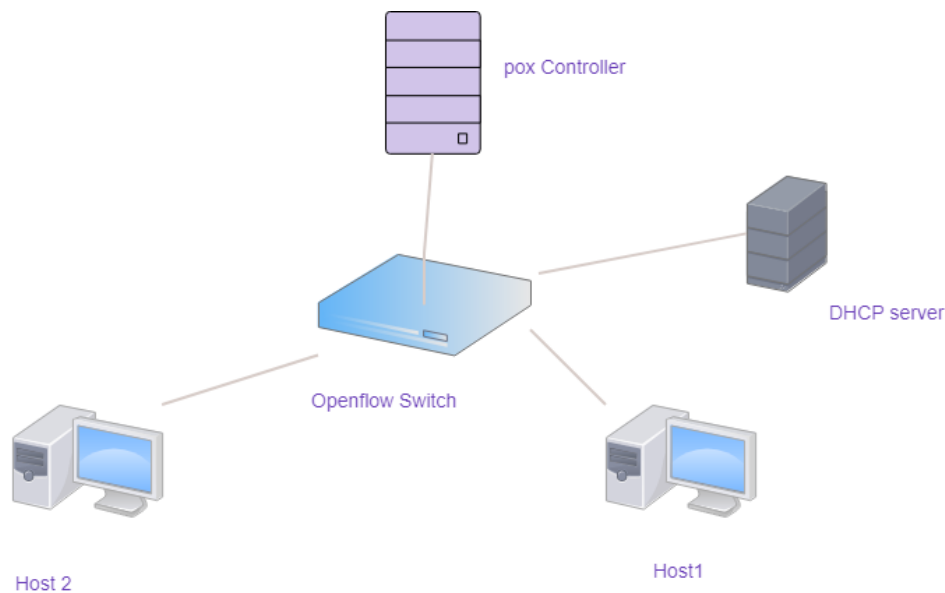


Fig5.2: Topology for Proposed Method

In this portion, we are showing a flow chart that explains the proposed method's working procedure. The flow chart is shown below.

Here we explained how the above topology works. Initially when the request arrives, it is checked whether it is an offer or acknowledgement. If the result is false then the mac is checked on the table. If the mac is found, then port is checked. If port is static then packet is ignored. If port is not static then state is checked ; meanwhile state was updated in another state. After that a new state value is assigned.

Again if the offer and acknowledgement results in a true value then mac is checked on table. If mac is not found then an entry will be added to the dynamic table. It indicates that the request has arrived for the first time. Then it also ensures that offer has been resulted in a true value.

Another part is when acknowledgement is true then the offer or acknowledgement results in a true value, then mac is checked on the table. If mac is found then ip is matched and the state is also matched. If both of them results in a true value then the verification state is achieved. Thus the process works.

Till this part it is similar as the existing one. We have set a counter for counting the number of time an ip request arrives. This is basically limiting the number of time an ip can send request to the system. We are using random numbers for state assignment.

### 5.1.3 Modified Topology with a Ryu Controller

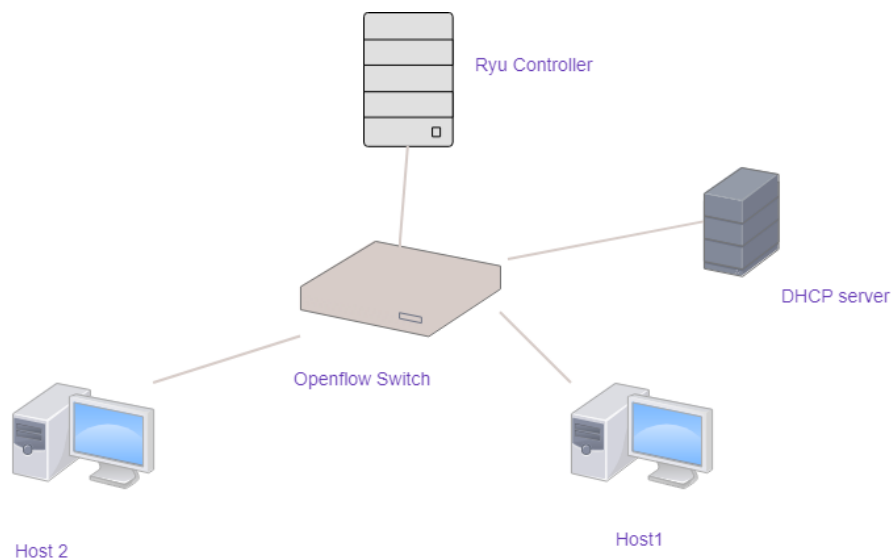


Fig5.3: Modified Topology for RYU Controller

Following is a flow chart showing the steps how this topology works. This flow chart contains all the steps for showing the whole implementation.

The flow chart is shown below.

This topology works just the same as the flow chart described above. The basic difference is we used a pox controller in the above topology and then we used a ryu controller for this

detection[16] method. Then we are comparing these modified methodologies. The comparison is shown in another chapter.

#### 5.1.4 Modified Topology with a Lighttpd Web Server

In this section we are using a lighttpd web server , basically it was done for providing a comparison among the testbed. We have been initialized a network address translator for involving into the web server. After that the detection process is being continued.

Following is a topology showing a NAT, an openflow switch, a dhcp server, hosts, lighttpd web server and a controller.

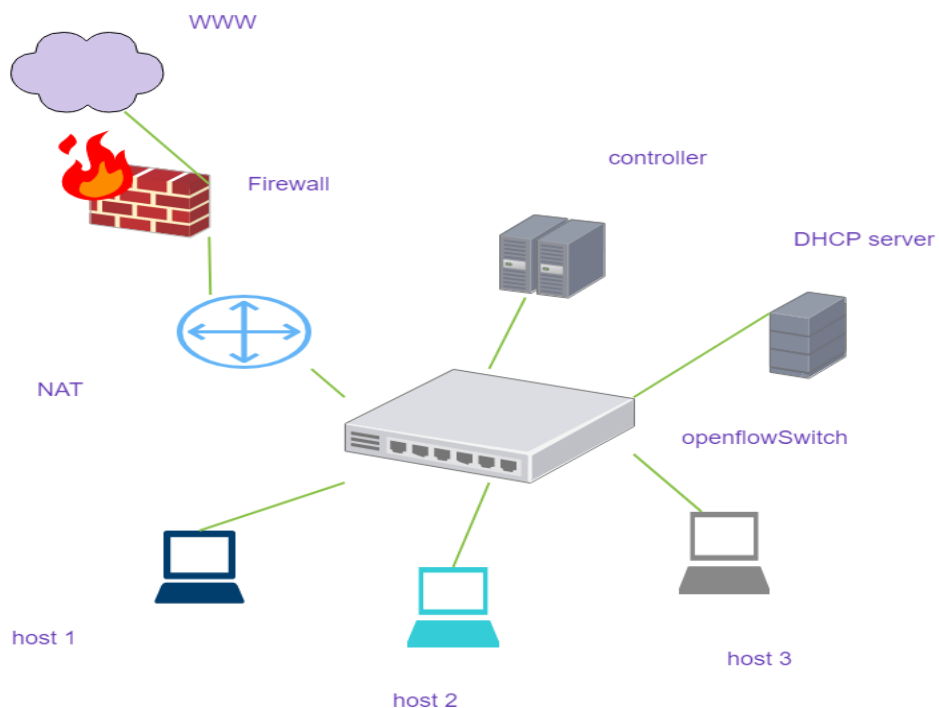


Fig5.4: Modified Topology with Lighttpd Web Server

Below we are presenting a flow chart showing how the whole process works in the above topology.

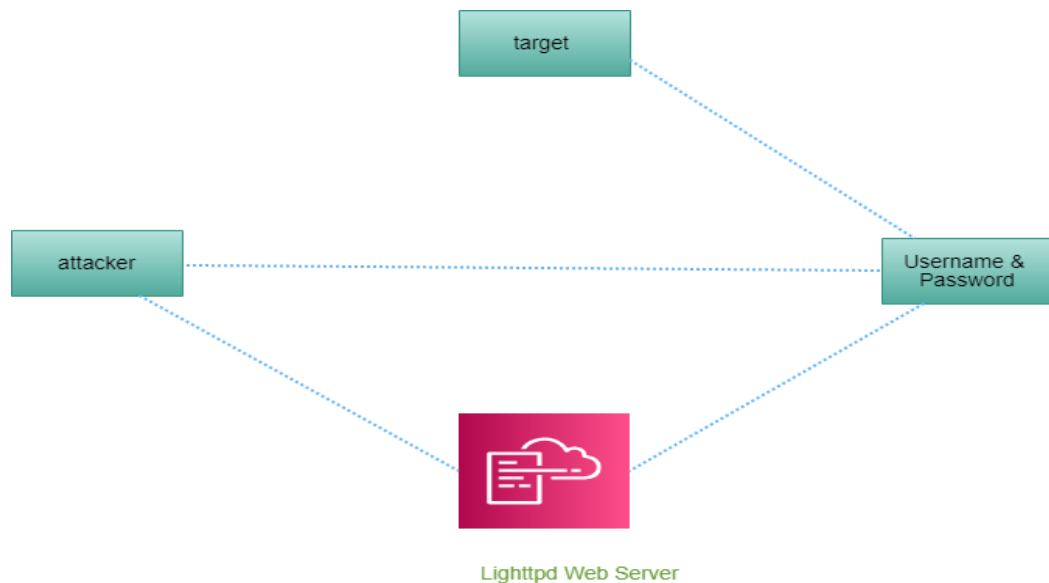


Fig 5.5: Use of network address translator

When a spoofing activity is performed initially. After that when a target ip uses its username and password for logging into the web server , the attacker gets target's user name and password. Thus the attacking activity has been performed.

### 5.1.5 Modified Topology with Additional Hosts

We are modifying the proposed topology for gathering some comparison. Hence we are adding hosts to the proposed topology and we are following the same algorithm for generating the detection procedure to a success. The topology is same as before. Just a slight difference is we are using a number of hosts instead of only two hosts. Before we were using only two hosts , they were the target and the attacker. These two hosts were used in the above topologies , but here we have added a lot of hosts to the topology. Another difference is we are not using any lighttpd web server here.

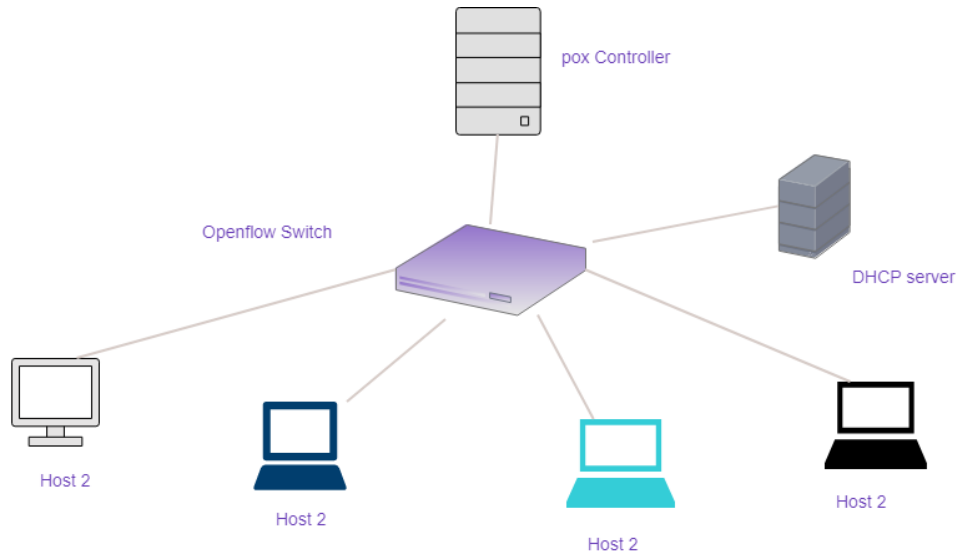


Fig5.6: Modified Topology with Additional Host.

This methodology follows the same algorithmic processes as show in the above flow chart before. The same processes are running on the switch, dhcp and controller.

## 5.2 Experimental Results

### 5.2.1 Existing Algorithm

Packet\_parse means for parsing or resolving into components. Different types of packets were declared. tcp, udp, ARP etc. have fixed amount of parameters and those were briefly initialized while declaring each packets.

```
File Edit View Search Terminal Help
root@kali:~# cd /root/Downloads
root@kali:~/Downloads# python pkt.py
Start Packet parsing on host with specified DNS server
* Starting Rogue packet parser server on
at
host default interface
Batch File:
pktdhcp_bt_file
CHADDR:
dhcp_chaddr
print CHADDR:
pkt_dhcp_ciaddr
Flags
pkt_dhcp_flags
GIADDR
pkt_dhcp_giaddr
GIADDR:
pkt_dhcp_giaddr
HLEN:
pkt_dhcp_hlen
Hops:
pkt_dhcp_hops
Htype
pkt_dhcp_htype
```

Fig5.7: packet Parse on mininet

Different types of packets have been initialized in this portion. Tcp, Udp , ARP or other types of packets have been initialized in this portion.

## Dhcp:

By creating a dhcp configuration file, finally dhcp server is activated and deactivated again and again. Host's links are configured using necessary parameter.it is basically used for assigning gateway, spoofer and target ip.

```
root@kali: ~/Downloads
File Edit View Search Terminal Help
root@kali:~# cd /root/Downloads
root@kali:~/Downloads# python dhcp.py
starting the dynamic host configuration protocol(DHCP)
creating mininet configuration with DHCP server
dhcp running!
*** Creating network
*** Adding controller
**** Adding hosts:
*** dhcp evil h1
*** *** Starting controller
*** Starting 1 switches
*** s1 (10.00Mbit 500ms delay)
*** Starting DHCP server on dhcp at 10.0.0.50
*** h1 waiting for IP address...
*** h1 is now using nameserver 8.8.8.8
*** Fetching google.com:
<HTML><HEAD><meta http-equiv="content-type" content="text/html; charset=utf-8">
<TITLE>301 Moved</TITLE></HEAD><BODY>
<H1>301 Moved</H1>
<A HREF="http://www.google.com/">here</A>.
</BODY></HTML>
You may want to do some DNS lookups using dig
*** Please go to amazon.com in Firefox
*** You may also wish to start up wireshark and look at bootp and/or dns
```

Fig5.8: Setting Dhcp Serever on Mininet

Here dhcp server was initialized on mininet.

```
root@kali: ~/Downloads
File Edit View Search Terminal Help
starting the dynamic host configuration protocol(DHCP)
creating mininet configuration with DHCP server
dhcp running!
*** Creating network
*** Adding controller
**** Adding hosts:
*** dhcp evil h1
*** *** Starting controller
*** Starting 1 switches
*** s1 (10.00Mbit 500ms delay)
*** Starting DHCP server on dhcp at 10.0.0.50
*** h1 waiting for IP address...
*** h1 is now using nameserver 8.8.8.8
*** Fetching google.com:
<HTML><HEAD><meta http-equiv="content-type" content="text/html; charset=utf-8">
<TITLE>301 Moved</TITLE></HEAD><BODY>
<H1>301 Moved</H1>
<A HREF="http://www.google.com/">here</A>.
</BODY></HTML>
You may want to do some DNS lookups using dig
*** Please go to amazon.com in Firefox
*** You may also wish to start up Wireshark and look at bootp and/or dns
*** Press return to start up evil DHCP server:
root@kali:~/Downloads#
```

Fig5.9: Setting Dhcp Server on Mininet

```
root@kali: ~/Downloads
File Edit View Search Terminal Help
root@kali:~# cd /root/Downloads
root@kali:~/Downloads# python dhcp_openflowswitch.py
starting the openflow switch with dhcp server
creating mininet configuration with DHCP
Start DHCP server on host with specified DNS server
* Starting Rogue DHCP server on
at
host default interface
/tmp/%s-udhcpd.conf
udhcpd -f
1>/tmp/%s-dhcp.log 2>&1 &
**UDHCPD from
is running.**
*Stop DHCP server on host
Stopping DHCP server
at
kill %udhcpd
* Start DHCP client on
* Intf for
dhclient -v -d -r
dhclient -v -d -l> /tmp/dhclient.log 2>&1
&
kill %dhclient
main
```

Fig5.10: Setting Dhcp Server on Mininet



```
root@kali: ~/Downloads
File Edit View Search Terminal Help
/tmp/%s-udhcpd.conf
udhcpd -f
1>/tmp/%s-dhcp.log 2>&1 &
**UDHCPD from
is running.**
*Stop DHCP server on host
Stopping DHCP server
at
kill %udhcpd
* Start DHCP client on
* Intf for
dhclient -v -d -r
dhclient -v -d -l> /tmp/dhclient.log 2>&1
&
kill %dhclient
__main__
*** Starting Mininet ***
ovs-vsctl set bridge s1 protocols=OpenFlow13
***Attempting to start dhcp server***
*** sudo /etc/init.d/isc-dhcp-server start
***** Running CLI *****
Stopping Network
sudo /etc/init.d/isc-dhcp-server stop
root@kali:~/Downloads#
```

Fig5.11: Setting Dhcp Server on Mininet

## POX Controller:

```
root@kali: ~/Downloads
File Edit View Search Terminal Help
root@kali:~# cd /root/Downloads
root@kali:~/Downloads# python pox openswitch.py
starting the openflow switch associated with pox
creating mininet configuration with POX & SWITCH
*** Removing excess controllers/ofprotocols/ofdatapaths/pings/noxes
killall -9 controller ofprotocol ofdatapath ping nox_core lt-nox_core ovs-open
dpbwtst 2> /dev/null
*** Removing junk from /tmp
rm -f /tmp/vconn* /tmp/vlogs* /tmp/*.out /tmp/*.log
*** Removing old screen sessions
*** Removing excess kernel datapaths
ps ax | egrep -o [dp[0-9]+] | [sed s/dp/nl:]/
*** Removing OVS datapaths
ovs-vsctl list-br
*** Removing all links of the pattern foo-ethX
ip link show | egrep -o (\w+-eth\w+)
*** Cleanup complete.
*** Shutting down stale SimpleHTTPServers
*** Shutting down stale web servers
server1 172.64.3.21
server2 172.64.3.22
client 10.0.1.100
sw0-eth1 10.0.1.1
sw0-eth2 172.64.3.1
*** Successfully loaded ip settings for hosts
```

Fig5.12: Configuring POX Controller on Mininet

In these terminals shown above and below we are showing how POX controller was initialized on mininet. These terminals show the setting up process of POX controller on mininet.

```
root@kali: ~/Downloads
File Edit View Search Terminal Help
*** Creating network
*** Creating network
*** Adding controller
*** Adding hosts:
client server1 server2
*** Adding switches:
sw0 sw1
*** Adding links:
(client, sw0) (server1, sw1) (server2, sw1) (sw0, sw1)
*** Configuring hosts
client server1 server2
*** Starting controller
*** Starting 2 switches
sw0 sw1
*** setting default gateway of host server1
server1 172.64.3.1
*** setting default gateway of host server2
server2 172.64.3.1
*** setting default gateway of host client
client 10.0.1.1
*** Starting SimpleHTTPServer on host server1
*** Starting SimpleHTTPServer on host server2
*** Starting CLI:
root@kali:~/Downloads#
```

Fig5.13: Configuring POX Controller on Mininet

## Openflow Switch:

switch holds a port for connecting hosts to it. the ports are disconnected when a poisonous entry is found. so it is necessary to verify whether a host connected to the port is valid or not. switch drops a packet by disconnecting a port when the controller tells to do so. otherwise switch works as a learning switch.

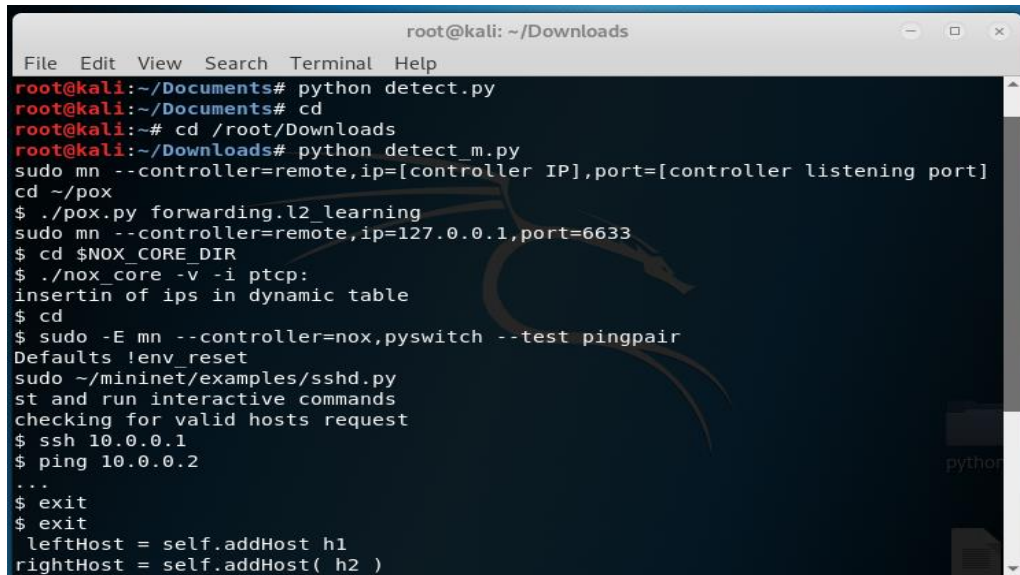
```
root@kali: ~/Documents
File Edit View Search Terminal Help
root@kali:~# cd /root/Documents
root@kali:~/Documents# python switch.py
starting the openflow switch
creating mininet configuration with openflow switch
starting the openflow switch in mininet
CNTRL++
switch++
| +-----+h1
| +-----+h2
+-----+s1+-----+h3
+-----+switch+-----+h4
+-----+h5
*** Adding Switch **
*** Adding hosts (h1, h2, and h3) ***
*** Linking hosts to s1 ***
*** Connecting to switch (s1) ***
*** Creating hosts h4 and h5 ***
*** Linking h4 and h5 to switch ***
*** Starting Mininet ***
*** Topology Created***
*** Running CLI ***
*** Assigning IPs to switch1-eth1 and switch1-eth2 ***
*** Configuring IP Tables for switch1 (allowing second interface).
*** Stopping CLI ***
root@kali:~/Documents#
```

Fig5.14: Configuring Openflowswitch on Mininet

This terminal shows how an openflow switch was set up on mininet.

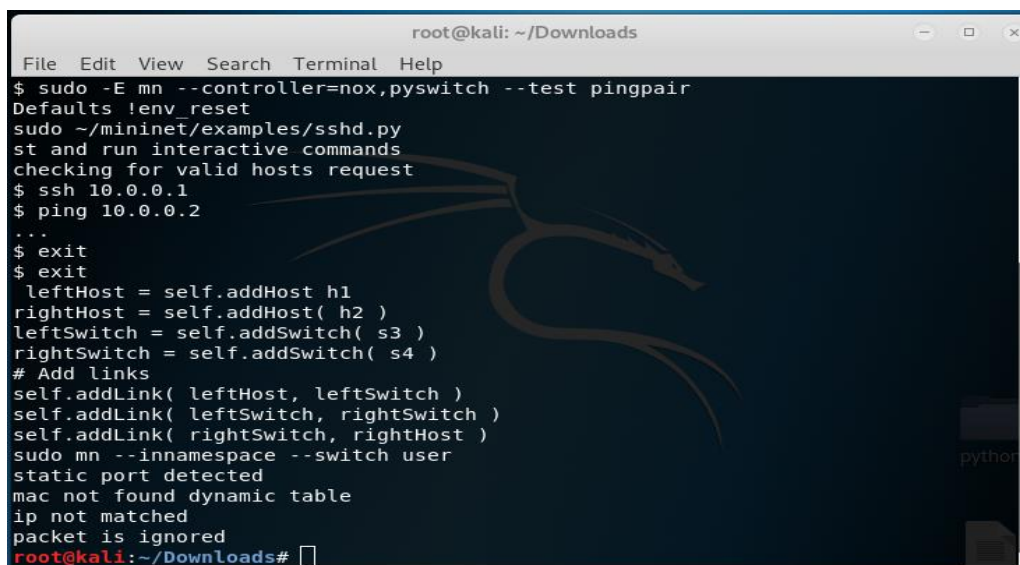
## Detection Code:

detection algorithm is implemented and then code was run on mininet. the output terminal is shown below.

A terminal window titled 'root@kali: ~/Downloads' showing the execution of a detection script. The user runs 'python detect.py' in ~/Documents, then 'cd /root/Downloads' and 'python detect\_m.py'. They then run 'sudo mn --controller=remote,ip=[controller IP],port=[controller listening port]', 'cd ~/pox', and './pox.py forwarding.l2\_learning'. Next, they run 'sudo mn --controller=remote,ip=127.0.0.1,port=6633', 'cd \$NOX\_CORE\_DIR', and './nox\_core -v -i ptcp:'. This results in an 'insertin of ips in dynamic table'. Finally, they run 'cd', 'sudo -E mn --controller=nox,pyswitch --test pingpair', 'Defaults !env reset', 'sudo ~/mininet/examples/sshd.py', and 'st and run interactive commands'. The terminal shows 'checking for valid hosts request', 'ssh 10.0.0.1', 'ping 10.0.0.2', and several 'exit' commands. At the bottom, there are two lines of Python code: 'leftHost = self.addHost h1' and 'rightHost = self.addHost( h2 )'.

```
root@kali: ~/Downloads
File Edit View Search Terminal Help
root@kali:~/Documents# python detect.py
root@kali:~/Documents# cd
root@kali:~/Downloads# python detect_m.py
sudo mn --controller=remote,ip=[controller IP],port=[controller listening port]
cd ~/pox
$ ./pox.py forwarding.l2_learning
sudo mn --controller=remote,ip=127.0.0.1,port=6633
$ cd $NOX_CORE_DIR
$ ./nox_core -v -i ptcp:
insertin of ips in dynamic table
$ cd
$ sudo -E mn --controller=nox,pyswitch --test pingpair
Defaults !env reset
sudo ~/mininet/examples/sshd.py
st and run interactive commands
checking for valid hosts request
$ ssh 10.0.0.1
$ ping 10.0.0.2
...
$ exit
$ exit
leftHost = self.addHost h1
rightHost = self.addHost( h2 )
```

Fig5.15: Running Detection Code on Controller

A terminal window titled 'root@kali: ~/Downloads' showing the execution of a detection script. The user runs 'sudo -E mn --controller=nox,pyswitch --test pingpair', 'Defaults !env reset', 'sudo ~/mininet/examples/sshd.py', and 'st and run interactive commands'. The terminal shows 'checking for valid hosts request', 'ssh 10.0.0.1', 'ping 10.0.0.2', and several 'exit' commands. Then, they run 'leftHost = self.addHost h1', 'rightHost = self.addHost( h2 )', 'leftSwitch = self.addSwitch( s3 )', and 'rightSwitch = self.addSwitch( s4 )'. They then run '# Add links', 'self.addLink( leftHost, leftSwitch )', 'self.addLink( leftSwitch, rightSwitch )', and 'self.addLink( rightSwitch, rightHost )'. Finally, they run 'sudo mn --innamespace --switch user', which results in 'static port detected', 'mac not found dynamic table', 'ip not matched', and 'packet is ignored'. The terminal ends with 'root@kali:~/Downloads#'.

```
root@kali: ~/Downloads
File Edit View Search Terminal Help
$ sudo -E mn --controller=nox,pyswitch --test pingpair
Defaults !env reset
sudo ~/mininet/examples/sshd.py
st and run interactive commands
checking for valid hosts request
$ ssh 10.0.0.1
$ ping 10.0.0.2
...
$ exit
$ exit
leftHost = self.addHost h1
rightHost = self.addHost( h2 )
leftSwitch = self.addSwitch( s3 )
rightSwitch = self.addSwitch( s4 )
# Add links
self.addLink( leftHost, leftSwitch )
self.addLink( leftSwitch, rightSwitch )
self.addLink( rightSwitch, rightHost )
sudo mn --innamespace --switch user
static port detected
mac not found dynamic table
ip not matched
packet is ignored
root@kali:~/Downloads#
```

Fig5.16: Running Detection Code on Controller

In the above terminals we have been running all the devices on mininet.

## Merging All the Devices to Create a Mininet Topology:

We will be using the following specifications for creating a merged topology that contains all the devices. The processes are shown below.

```
>sudo mn topo linear
```

this command will create a mac switch ovsk controller. Then a network will be created. A controller will also be added, hosts will also be added. This portion will use a temporary host name. In our example the hosts are:

```
h1s1    h1s2          h1s3          h2s1    h2s2          h2s3
```

Then the switches will be added. In our work switches are:

```
s2    s3
```

After that links will be added. We have used the following links.

```
(h1s1,s1) (h1s2,s2) (h1s3,s3) (h2s1,s1)
(h2s2,s2) (h2s3,s3) (s2,s1) (s3,s2)
```

The controller is named as:

```
C0
```

Then we will be using the command

```
sudo ovs-ofctl show s2
```

Then ofpt\_features\_reply will be arrived.

Then we will be using another command and that is

```
sudo ovs-ofctl show s2
```

ofpt\_features\_reply will provide dpid. Here the dpid is 00000000000000000002. Then the capabilities and actions are shown. enqueue basically shows the switch port and the Ethernet port . Then for a particular mac address current, speed , conFig and states are shown.

```
enqueue
```

```
s2_eth1
```

```
conFig: 0
```

```
state:0
```

```
addr: 0a:6b:3c:f7:5f:e1
```

After executing the above commands we are going to set up our controller. In this case we are using pox controller. So we will use the following command:

```
cd pox
```

```
sudo su
```

Then pox.py will be called and pox will be started through mininet. Then that pox controller is up.

Whenever the core pox is up, it will be running on Cpython. Openflow of\_01 will be listening on 0.0.0.0.6633 and when we want to test the connectivity we will use the following command:

```
mininet>pingall
```

Here ping reachability will be tested. And the output will be like below:

```
h1s1 -> h1s1 h1s3 h2s1 h2s2 h2s3
```

In this section openflow switch will be started. Here also reachability will be tested. Then the result is shown below:

Results: 0% dropped (30/30 received)

After that dhcp server will be on and the results will be same as explained before in openflow switch. Percentage of dropped packets are also shown here.

Tcp bandwidth will also be tested and the result is:

```
Ipref: testing TCP bandwidth between h1s1 and h2s3
```

Results:

1.76

Gbits/sec

flow tables can be accessed and it can also be controlled. Flow tables can also be edited. The command shown below gives us the result:

```
Dpctl
```

Tcp speed can be tested and the command is shown below.

```
Iperf
```

We are also able to see the list of nodes available and for this we need to type the following command.

```
mininet>help
```

```
log.level_DEBUG misc.of
```

This tells POX to enable verbose logging and of\_component is started. In a new SSH terminal a topology was shown. Nodes connections were wired links and then they are shown as below:

```
tcpdump xx n I h2_eth0
```

```
tcpdump xx n I h3_eth0
```

Then ping requests are sent and we can also ping the controller C0 to test connectivity.

Now we will be showing how results have been shown on mininet. Mininet uses a tool that is called Wireshark. Using Wireshark we will generate a warning message if there exists any duplicate IP address.

## WIRESHARK:

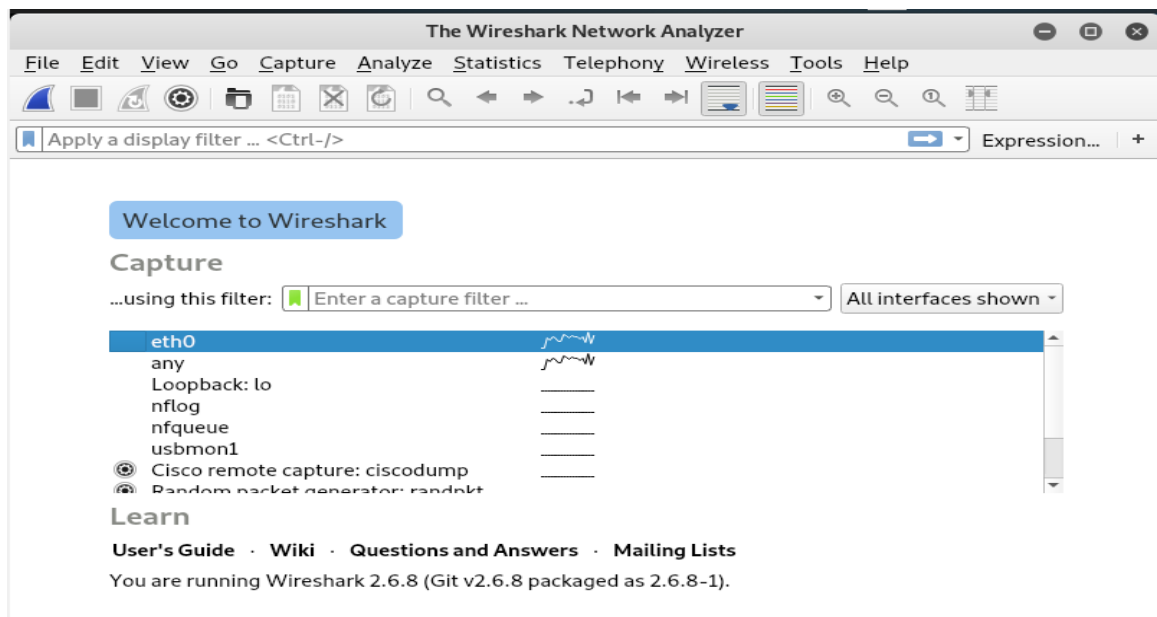


Fig5.17: Initializing Wireshark tool

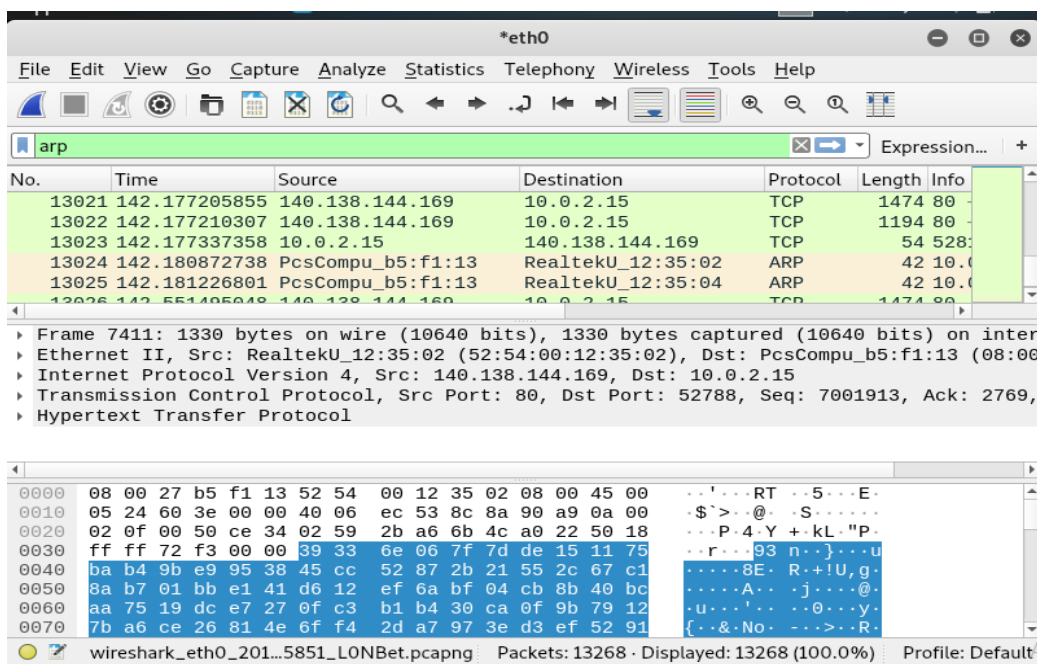


Fig5.18: Detecting ARP Poisoning

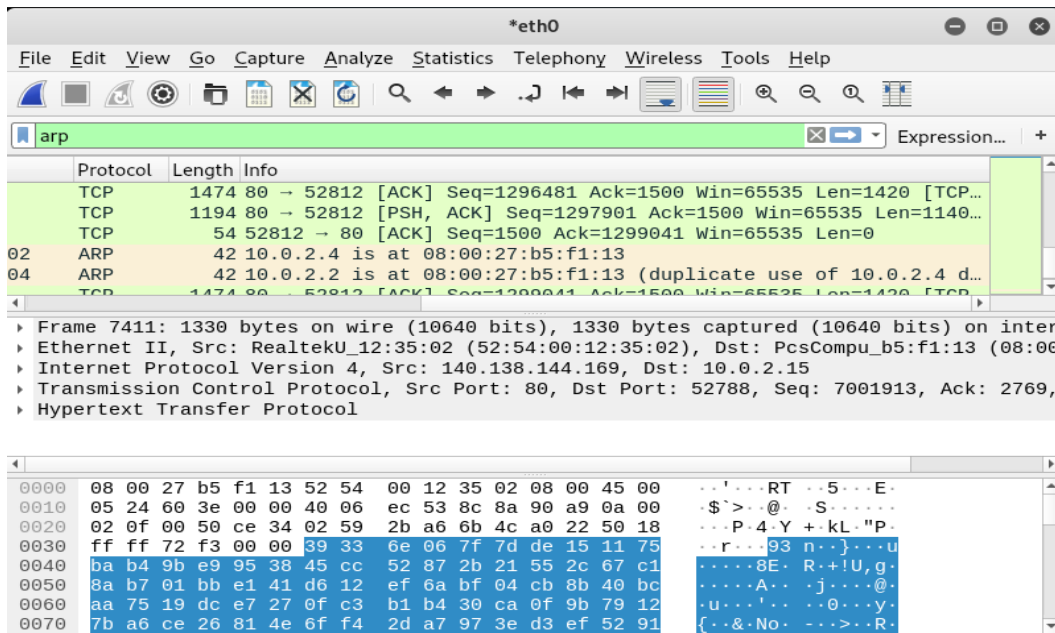


Fig5.19: Detecting Duplicate use of IP

Here duplicate use of ip is detected. As the target thinks that the attacker's ip address is its gateway when the gateway has its own ip address. Thus in a network there exists two similar ip addresses. Thus the attack is detected when two ip's are same. Wireshark uses a warning message that is

‘duplicate use of ip’

## 5.2.2 Proposed Algorithm

Proposed algorithm also requires to set up the devices on mininet. Then the proposed method will combine all the devices on mininet. After that the detection code also needed to be run on mininet and the poisoning code is also needed to be run on mininet. There is no terminal shown for poison code as the python cado is just run as a python code on mininet and like others there is no output terminal showing the set up on mininet.

Hence the setting up of all the devices and other necessary tools have been shown in the previous section. We are not going to repeat that again.

We will show the output here, in the proposed method we are using the following tool for showing output. The outputs are shown below:



The topology were used as before. Topology where we used dhcp, openflow switch, pox controller, hosts, network address translator. They were set on mininet. After setting them on mininet, we used terminals for showing them. They were shown on the immediate previous section. So in this section we have shown the detection output only[17].

	IP	MAC	Host	Vendor	Interface	Online	Cache	First seen	
✗	137.59.183.177	64-d1-54-3b-32-54	137.59.183.177	unknown	0x8 - Realtek Et...	yes	no	21/07/2019	
✗	169.254.144.4	84-a9-3e-09-01-24	169.254.144.4	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019	
✗	169.254.174.55	f8-ca-b8-04-a6-ce	169.254.174.55	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019	
✗	169.254.230.223	ac-e2-d3-6c-6d-16	169.254.230.223	unknown	0x8 - Realtek Et...	no	no	21/07/2019	
✗	192.168.1.1	80-14-a8-31-f3-c3	192.168.1.1	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019	
✓	192.168.1.11	1c-1b-0d-65-83-02	192.168.1.11	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019	
✗	192.168.1.22	84-34-97-16-31-96	DESKTOP-THD...	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019	
✓	192.168.1.23	68-ff-7b-e0-0d-ad	192.168.1.23	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019	
✓	192.168.1.25	2c-56-dc-a9-27-4e	192.168.1.25	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019	
✓	192.168.1.32	6c-c2-17-ef-0b-b5	192.168.1.32	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019	
✓	192.168.1.34	dc-4a-3e-e2-9c-dc	192.168.1.34	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019	
✓	192.168.1.35	f4-30-b9-13-fa-c4	DESKTOP-5RK...	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019	
✓	192.168.1.37	6c-c2-17-78-44-74	192.168.1.37	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019	
✗	192.168.1.40	a0-1d-48-ad-a9-1c	DESKTOP-FNP...	unknown	0x8 - Realtek Et...	yes	no	21/07/2019	

Fig5.20: Detecting ARP Spoofing

	IP	MAC	Host	Vendor	Interface	Online	Cache	First seen	
✓	192.168.1.37	6c-c2-17-78-44-74	192.168.1.37	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019	
✗	192.168.1.40	a0-1d-48-ad-a9-1c	DESKTOP-FNP...	unknown	0x8 - Realtek Et...	yes	no	21/07/2019	
✓	192.168.1.44	88-d7-f6-1d-c7-16	192.168.1.44	unknown	0x8 - Realtek Et...	no	yes	21/07/2019	
✓	192.168.1.45	84-7b-eb-29-6f-7a	DESKTOP-THT...	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019	
✓	192.168.1.46	1c-b7-2c-3b-4c-42	192.168.1.46	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019	
✗	192.168.1.47	00-e0-4c-c0-34-6a	DESKTOP-51K...	Realtek Semic...	0x8 - Realtek Et...	yes	yes	21/07/2019	
✗	192.168.1.48	ac-e2-d3-55-35-a7	DESKTOP-SGJ0...	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019	
✓	192.168.1.49	94-57-a5-b0-d2-44	192.168.1.49	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019	
✓	192.168.1.51	54-e1-ad-90-9f-1a	192.168.1.51	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019	
✓	192.168.1.53	34-17-bf-62-84-86	DESKTOP-FHC...	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019	
✓	192.168.1.57	4c-cc-6a-28-74-f6	192.168.1.57	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019	
✓	192.168.1.59	18-31-bf-17-97-74	192.168.1.59	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019	
✗	192.168.1.60	e0-d5-5e-16-02-ad	192.168.1.60	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019	
✓	192.168.1.62	4c-ed-fb-d8-4b-cf	Bayazid	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019	

Fig5.21: Detecting ARP Spoofing

	IP	MAC	Host	Vendor	Interface	Online	Cache	First seen	
✗	192.168.1.60	e0-d5-5e-16-02-ad	192.168.1.60	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019	
✗	192.168.1.62	4c-ed-fb-d8-4b-cf	Bayazid	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019	
✓	192.168.1.67	38-d5-47-48-4f-4b	192.168.1.67	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019	
✓	192.168.1.69	b0-5a-da-e7-60-6e	192.168.1.69	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019	
✓	192.168.1.70	70-85-c2-53-2e-93	192.168.1.70	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019	
✓	192.168.1.71	30-9c-23-17-80-0c	192.168.1.71	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019	
✓	192.168.1.73	dc-4a-3e-f6-95-93	192.168.1.73	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019	
✓	192.168.1.76	e0-db-55-a7-9a-d1	192.168.1.76	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019	
✓	192.168.1.79	20-47-47-df-64-12	192.168.1.79	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019	
✓	192.168.1.80	fc-3f-db-89-fd-13	192.168.1.80	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019	
✓	192.168.1.81	5c-26-0a-3b-be-1f	192.168.1.81	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019	
✗	192.168.1.82	48-ba-4e-5b-50-73	DESKTOP-Q2B...	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019	
✗	192.168.1.86	c8-d3-ff-74-7b-39	SAZZAD-PC	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019	
✗	192.168.1.89	1c-1b-0d-c2-67-bb	DESKTOP-3SJ5...	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019	

Fig5.22: Detecting ARP Spoofing

	IP	MAC	Host	Vendor	Interface	Online	Cache	First seen
✗	192.168.1.86	c8-d3-ff-74-7b-39	SAZZAD-PC	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019
✗	192.168.1.89	1c-1b-0d-c2-67-bb	DESKTOP-3SJ5...	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019
✓	192.168.1.95	50-65-f3-08-a7-e8	192.168.1.95	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019
✓	192.168.1.102	d8-cb-8a-70-3f-f6	192.168.1.102	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019
✓	192.168.1.109	08-62-66-70-df-84	192.168.1.109	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019
✗	192.168.1.120	c8-5b-76-1a-29-ce	DESKTOP-VJM...	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019
✓	192.168.1.128	c4-54-44-30-c0-19	192.168.1.128	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019
✗	192.168.1.130	78-24-af-9c-2e-cb	Taufique	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019
✓	192.168.1.131	6c-c2-17-67-88-74	192.168.1.131	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019
✓	192.168.1.139	40-8d-5c-5c-9e-b5	192.168.1.139	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019
✓	192.168.1.143	0c-80-63-b6-69-47	192.168.1.143	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019
✗	192.168.1.148	2c-fd-a1-7e-e8-60	DESKTOP-941...	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019
✓	192.168.1.157	08-62-66-13-f4-c6	192.168.1.157	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019
✗	192.168.1.158	c4-54-44-d7-0e-17	DESKTOP-SHT...	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019

Fig5.23: Detecting ARP Spoofing

	IP	MAC	Host	Vendor	Interface	Online	Cache	First seen
✗	192.168.1.128	c4-54-44-30-c0-19	192.168.1.128	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019
✗	192.168.1.130	78-24-af-9c-2e-cb	Taufique	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019
✓	192.168.1.131	6c-c2-17-67-88-74	192.168.1.131	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019
✓	192.168.1.139	40-8d-5c-5c-9e-b5	192.168.1.139	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019
✓	192.168.1.143	0c-80-63-b6-69-47	192.168.1.143	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019
✗	192.168.1.148	2c-fd-a1-7e-e8-60	DESKTOP-941...	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019
✓	192.168.1.157	08-62-66-13-f4-c6	192.168.1.157	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019
✗	192.168.1.158	c4-54-44-d7-0e-17	DESKTOP-SHT...	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019
✓	192.168.1.160	70-8b-cd-05-bf-b9	192.168.1.160	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019
✓	192.168.1.162	d0-17-c2-1c-86-e9	192.168.1.162	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019
✓	192.168.1.163	4c-cc-6a-28-74-eb	192.168.1.163	unknown	0x8 - Realtek Et...	yes	yes	21/07/2019
✓	192.168.56.1	0a-00-27-00-00-11	DESKTOP-FNP...	unknown	0x11 - Oracle	yes	no	21/07/2019
✗	192.168.101.1	e0-67-b3-0f-97-c9	192.168.101.1	unknown	0x8 - Realtek Et...	no	no	21/07/2019
✗	192.168.137.1	20-47-47-da-6c-76	192.168.137.1	unknown	0x8 - Realtek Et...	no	no	21/07/2019

Fig5.24: Detecting ARP Spoofing

Above the red cross marked hosts represent the spoofed hosts and the green tick hosts are not ARP spoofed.

## 5.3 Finding Delay

### 5.3.1 Finding delay for Existing Algorithm

The time gap between the transmission time and the received time are used to find delay for packets. Here we may find the difference between the time a packet has been transmitted and the time when a packet was received. Thus we may calculate delay.

Here we are generating a full process for finding delay and we are also providing the full calculation for finding the delay. We will calculate delay for each and every packet transmitted and then we will show an average delay. Following we are showing the process in brief.

For delay calculation we have mentioned all necessary packages earlier at the very beginning of the code. Then some global variables have been declared and initially they all are set to zero.

SentTime=0.0

ReceivedTime=0.0

We are measuring delay by calculating two different one way delays. they are OWD1 and OWD2. Here we are finding a difference between owd1 and owd2. After finding the difference we are plotting the delay. Delay is represented in a fraction of seconds.

While getting the result we are making the difference between two values and then the result will be found. The formula used for delay calculation is shown below.

Find\_delay= owd1-owd2

Here owd1 is when a packet has been sent from one user to another and owd2 is when a packet is sent from the destination. In short , time owd1 measures the time when packet is sent from the sender may be as a request. And owd2 is the time when a packet is sent on behalf of the receiver. By calculating the difference between these two times we are finding the delay. If we represent owd1 as T1 and owd2 as T2, then the difference between T1 and T2 is the delay.

delay= T1-T2

Initially we are representing all the delay printed for each and every packet. Then we will calculate the average delay.

Averagedelay=(delay1+delay2+delay3+delay4+delay5+delay6+.....  
....+delay40+delay41+delay42)/total\_no.of received\_pkt

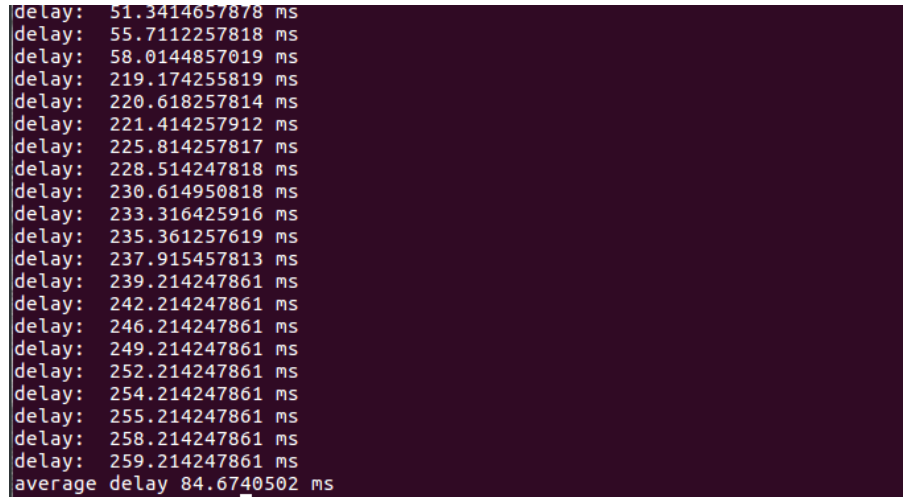
average delay= 63.0674657575 ms

```
delay: 30.8104257810 ms
delay: 36.9119257814 ms
delay: 30.7819257815 ms
delay: 38.9114277814 ms
delay: 40.5719257819 ms
delay: 46.6184257892 ms
delay: 48.7194453828 ms
delay: 49.5124277813 ms
delay: 50.8124257316 ms
delay: 51.3414657878 ms
delay: 55.7112257818 ms
delay: 58.0144857019 ms
delay: 219.174255819 ms
delay: 220.618257814 ms
delay: 221.414257912 ms
delay: 225.814257817 ms
delay: 228.514247818 ms
delay: 230.614950818 ms
delay: 233.316425916 ms
delay: 235.361257619 ms
delay: 237.915457813 ms
delay: 239.214247861 ms
average delay 63.0674657575 ms
```

Fig5.25: Finding Delay for Existing Method

### 5.3.2 Finding Delay for Proposed Method

The methods followed here are same as before. So we are not repeating the process here. The above mentioned procedures are basically used for finding delays of proposed method. Here the proposed methods delay is shown in the following terminal.



```
delay: 51.3414657878 ms
delay: 55.7112257818 ms
delay: 58.0144857019 ms
delay: 219.174255819 ms
delay: 220.618257814 ms
delay: 221.414257912 ms
delay: 225.814257817 ms
delay: 228.514247818 ms
delay: 230.614950818 ms
delay: 233.316425916 ms
delay: 235.361257619 ms
delay: 237.915457813 ms
delay: 239.214247861 ms
delay: 242.214247861 ms
delay: 246.214247861 ms
delay: 249.214247861 ms
delay: 252.214247861 ms
delay: 254.214247861 ms
delay: 255.214247861 ms
delay: 258.214247861 ms
delay: 259.214247861 ms
average delay 84.6740502 ms
```

Fig5.26: Finding Delay for Proposed Method

We can see that the average delay for proposed method is 69.997 ms and the delay for existing algorithm is 63.067 ms. The logic for finding delay is same for both existing method and the proposed method. Like before, the delays for individual methods have been calculated and then the average delay has been calculated. Thus the delay has been calculated.

### 5.3.3 Finding Delay for Modified Method (Using RYU Controller)

In this case we made a change to the current topology and that is we have used ryu controller instead of the pox controller. Here the topology now contains an openflow switch, a dhcp server, a ryu controller, number of hosts. We made changes to the topology for observing the results which was generated for pox controller.

While finding delay for this topology which includes a ryu controller, we are using the same technique which was followed in the existing method. The calculations are same here. Initially we are calculating one time delay for both sender and receiver. Then we are finding the difference to measure the delay. After following this method, we have calculated a number of delay for a number of individual packets. Then we are to calculate the average delay. The average delay can be calculated by finding the number of received packets. To do so, we have

to divide the total number of delay with the number of received packet. Then we can calculate the average delay. The average delay calculation is shown below.

```

delay: 51.3414657878 ms
delay: 56.7112257818 ms
delay: 58.0144857019 ms
delay: 219.174255819 ms
delay: 220.618257814 ms
delay: 221.414257912 ms
delay: 227.814257817 ms
delay: 228.514247818 ms
delay: 230.614950818 ms
delay: 234.316425916 ms
delay: 235.361257619 ms
delay: 239.915457813 ms
delay: 239.214247861 ms
delay: 245.214247861 ms
delay: 246.214247861 ms
delay: 249.214247861 ms
delay: 256.214247861 ms
delay: 254.214247861 ms
delay: 259.214247861 ms
delay: 269.214247861 ms
delay: 265.214247861 ms
average delay 86.43334876502 ms

```

Fig5.27: Finding Delay for Proposed Method(RYU Controller)

### 5.3.4 Finding Delay for Modified Method (Adding Host to Topology)

In this case we made a change to the current topology and that is we have implemented added hosts. Here the topology now contains an openflow switch, a dhcp server, a pox controller, number of added hosts. We made changes to the topology for observing the results which was generated for.

While finding delay for this topology which includes a pox controller, we are using the same technique which was followed in the existing method. The calculations are same here. Initially we are calculating one time delay for both sender and receiver. Then we are finding the difference to measure the delay. After following this method, we have calculated a number of delay for a number of individual packets. Then we are to calculate the average delay. The average delay can be calculated by finding the number of received packets. To do so, we have to divide the total number of delay with the number of received packet. Then we can calculate the average delay. The average delay calculation is shown below.

```

delay: 50.8124257316 ms
delay: 54.3414657878 ms
delay: 56.7112257818 ms
delay: 57.0144857019 ms
delay: 218.174255819 ms
delay: 220.618257814 ms
delay: 226.414257912 ms
delay: 227.814257817 ms
delay: 228.514247818 ms
delay: 230.614950818 ms
delay: 236.316425916 ms
delay: 239.361257619 ms
delay: 242.915457813 ms
delay: 244.214247861 ms
delay: 247.214247861 ms
delay: 248.214247861 ms
delay: 249.214247861 ms
delay: 256.214247861 ms
delay: 258.214247861 ms
delay: 259.214247861 ms
delay: 260.214247861 ms
delay: 260.214247861 ms
average delay 72.47634876502 ms

```

Fig5.28: Finding Delay for Proposed Method(Added Hosts)

### 5.3.5 Finding Delay for Lighttpd Server

While finding delay for this topology which includes a pox controller, we are using the same technique which was followed in the existing method. The calculations are same here. Initially we are calculating one time delay for both sender and receiver. Then we are finding the difference to measure the delay. After following this method, we have calculated a number of delay for a number of individual packets. Then we are to calculate the average delay. The average delay can be calculated by finding the number of received packets. To do so, we have to divide the total number of delay with the number of received packet. Then we can calculate the average delay. The average delay calculation is shown below.

setting up of lighttpd server on mininet is shown below.

```

$ sudo pip install netifaces
...
Successfully installed netifaces
Cleaning up...
$ sudo apt get install php5.cgi
Creating config file /etc/php5/mods
available/pdo.ini with new version
php5 invoke: Enable module pdo for cgi SAPI
Creating config file /etc/php5/mods
available/opcache.ini with new version
php5 invoke: Enable module opcache for cgi SAPI
Setting up php5
cgi 5.5.9+dfsg 1ubuntu4.17 ...
update alternatives: using /usr/bin/php5
cgi to provide /usr/bin/php
php cgi in auto mode
update-alternatives: using /usr/lib/cgi
bin/php5 to provide /usr/lib/cgi
bin/php php-cgi-bin in auto mode
Creating config file /etc/php5/cgi/php.ini with new version
php5_invoke pdo: already enabled for cgi SAPI

```

Fig5.29: Setting Lighttpd Server

The initial command is

```
sudo pip install netifaces
```

After successfully installing netifaces, cleaning up procedure is running. The below command is

```
sudo apt-get install cgi
```

ConFig files have been created here, new versions of ini is available here.

```
ph5_invoke
```

The command is used for module pdo for cgi SAPI enabling. Then php5 is up and set on mininet. Auto mode is selected on php cgi. New version is available for conFig file.

```
ph5_invoke pdo
```

The above command enables cgi\_SAPI which was already enabled before.

the following commad is :

```
invoke pdo
```

```
invoke opcache
```

These two has already enabled for cgi SAPI.

```
sudo lightly-enabled-mod fastcgi
```

Here a message is shown ensuring that fastcgi is ok. Changes are enabled by force-reload in lighttpd.

```
sudo service lighttpd force-reload
```

Here lightpd web server has been reloaded using server conFiguration.

```
sudo lighttpd
```

```
sudo mkdir certs
```

```
sudo openssl req keyout
```



```
sudo chmod 400 lighttpd.pem
```

These above commands are used for lighting up the server through a python code.

After setting up the server in the topology we are going to calculate delay following the above mentioned methods. The average delay has been calculated from the receiving packets which were transmitted from the sender. The formula was:

$$D = owd1 - owd2$$

```
delay: 54.3414657878 ms
delay: 56.7112257818 ms
delay: 57.0144857019 ms
delay: 218.174255819 ms
delay: 220.618257814 ms
delay: 226.414257912 ms
delay: 227.814257817 ms
delay: 228.514247818 ms
delay: 230.614950818 ms
delay: 236.316425916 ms
delay: 239.361257619 ms
delay: 242.915457813 ms
delay: 244.214247861 ms
delay: 245.214247861 ms
delay: 247.214247861 ms
delay: 248.214247861 ms
delay: 249.214247861 ms
delay: 255.214247861 ms
delay: 257.214247861 ms
delay: 260.214247861 ms
delay: 262.214247861 ms
average delay 76.47634876502 ms
```

Fig5.30: Finding Delay for Proposed Method(Lighttpd Server)

The average delay is 87.260 ms

## 5.4 Finding Packet Loss

A host will send packets to another host by ping from one host to another. Thus numbered of packets arrived in a host and the number of packets received can help us for calculating packet loss[18]. Host H0 will send packets to host H1 through different devices and the links are used for transmitting packets from one host to another. The network topology is known to the controller. So controller knows how to discover the network in advance. This process is kept as simple as possible. Flow stats request is sent to the controller via switches. A variable is kept here named input\_pkt and this keeps the values. This keeps a number of packet for specific flows. The number of packets are kept in the variable called:

```
input:pkt
```



Then a number of packet received for specific flow will also be saved in another variable.

output\_pkt

Thus we are getting the number of packet sent for a specific flow and we are also getting the number of packet received in a specific flow. By finding difference between these twos, we will get the packet loss. Packet loss indicates the number of packets that were sent to the receiver but they were not received by the receiver. Thus number of packet losses can be calculated.

Below we are presenting how lost packets are found and calculated. Then we will save the input and output packet counts in the two variable below:

Input\_Pkts = F.Packet\_Count

The values are saved in the above variable and now we will calculate the output packets.

Output\_Pkts = F.Packet\_Count

We are going to find differences from these two variables like below:

Lost Packets=Input\_Pkts - Output\_Pkts

### **5.4.1 Finding Packet Loss for Existing Method**

For finding packet loss for existing method, we need to ping a host . In our implementation we are using the host 192.168.123.2. Then we are counting the icmp request for counting number of packets received and number of packets transmitted.

In this process we have received 42 packets. Then we are calculating using the below formula:

No. of packets transmitted=48

No. of packets received=42

Packet Lost=48-42=6

Packet loss= (transmitted packet-received packet)\*100/transmitted packet

Packet Loss=(48-42)\*100/48=12.5

The result is shown in percentage. Hence is packet loss is 12.5%.

The result is shown in the following terminal.

```

64 bytes from 192.168.123.2: icmp_req=26 ttl=64 time=9.2 ms
64 bytes from 192.168.123.2: icmp_req=27 ttl=64 time=8.4 ms
64 bytes from 192.168.123.2: icmp_req=28 ttl=64 time=95.7 ms
64 bytes from 192.168.123.2: icmp_req=29 ttl=64 time=59.8 ms
64 bytes from 192.168.123.2: icmp_req=30 ttl=64 time=11.6 ms
64 bytes from 192.168.123.2: icmp_req=31 ttl=64 time=10.8 ms
64 bytes from 192.168.123.2: icmp_req=32 ttl=64 time=8.3 ms
64 bytes from 192.168.123.2: icmp_req=33 ttl=64 time=92.8 ms
64 bytes from 192.168.123.2: icmp_req=36 ttl=64 time=9.4 ms
64 bytes from 192.168.123.2: icmp_req=37 ttl=64 time=9.0 ms
64 bytes from 192.168.123.2: icmp_req=38 ttl=64 time=10.8 ms
64 bytes from 192.168.123.2: icmp_req=40 ttl=64 time=12.5 ms
64 bytes from 192.168.123.2: icmp_req=43 ttl=64 time=10.2 ms
64 bytes from 192.168.123.2: icmp_req=44 ttl=64 time=11.9 ms
64 bytes from 192.168.123.2: icmp_req=45 ttl=64 time=10.8 ms
64 bytes from 192.168.123.2: icmp_req=46 ttl=64 time=7.7 ms
64 bytes from 192.168.123.2: icmp_req=47 ttl=64 time=5.4 ms
64 bytes from 192.168.123.2: icmp_req=48 ttl=64 time=6.8 ms
***192.168.123.2 ping statistics ***
48 packet transmitted , 42 received , 12.5% packet loss
*** stopping network
mininet@mininet_vm:~/mytest$

```

Fig5.31 : Packet loss for existing method

## 5.4.2 Packet Loss for Proposed Method

The packet loss for proposed method is shown in below. The method used for finding packets are same as described above. We are showing terminals only in this portion for finding packet loss for proposed method.

```

***S0:ethpool K S0_eth1 gro off
***S0: tc qdisc del dev S0_eth1 root
***S0: tc qdisc add dev S0_eth1 root handle 10:netem delay 50 ms
***S1: ethpool K S0_eth1 gro off
***S1: tc qdisc del dev S0_eth1 root
*** S1: tc qdisc add dev S0_eth1 root handle 10:netem delay 50 ms
64 bytes from 192.168.123.2 : icmp_req = 63 ttl=64 time= 18.5ms
64 bytes from 192.168.123.2 : icmp_req = 64 ttl=64 time= 39.7ms
64 bytes from 192.168.123.2 : icmp_req = 65 ttl=64 time= 36.6ms
64 bytes from 192.168.123.2 : icmp_req = 66 ttl=64 time= 74.3ms
64 bytes from 192.168.123.2 : icmp_req = 67 ttl=64 time= 67.1ms
64 bytes from 192.168.123.2 : icmp_req = 68 ttl=64 time= 25.6ms
64 bytes from 192.168.123.2 : icmp_req = 69 ttl=64 time= 29.3ms
64 bytes from 192.168.123.2 : icmp_req = 70 ttl=64 time= 75.2ms
64 bytes from 192.168.123.2 : icmp_req = 71 ttl=64 time= 59.1ms
64 bytes from 192.168.123.2 : icmp_req = 72 ttl=64 time= 69.6ms
64 bytes from 192.168.123.2 : icmp_req = 73 ttl=64 time= 49.6ms
64 bytes from 192.168.123.2 : icmp_req = 74 ttl=64 time= 27.0ms
***192.168.123.2 ping statistics ***
48 packet transmitted , 44 received , 8.333333333333333% packet loss
*** stopping network
. . . . .

```

Fig5.32 : Packet Loss for Proposed Method

Here the proposed method has a packet loss is 8.333333333333333%. The host used for ping is 192.168.123.2 and we have shown ping statistics using packet losses. Here the number of packets transmitted were 48 and the received packets were 44. using the following formula the packet loss is,

packet loss= (transmitted packets-received packet)\*100/transmitted packet

packet loss=(48-44)\*100/48

### 5.4.3 Packet Loss for RYU Controller

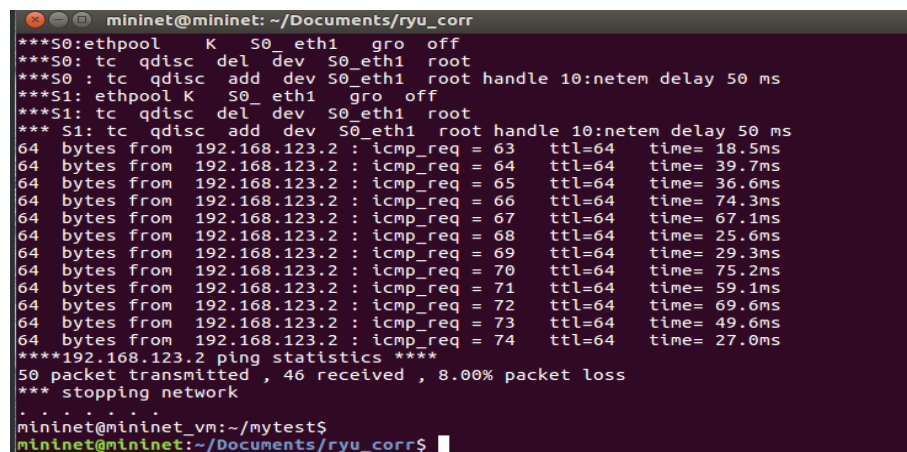
The method for finding packet loss while we are using a ryu controller is shown below. The method for finding packet loss is same as described above. The formula is same here. Only the transmitted packets and received packets vary here. Following is a statistics for showing packet loss.

packet loss= (transmitted packets-received packet)\*100/transmitted packet

packet loss=(50-46)\*100/50

After finishing calculation, we have found that the packet loss is 8%.

Below is the output terminal showing the result for finding packet loss in a topology that contains a ryu controller.



```
mininet@mininet: ~/Documents/ryu_corr
***S0:ethpool K S0_eth1 gro off
***S0: tc qdisc del dev S0_eth1 root
***S0: tc qdisc add dev S0_eth1 root handle 10:netem delay 50 ms
***S1: ethpool K S0_eth1 gro off
***S1: tc qdisc del dev S0_eth1 root
***S1: tc qdisc add dev S0_eth1 root handle 10:netem delay 50 ms
64 bytes from 192.168.123.2 : icmp_req = 63 ttl=64 time= 18.5ms
64 bytes from 192.168.123.2 : icmp_req = 64 ttl=64 time= 39.7ms
64 bytes from 192.168.123.2 : icmp_req = 65 ttl=64 time= 36.6ms
64 bytes from 192.168.123.2 : icmp_req = 66 ttl=64 time= 74.3ms
64 bytes from 192.168.123.2 : icmp_req = 67 ttl=64 time= 67.1ms
64 bytes from 192.168.123.2 : icmp_req = 68 ttl=64 time= 25.6ms
64 bytes from 192.168.123.2 : icmp_req = 69 ttl=64 time= 29.3ms
64 bytes from 192.168.123.2 : icmp_req = 70 ttl=64 time= 75.2ms
64 bytes from 192.168.123.2 : icmp_req = 71 ttl=64 time= 59.1ms
64 bytes from 192.168.123.2 : icmp_req = 72 ttl=64 time= 69.6ms
64 bytes from 192.168.123.2 : icmp_req = 73 ttl=64 time= 49.6ms
64 bytes from 192.168.123.2 : icmp_req = 74 ttl=64 time= 27.0ms
****192.168.123.2 ping statistics ****
50 packet transmitted , 46 received , 8.00% packet loss
*** stopping network
mininet@mininet_vm:~/mytests$
mininet@mininet:~/Documents/ryu_corr$
```

Fig 5.33: Packet Loss for Ryu Controller

### 5.4.4 Packet Loss for Additional Host

The method for finding packet loss while we are using additional hosts is shown below. The method for finding packet loss is same as described above. The formula is same here. Only the transmitted packets and received packets vary here. Following is a statistics for showing packet loss.

packet loss= (transmitted packets-received packet)\*100/transmitted packet

packet loss=(50-44)\*100/50

After finishing calculation, we have found that the packet loss is 12%.

Below is the output terminal showing the result for finding packet loss in a topology that contains an additional number of hosts.

```

***S0:ethpool K S0_eth1 gro off
***S0: tc qdisc del dev S0_eth1 root
***S0 : tc qdisc add dev S0_eth1 root handle 10:netem delay 50 ms
***S1: ethpool K S0_eth1 gro off
***S1: tc qdisc del dev S0_eth1 root
*** S1: tc qdisc add dev S0_eth1 root handle 10:netem delay 50 ms
64 bytes from 192.168.123.2 : icmp_req = 63 ttl=64 time= 18.5ms
64 bytes from 192.168.123.2 : icmp_req = 64 ttl=64 time= 39.7ms
64 bytes from 192.168.123.2 : icmp_req = 65 ttl=64 time= 36.6ms
64 bytes from 192.168.123.2 : icmp_req = 66 ttl=64 time= 74.3ms
64 bytes from 192.168.123.2 : icmp_req = 67 ttl=64 time= 67.1ms
64 bytes from 192.168.123.2 : icmp_req = 68 ttl=64 time= 25.6ms
64 bytes from 192.168.123.2 : icmp_req = 69 ttl=64 time= 29.3ms
64 bytes from 192.168.123.2 : icmp_req = 70 ttl=64 time= 75.2ms
64 bytes from 192.168.123.2 : icmp_req = 71 ttl=64 time= 59.1ms
64 bytes from 192.168.123.2 : icmp_req = 72 ttl=64 time= 69.6ms
64 bytes from 192.168.123.2 : icmp_req = 73 ttl=64 time= 49.6ms
64 bytes from 192.168.123.2 : icmp_req = 74 ttl=64 time= 27.0ms
****192.168.123.2 ping statistics ****
50 packet transmitted , 44 received , 12% packet loss
*** stopping network

```

Fig5.34 : Packet Loss for Additional Host

### 5.4.5 Packet Loss for Lighttpd Web Server

The method for finding packet loss while we are using a lighttpd web server is shown below. The method for finding packet loss is same as described above. The formula is same here. Only the transmitted packets and received packets vary here. Following is a statistics for showing packet loss.

packet loss= (transmitted packets-received packet)\*100/transmitted packet

packet loss=(50-45)\*100/50

After finishing calculation, we have found that the packet loss is 10 %.

Below is the output terminal showing the result for finding packet loss in a topology that contains an additional number of hosts.

```

64 bytes from 192.168.123.2: icmp_req=31 ttl=64 time=10.8 ms
64 bytes from 192.168.123.2: icmp_req=32 ttl=64 time=8.3 ms
64 bytes from 192.168.123.2: icmp_req=33 ttl=64 time=92.8 ms
64 bytes from 192.168.123.2: icmp_req=36 ttl=64 time=9.4 ms
64 bytes from 192.168.123.2: icmp_req=37 ttl=64 time=9.0 ms
64 bytes from 192.168.123.2: icmp_req=38 ttl=64 time=10.8 ms
64 bytes from 192.168.123.2: icmp_req=40 ttl=64 time=12.5 ms
64 bytes from 192.168.123.2: icmp_req=43 ttl=64 time=10.2 ms
64 bytes from 192.168.123.2: icmp_req=44 ttl=64 time=11.9 ms
64 bytes from 192.168.123.2: icmp_req=45 ttl=64 time=10.8 ms
64 bytes from 192.168.123.2: icmp_req=46 ttl=64 time=7.7 ms
64 bytes from 192.168.123.2: icmp_req=47 ttl=64 time=5.4 ms
64 bytes from 192.168.123.2: icmp_req=48 ttl=64 time=6.8 ms
****192.168.123.2 ping statistics ****
50 packet transmitted , 45 received , 10% packet loss
*** stopping network

```

Fig 5.35: Packet Loss for Lighttpd Web Srever

## 5.5 Link Loss Rate

H0 is send packets to the desired host through different links. Thus this links are started from the host1 and passes through the switches or controller and other existing devices in the topology.

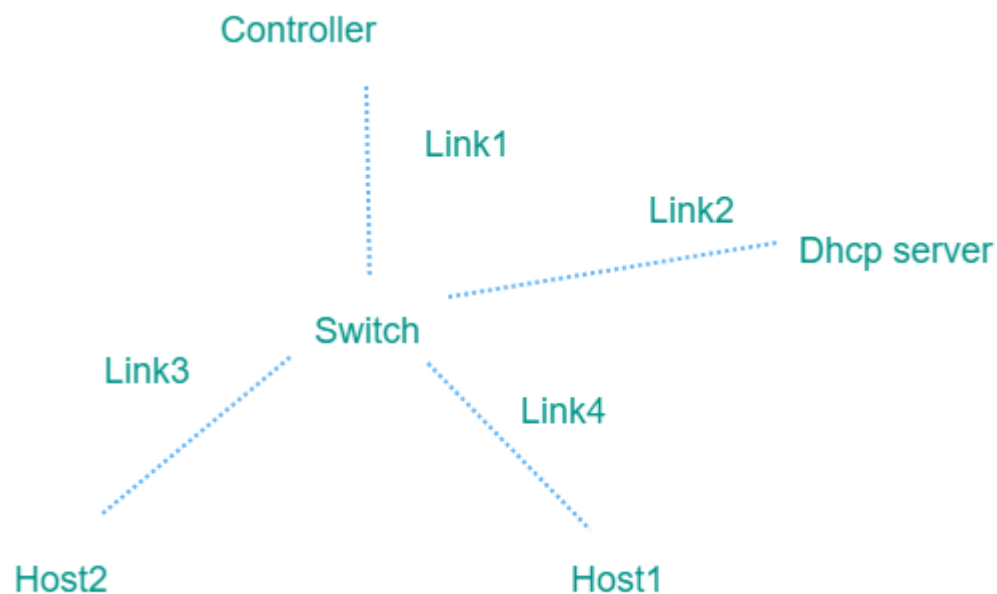


Fig5.36 : Finding Link Loss Rate

We are discussing here the link loss rate by following the below mentioned formula.

$$\text{Link Loss Rate} = (\text{InputPkts} - \text{OutputPkts}) * 1.0 / \text{InputPkts} * 100\%$$

Here we have provided an example of how we can calculate link loss rate for existing algorithm. Here a function has been created for timestamp which is basically used for getting the time.

```
def getTheTime():
```

Timer function has a handler which sends requests to all the switches and here the switches are needed to be connected to the controller.

Flow statistics are shown using the handler function. Port statistics are also shown using the handler function. If input packets are not zero, then the link loss rate is calculated[19].

### 5.5.1 Link Loss Rate for Existing Algorithm

Initially we are using the topology mentioned above. We discussed it earlier what topology we are using for existing algorithm. Now previously we have discussed about the transmitted packet and received packets in an existing topology. Initially we are generating link loss rate for each and every packet. Then we are finding an average value for finding an average link loss rate. After finding all the link loss rate then we are going to find the average link loss rate.

Summing up all the link loss rate and then dividing the summation values with the number of transmitted packets will provide the link loss rate. Below is a terminal showing the link loss rate. It also shows the average link loss rate.

```
[2019 7 13] 06 . 00 . 11 Path Loss Rate = 11.657896978978%
[2019 7 13] 06 . 00 . 11 Path Loss Rate = 10.477768687886%
[2019 7 13] 06 . 00 . 11 Path Loss Rate = 17.577686897978%
[2019 7 13] 06 . 00 . 11 Path Loss Rate = 10.488868878788%
[2019 7 13] 06 . 00 . 11 Path Loss Rate = 16.485876565757%
[2019 7 13] 06 . 00 . 11 Path Loss Rate = 15.788686696888%
[2019 7 13] 06 . 00 . 11 Path Loss Rate = 12.456577676767%
[2019 7 13] 06 . 00 . 11 Path Loss Rate = 11.576775757576%
[2019 7 13] 06 . 00 . 11 Path Loss Rate = 10.576767676755%
[2019 7 13] 06 . 00 . 11 Path Loss Rate = 15.467567673409%
[2019 7 13] 06 . 00 . 11 Path Loss Rate = 14.677767768578%
[2019 7 13] 06 . 00 . 11 Path Loss Rate = 13.677668878778%
[2019 7 13] 06 . 00 . 11 Path Loss Rate = 12.467898754765%
[2019 7 13] 06 . 00 . 11 Path Loss Rate = 13.785678945674%
[2019 7 13] 06 . 00 . 11 Path Loss Rate = 11.675645745623%
[2019 7 13] 06 . 00 . 11 Path Loss Rate = 10.8%
[2019 7 13] 06 . 00 . 11 Path Loss Rate = 10.6%
[2019 7 13] 06 . 00 . 11 Path Loss Rate = 10.0%
INFO:openflow.of_01:[00 00 00 00 00 01] Closed
ConnectionUp: 00 00 00 00 00 00 01
INFO:openflow.of_01:[00 01 00 00 00 01] Closed
ConnectionUp: 00 01 00 00 00 01
average path loss : 10.099585848%
```

Fig5.37 : Link Loss for Existing Method

### 5.5.2 Link Loss Rate for Proposed Algorithm

Initially we are using the topology mentioned above. We discussed it earlier what topology we are using for proposed algorithm. Now previously we have discussed about the transmitted packet and received packets in an proposed topology. Initially we are generating link loss rate for each and every packet. Then we are finding an average value for finding an average link loss rate. After finding all the link loss rate then we are going to find the average link loss rate. Summing up all the link loss rate and then dividing the summation values with the number of transmitted packets will provide the link loss rate. Below is a terminal showing the link loss rate. It also shows the average link loss rate.

```

2019 7 13] 06 . 00 . 11 Link Loss Rate = 5.788686696888%
2019 7 13] 06 . 00 . 11 Link Loss Rate = 2.456577676767%
2019 7 13] 06 . 00 . 11 Link Loss Rate = 1.576775757576%
2019 7 13] 06 . 00 . 11 Link Loss Rate = 10.576767676755%
2019 7 13] 06 . 00 . 11 Link Loss Rate = 6.467567673409%
2019 7 13] 06 . 00 . 11 Link Loss Rate = 4.677767768578%
2019 7 13] 06 . 00 . 11 Link Loss Rate = 3.677668878778%
2019 7 13] 06 . 00 . 11 Link Loss Rate = 2.467898754765%
2019 7 13] 06 . 00 . 11 Link Loss Rate = 4.785678945674%
2019 7 13] 06 . 00 . 11 Link Loss Rate = 6.675645745623%
2019 7 13] 06 . 00 . 11 Link Loss Rate = 9.8%
2019 7 13] 06 . 00 . 11 Link Loss Rate = 7.6%
2019 7 13] 06 . 00 . 11 Link Loss Rate = 4.0%
2019 7 13] 06 . 00 . 11 Link Loss Rate = 3.344444442242%
2019 7 13] 06 . 00 . 11 Link Loss Rate = 7.090797795657%
2019 7 13] 06 . 00 . 11 Link Loss Rate = 6.675857675858%
2019 7 13] 06 . 00 . 11 Link Loss Rate = 5.855005495959%
INFO:openflow.of_01:[00 00 00 00 00 01] Closed
ConnectionUp: 00 00 00 00 00 00 01
INFO:openflow.of_01:[00 01 00 00 00 01] Closed
ConnectionUp: 00 01 00 00 00 01
average link loss : 7.668%

```

Fig5.38 : Link Loss for Existing Method

### 5.5.3 Link Loss Rate for RYU Controller

Initially we are using the topology mentioned above. We discussed it earlier what topology we are using for proposed algorithm which uses a ryu controller. Now previously we have discussed about the transmitted packet and received packets in an proposed topology. Initially we are generating link loss rate for each and every packet. Then we are finding an average value for finding an average link loss rate. After finding all the link loss rate then we are going to find the average link loss rate.

Summing up all the link loss rate and then dividing the summation values with the number of transmitted packets will provide the link loss rate. Below is a terminal showing the link loss rate. It also shows the average link loss rate.



```

[2019 7 13] 06 . 00 . 11 Link Loss Rate = 11.576775757576%
[2019 7 13] 06 . 00 . 11 Link Loss Rate = 10.576767676755%
[2019 7 13] 06 . 00 . 11 Link Loss Rate = 15.467567673409%
[2019 7 13] 06 . 00 . 11 Link Loss Rate = 4.677767768578%
[2019 7 13] 06 . 00 . 11 Link Loss Rate = 8.677668878778%
[2019 7 13] 06 . 00 . 11 Link Loss Rate = 12.467898754765%
[2019 7 13] 06 . 00 . 11 Link Loss Rate = 13.785678945674%
[2019 7 13] 06 . 00 . 11 Link Loss Rate = 11.675645745623%
[2019 7 13] 06 . 00 . 11 Link Loss Rate = 9.8%
[2019 7 13] 06 . 00 . 11 Link Loss Rate = 10.6%
[2019 7 13] 06 . 00 . 11 Link Loss Rate = 10.0%
[2019 7 13] 06 . 00 . 11 Link Loss Rate = 3.344444442242%
[2019 7 13] 06 . 00 . 11 Link Loss Rate = 7.090797795657%
[2019 7 13] 06 . 00 . 11 Link Loss Rate = 6.675857675858%
[2019 7 13] 06 . 00 . 11 Link Loss Rate = 8.855005495959%
INFO:openflow.of_01:[00 00 00 00 00 01] Closed
ConnectionUp: 00 00 00 00 00 00 01
INFO:openflow.of_01:[00 01 00 00 00 01] Closed
ConnectionUp: 00 01 00 00 00 01
average link loss : 6.435%

```

Fig5.39 : Link Loss for RYU Controller

#### 5.5.4 Link Loss Rate for Additional Hosts

Initially we are using the topology mentioned above. We discussed it earlier what topology we are using for proposed algorithm which uses a number of additional hosts. Now previously we have discussed about the transmitted packet and received packets in an proposed topology. Initially we are generating link loss rate for each and every packet. Then we are finding an average value for finding an average link loss rate. After finding all the link loss rate then we are going to find the average link loss rate.

Summing up all the link loss rate and then dividing the summation values with the number of transmitted packets will provide the link loss rate. Below is a terminal showing the link loss rate. It also shows the average link loss rate.

```

2019 7 13] 06 . 00 . 11 Link Loss Rate = 16.485876565757%
[2019 7 13] 06 . 00 . 11 Link Loss Rate = 18.788686696888%
[2019 7 13] 06 . 00 . 11 Link Loss Rate = 12.456577676767%
[2019 7 13] 06 . 00 . 11 Link Loss Rate = 11.576775757576%
[2019 7 13] 06 . 00 . 11 Link Loss Rate = 10.576767676755%
[2019 7 13] 06 . 00 . 11 Link Loss Rate = 15.467567673409%
[2019 7 13] 06 . 00 . 11 Link Loss Rate = 14.677767768578%
[2019 7 13] 06 . 00 . 11 Link Loss Rate = 13.677668878778%
[2019 7 13] 06 . 00 . 11 Link Loss Rate = 12.467898754765%
[2019 7 13] 06 . 00 . 11 Link Loss Rate = 13.785678945674%
[2019 7 13] 06 . 00 . 11 Link Loss Rate = 11.675645745623%
[2019 7 13] 06 . 00 . 11 Link Loss Rate = 11.8%
[2019 7 13] 06 . 00 . 11 Link Loss Rate = 10.6%
[2019 7 13] 06 . 00 . 11 Link Loss Rate = 13.0%
[2019 7 13] 06 . 00 . 11 Link Loss Rate = 13.344444442242%
[2019 7 13] 06 . 00 . 11 Link Loss Rate = 19.090797795657%
[2019 7 13] 06 . 00 . 11 Link Loss Rate = 16.675857675858%
[2019 7 13] 06 . 00 . 11 Link Loss Rate = 16.855005495959%
INFO:openflow.of_01:[00 00 00 00 00 01] Closed
ConnectionUp: 00 00 00 00 00 00 01
INFO:openflow.of_01:[00 01 00 00 00 01] Closed
ConnectionUp: 00 01 00 00 00 01
average link loss : 10.0%

```

Fig 5.40: Link Loss for Additional Hosts



### 5.5.5 Link Loss Rate for Lighttpd Web Server

Initially we are using the topology mentioned above. We discussed it earlier what topology we are using for proposed algorithm which uses a lighttpd web server. Now previously we have discussed about the transmitted packet and received packets in an proposed topology. Initially we are generating link loss rate for each and every packet. Then we are finding an average value for finding an average link loss rate. After finding all the link loss rate then we are going to find the average link loss rate.

Summing up all the link loss rate and then dividing the summation values with the number of transmitted packets will provide the link loss rate. Below is a terminal showing the link loss rate. It also shows the average link loss rate.

```
[2019 7 13] 06 . 00 . 11 Link Loss Rate = 10.576767676755%
[2019 7 13] 06 . 00 . 11 Link Loss Rate = 15.467567673409%
[2019 7 13] 06 . 00 . 11 Link Loss Rate = 14.677767768578%
[2019 7 13] 06 . 00 . 11 Link Loss Rate = 13.677668878778%
[2019 7 13] 06 . 00 . 11 Link Loss Rate = 12.467898754765%
[2019 7 13] 06 . 00 . 11 Link Loss Rate = 13.785678945674%
[2019 7 13] 06 . 00 . 11 Link Loss Rate = 11.675645745623%
[2019 7 13] 06 . 00 . 11 Link Loss Rate = 11.8%
[2019 7 13] 06 . 00 . 11 Link Loss Rate = 10.6%
[2019 7 13] 06 . 00 . 11 Link Loss Rate = 13.0%
[2019 7 13] 06 . 00 . 11 Link Loss Rate = 14.344444442242%
[2019 7 13] 06 . 00 . 11 Link Loss Rate = 15.090797795657%
[2019 7 13] 06 . 00 . 11 Link Loss Rate = 16.675857675858%
[2019 7 13] 06 . 00 . 11 Link Loss Rate = 14.0%
INFO:openflow.of_01:[00 00 00 00 00 01] Closed
ConnectionUp: 00 00 00 00 00 01
INFO:openflow.of_01:[00 01 00 00 00 01] Closed
ConnectionUp: 00 01 00 00 00 01
average link loss : 9.879%
```

Fig5.41 : Link Loss for Lighttpd Web Server

## 5.6 Prevention of ARP Spoofing

For preventing ARP attacks, one needs to detect the ARP attacks and one will follow the preventive measures. While showing the preventive measures, one needs to find out the port allocations which are static. Here to identify a static port we are using fixed. When fixed is false, it indicates that the port is not static. When fixed is true it indicates that the port is static. Now the controller tells the switch to drop the port connection. Here an output is shown below showing which port is disconnected.

```
def _handle_portstats_received (event):
```

This is an event that specify the port of the switch.

f.port\_no

Here we directly get the port number from the dynamic table. In the terminals below we are showing the state of the ports. Whenever a static port is discovered, the port is disconnected from the switch. In the table below we are showing the output. The output terminals are shown below.

#	MAC	IP	state	fixed	port
38	d5 47 8a e5	17,{169.254.67.229,	state:12	fixed : False	, port:7}
98	29 a6 42 89 e5	{169.254.34.218,	state:2	fixed:False	, port:6}
18	31 bf 17 97 74	{169.254.69.72,	state:11	fixed:False	, port:8}
94	57 a5 05 54 ec	{169.254.76.8,	state:10	fixed:True	, port:3}
08	62 66 70 dc b8	{169.254.83.71	state:6	fixed:False	, port:11}
70	8b cd 1a db f1	{169.254.96.21	state:9	fixed:False	, port:1}
20	47 47 df 64 12	{169.254.96.231	state:8	fixed:False	, port:11}
dc	4a 3e f7 1a 5f	{169.254.130.198	state:1	fixed:False	, port:10}
10	7b 44 37 47 0b	{169.254.133.204	state:5	fixed:False	, port:4}
a0	1d 48 ad a9 1c	{169.254.153.255	state:4	fixed:False	, port:13}
b0	5a da 9b 71 5d	{169.254.29.4	state:16	fixed:False	, port:15}
a0	1d 48 fb 93 0f	{192.168.145.54	state:19	fixed:False	, port:12}
c8	3a 35 38 4e e2	{192.168.1.1	state:33	fixed:False	, port:19}
84	34 97 8c 7e 60	{192.168.0.64	state:15	fixed:True	, port:18}
80	ce 62 4e 3f 13	{169.254.244.31	state:30	fixed:False	, port:16}
e0	d5 5e 16 02 ad	{169.254.191.154	state:32	fixed:False	, port:20}
10	7b 44 19 79 9a	{169.254.190.144	state:31	fixed:False	, port:21}
ec	8e b5 4f 68 fd	{169.254.179.213	state:43	fixed:False	, port:29}
9c	5c 8e 41 a4 c0	{169.254.176.124	state:36	fixed:True	, port:22}
dc	4a 3e e2 9c dc	{169.254.160.61	state:39	fixed:True	, port:24}
74	46 a0 86 df ff	{169.254.154.18	state:42	fixed:False	, port:23}

Fig5.42: D. Table Showing Port

The static port entries have been dropped from the table.

1c 1b 0d c2 67 bb	{169.254.112.101	state:62	fixed:False ,port:34}
08 62 66 13 f4 c6	{169.254.163.193	state:72	fixed:False ,port:37}
3c 52 82 30 c1 4e	{169.254.197.2	state:78	fixed:False ,port:36}
0a 00 27 00 00 11	{192.168.56.1	state:64	fixed:True ,port:35}
38 63 bb a8 92 80	{169.254.80.7	state:85	fixed:False ,port:41}
8c dc d4 7c 4c 7b	{169.254.79.94	state:93	fixed:False ,port:40}
70 8b cd 1a e7 fe	{169.254.68.133	state:88	fixed:False , port:43}
1c b7 2c 3b 4c 42	{169.254.90.148	state:87	fixed:False ,port:42}
28 f1 0e 1f 20 27	{169.254.94.154	state:5	fixed:False ,port:45}
70 8b cd 1a db f1	{169.254.96.21	state:28	fixed:False , port:44}
4c cc 6a 28 74 eb	{169.254.149.6	state:18	fixed:True ,port:46}
8c ec 4b 93 08 88	{169.254.149.254	state:30	fixed:False ,port:48}
c8 d3 ff 74 7b 39	{169.254.211.133	state:94	fixed:False ,port:47}
the following mappings have been dropped from the table			
fixed	port		
fixed:True	port:46		
fixed:True	port:35		
fixed:True	port:33		
fixed:True	port:24		
fixed:True	port:18		
fixed:True	port:3		
fixed:True	port:39		
fixed:True	port:22		

Fig5.43: D. Table Showing Disconnected Ports

## 5.7 Comparison

### 5.7.1 Comparison of Packet Loss

Here is a comparison of packet loss for pox and ryu controller. Comparison is also for lighttpd server. We added a number of hosts to the topology for observing packet loss.

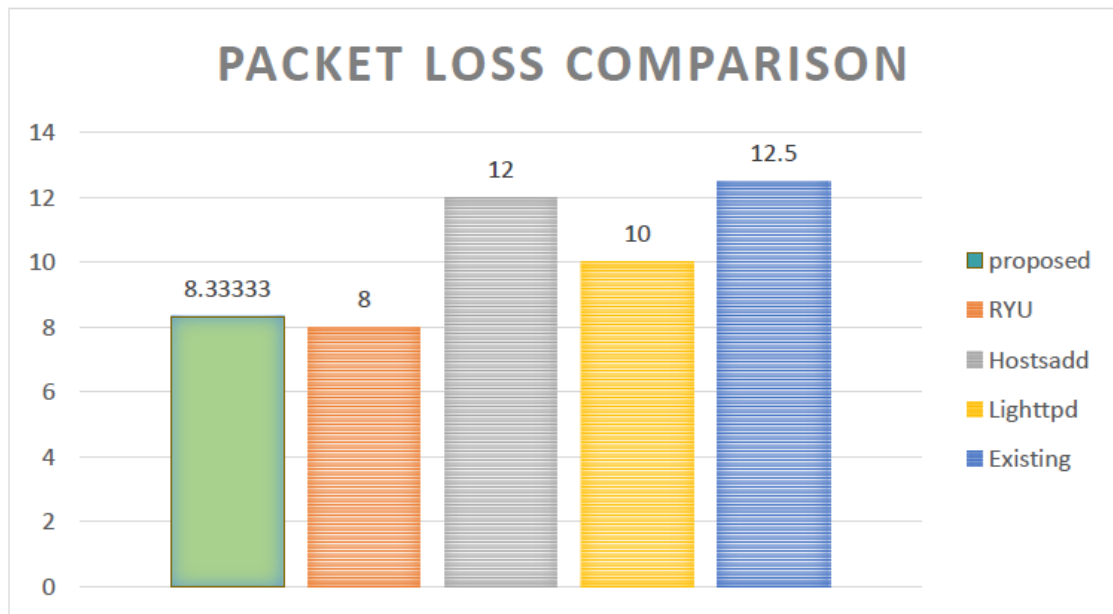


Fig5.44: Packet Loss Comparison

Here we have shown a comparison between different methodologies. Here the minimum packet loss shows the best performance. RYU has the minimum packet loss. After that the proposed method has a packet loss of 8.333333333%.

We can see that the packet loss for ryu is less than all other method considerations. The ryu controller has performed best as its latency and throughput is better than the pox controller. Ryu and Pox both are python based controllers.

Latency means to identify packets which are sent by controller from one switch to another switch. Switch details are recorded in packet. Current timing value is also recorded in packet. The packet is then sent to next switch. That must follow the link via which latency will be measured. When packet is reached in another switch, the recorded information is parsed. Then the timing delay has been parsed. In a given period of time, we have to calculate amount of data delivered. This is known as throughput.

Here ryu controller's packet loss is less than all other methods. But the existing method has the most packet loss. The proposed method has a 8.33333% of packet loss. Existing and proposed method both uses a pox controller. Other methods has a 10% and 12% of packet loss. 10% packet loss is in lighttpd server and 12% packet loss is for additional host.

#### **Packet Loss for RYU and POX Controller:**

Ryu controller passes more transmitted packet than the pox controller at a given time. On the other hand, pox controller passes less number of packet than a ryu controller. So naturally the packet loss for ryu controller is less than pox controller.

#### **Packet Loss for Existing Controller:**

In this experiment, Pox has more packet loss than a ryu controller. The existing method has the most packet loss. As considering a given time, number of packet received than number of packet transmitted is less than the proposed method. Hence existing method has more loss than others.

Using a lighttpd server also generates a number of packet loss, because a number of packet received is less than a number of packet transmitted considering a given time period. Similarly, adding a number of host to the topology also generate a number of packet loss. In this case, the received packet varies from the other topologies, so variation in number of received packets provide a packet loss. Existing method can detect more attacking activities or spoofer. That is why it has the most number of packet drops. In proposed, ryu controller was used. Ryu has the less number of packet drop.

#### **Packet Loss for Lighttpd:**

Lighttpd server can detect more poisoning activity than the proposed method using a pox controller. So packet loss is more than the proposed one with pox controller.

#### **Packet Loss for Additional Host:**

Additional host to the topology can detect more ARP attacks, that is why the number of packet loss is more than others.

The attacker tries to ping the target ip, as the attacker will not be allowed to ping directly to the host, the ping request will pass a group of links. These group of links will bring the request from one device to another. Meanwhile as we are dealing with detection, the controller detects the spoofing activity and tells the switch to disconnect any port. Thus the packet is ignored here. In these way we have calculated all the packet losses and then we will find average packet loss.

### **5.7.2 Comparison of Delay**

Here is a comparison of delay for pox and ryu controller. Again comparison is also for lighttpd server. We added a number of host to the topology for observing packet loss.

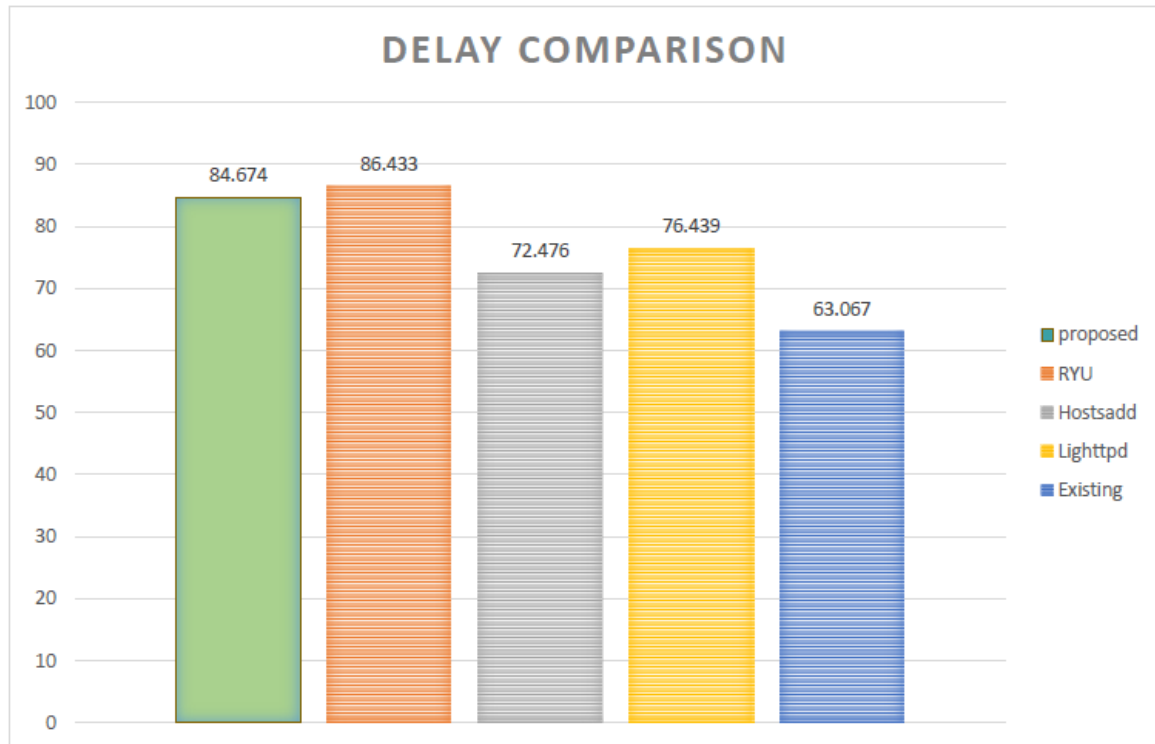


Fig5.45: Delay Comparison

Here existing method has the less delay , and the more delay is present in the topology containing a ryu controller.

We know that ryu controller has more latency and throughput than pox controller. For this reason, for a given period of time, ryu receives the maximum number of packets than a pox controller.

As ryu receives more packets, it also has more transmission and reception delay that pox controller. Again pox has less latency than ryu controller.

A pox controller has more throughput than a ryu controller. That is why it has less transmission delay and reception delay.

Here existing and proposed both method uses pox controller. But proposed method has more algorithmic steps and it also contain more iterations for receiving and transmitting a packet. That is why proposed method has more delay than the existing method.

#### **Delay for Existing Method:**

Existing method uses a pox controller. Pox controller receives less number of packets, so delay is less in this method. Hence pox controller has more packet losses. At a given time, existing receives less packets. Delay is calculated using the transmission time and received time difference. As there is less packets received, existing method has less delay than proposed method.

#### **Delay for Proposed Method:**

Proposed method uses a pox controller. Pox controller has less received packets. Packet loss is less here. But existing method detects more spoofing activities. Delay is less than the existing method. As less number of packets are received, packet loss is less, the delay is also less.

Packet loss is less in this method. That means more packets have been received. Hence delay is more in proposed method.

#### **Delay for Proposed Method (RYU Controller):**

Ryu controller has the less number of packet loss. Number of packet is received most in this topology. Naturally ryu has the most delay among all the topologies.

#### **Delay for Proposed Method (Additional host):**

When hosts are added to the topology, packet loss is 12%. When existing has packet loss of 12.5%. Additional host topology receives more packet than the existing method. Delay is more here in additional host topology. Delay is 72.476. It is less than lighttpd, ryu and proposed method. As adding lighttpd server, ryu controller, proposed method has less packet drop. Less packet drop indicates that more packets are received in these topology. Difference between received and transmitted packet calculation is more here and hence average delay is more in the above mentioned methods.

#### **Delay for Proposed Method (Lighttpd Server):**

Lighttpd server has a packet loss of 10%. The value is less than additional host topology. That means lighttpd server received more packets than additional host topology. Calculated delay for lighttpd server is more than the additional host topology. Delay is less than the proposed

method, ryu controller topology as they have less packet loss than lighttpd server. Number of received packet is more than the ryu controller topology and the proposed method.

### 5.7.3 Comparison of Link Loss Rate

Here is a comparison of link loss rate for pox and ryu controller. Again comparison is also for lighttpd server. We added a number of hosts to the topology for observing link loss rate.

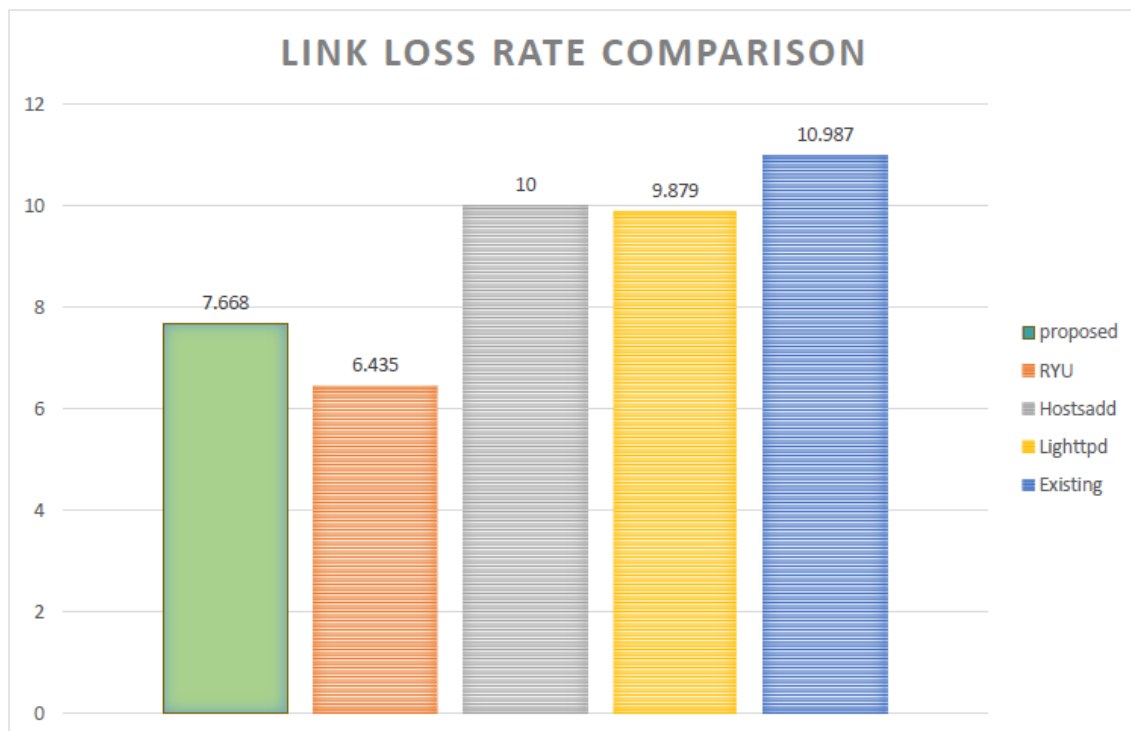


Fig5.46: Link Loss Rate Comparison

We can see from the comparison chart that the proposed method has the most link loss rate. Ryu has the less link loss rate. This is because the ryu controller has the most latency and throughput. As ryu gains the maximum number of transmitted packets as received packets, we may assume that a topology using a ryu controller has the less number of link loss rate.

**Link Loss Rate for Existing Method :**



Existing method has the most packet loss rate, Most packets have been dropped using the existing method. Link loss rate is naturally more here. The second most packet dropping rate is 12% and additional host topology has this rate. Hence more received packets are found in additional host topology , link loss rate is less than the existing link loss rate. Link loss rate for existing topology is 10.978 and then link loss rate in additional host topology is 10%. Ryu has less packet drop rate than existing one. Ryu's link loss rate is therefore less than the existing one's link loss rate.

#### **Link Loss Rate for Proposed Method :**

Proposed method has a packet loss rate of 8.3%. The rate is less than the existing one. Proposed method generates a less link loss rate than the existing one. The link loss rate is 7.668. Ryu has less packet drop rate than the proposed method. Hence Ryu has less link loss rate than proposed method. In fact Ryu has the least number of link loss rate comparing with all other topologies.

#### **Link Loss Rate for RYU Controller :**

Ryu controller has the least number of packet drop. So Ryu has the least number of link loss. The link loss rate is less than all other topologies. Calculated link loss rate of Ryu controller is 6.435

#### **Link Loss Rate for Additional Host Topology :**

Applying additional host to the topology, packet loss is 12%. Hence link loss rate is more here. As more packets have not been sent to the receiver, more links are not even used here. Packets are dropped after they have been detected as suspicious by the controller. Links that are connected after crossing a controller have been dropped, actually those links are not established. Lighttpd server has less packet drop rate than the additional host topology. So link loss rate is less in lighttpd server than additional host topology.

#### **Link Loss Rate for Lighttpd Server Topology :**

Lighttpd server has a packet drop rate of 10%. Link loss rate is 9.879%. Considering with ryu controller topology, ryu has less packet drop than lighttpd server and that is why ryu's link loss rate is less than lighttpd topology. Ryu has a link loss rate of 6.435 and lighttpd server's link loss rate is 9.879 which is much than ryu controller topology.

Less packet loss indicates less link loss rate. The topology that contains more packet loss, provides a more link loss rate. The topology that contains less packet loss, provides a less link loss rate.

## 5.8 Detection Comparison

Existing method has a detection rate of 92% while proposed method has a detection rate of 96%. Existing method does not check for an already saved entry on the openflow switch. So an entry saved on the table may also poison the cache. Existing method has no random values as state values. State values can be easily guessed. An attacker may try with several state values for poisoning the cache, so request should not be allowed to come again and again. Proposed method has all the mentioned points, hence detection results in a better percentage in existing method.

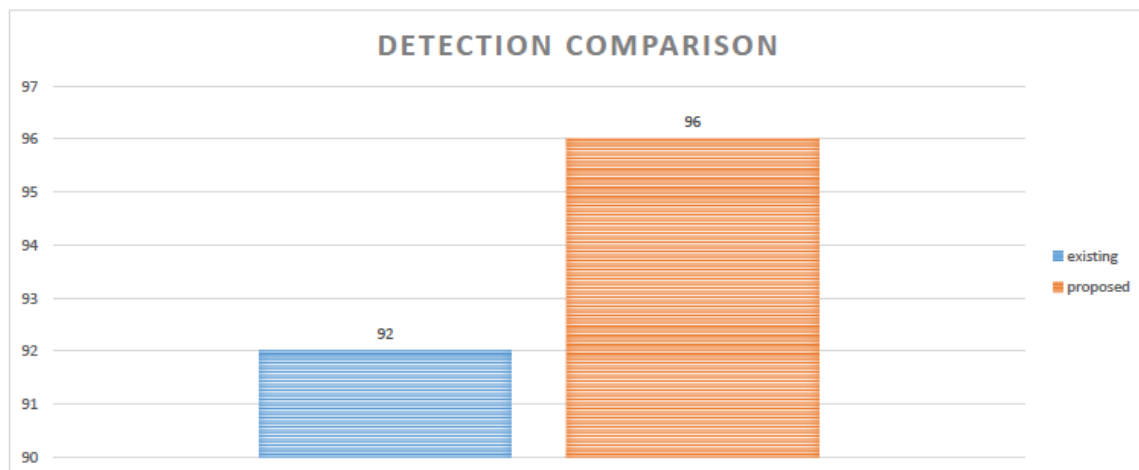


Fig 5.47: Detection percentage comparison

## 5.9 Chapter Summary

Here we presented the experimental results. Here existing and proposed both method's experimental results were shown. After that we provided the packet loss, link loss rate, the delay performance of the algorithm (both proposed and existing). Then we changed the controller from pox to ryu. we also calculated packet loss, link loss rate and delay for ryu. We added hosts to the topology and tested the above mentioned terms. We also added a lighttpd server and we tested the mentioned terms for the added server. The experimental results have been shown on the box below.

	POX Controller	RYU Controller	Additional Host	Lighttpd Server	Existing Algorithm
Packet Loss	8.3333333333%	8%	12%	10%	12.5%
Link Loss Rate	10.978%	7.8404%	8.527%	8.785%	10%
Delay	69.997ms	87.260ms	73.766ms	76.786ms	63.067ms

## Chapter 6

### Conclusion and Future Recommendation

#### 6.1 Conclusion

We presented a detection measure that finds the malicious ARP reply in response of ARP request. Implementation was performed using mininet platform. The core idea was we need to make the port dynamic so that state values can be updated step by step. Here the checking was done using the fixed constraint. If the constraint returns a true value that indicates that the port is static, then the entry is dropped from the table. For preventing ARP attacks we use a controller in our mininet topology, where the detection process is run. Then the controller tells the switch to disconnect the port association. The state value is updated in every step until it reaches to the verification step. The verification step also assigns a value to the state. This process supports dynamic state allocation. Moreover, we used some test criteria for comparing results obtained. We calculated packet loss, link loss rate, delay for various combination. We implemented a test case using pox controller, then we considered a ryu controller. We added some hosts to the topology mentioned above. Again, a lighttpd web server was initiated. This was done for observing the resulting values.

#### 6.2 Future Improvements

Future improvements may make the project more effective. As providing security to network has become a growing concern, better cautionary measures may be useful for making a successful ARP poisoning detection.

- Number of hosts detected need to be increased
- Number of packet loss need to be reduced. Although when we used ryu controller, the loss was minimized at an extent.
- Depending on the controller, the algorithm may provide an excellent detection process. So various controller need to be tested for finding a best performance.
- For minimizing delay, necessary measures need to be taken. Then detection and prevention of ARP based attacks will be performed in a faster time.

