Embedding Location-Based Network Connectivity within IPv6 Address

A Thesis

Submitted to

The Faculty of the Department of Engineering Technology

University of Houston

In Partial Fulfillment

Of the Requirements for the Degree

Of

Master of Science in Engineering Technology - Network Communications Track

---

By

Bahaa Araji

May 2014

ACKNOWLEDGMENTS

For their advice, support, I wish to thank my brother Ali Araji and my cousin Mohamad Berjawi, without their help I am sure I won't do what I did in the past years.

For the two years of advice and encouragement, thanks to Dr.Deniz Gurkan, I learned a lot from you during this period of time.

I would also like to thank POF developers (Huawei Technologies) who have provided almost instant help with all my queries.

My deepest thanks to my parents who made me, the human being I am now.

Last but not least, thanks for who have always been there.

Embedding Location-Based Network Connectivity within IPv6 Address

An Abstract of a Thesis

Submitted to

The Faculty of the Department of Engineering Technology

University of Houston

In Partial Fulfillment

Of the Requirements for the Degree

Of

Master of Science in Engineering Technology - Network Communications Track

By

Bahaa Araji

May 2014

Abstract

IPv4 (Internet Protocol Version 4) the famous 32-bit address, has been used in networks for many decades [1] and would not have sustained its usability without NAT(Network Address Translation). IPv6 (Internet Protocol version 6) with its 128-bit address, provides slight routing information [2]. In this thesis, we present ESPM (Embedding Switch ID, Port number, MAC address), Embedding Switch Identification number, Port number and MAC (Media Access Control) Address within IPv6 protocol and SDN technology, imposing a device connectivity hierarchy upon the address space. We amend the IPv6 global addressing scheme for hosts to include their MAC address as well as the switch and port numbers that they are connected to. This scheme encodes information that would ordinarily require a lookup or query packets and decrease CAM (Content Addressable Memory) table entries on the switch by forwarding the packets using the ESPM algorithm. After processing ESPM algorithm to check for OF (Open Flow) controller ID, OF switch ID, and the Port ID, the amount of total packets transferred on the network to fulfill an ICMP (Internet Control Message Protocol) request-reply process decreased by 28.1% in 1-switch-2 host. In order to demonstrate the feasibility of such an addressing scheme, we use POF (Protocol Oblivious Forwarding) controller and POF switch [3] to implement ESPM and then measure the impact on the number of network management packets transferred between hosts during connectivity tests.

Table of Contents

LIST OF FIGURES

Chapter 1 Introduction

1.1 Ethernet and MAC

Network technologies are often split into layers based on the OSI model. The most popular network layer protocol used in networks is Ethernet[4]. Ethernet is the predominate standard applied to construct and access modern computer networks. Ethernet is promulgated by the Institute of Electrical and Electronics Engineers (IEEE) in various specifications as part of the IEEE 802 family of standards. Ethernet defines a number of wiring and signaling standards for the lower layers of the network. Ethernet networks carry all kinds of traffic over multiple types of physical connections (wired or wireless), including 10 mega-bits per second (Mbps), 100 Mbps, 1 giga-bits per second (Gbps), 10 Gbps, and 100 Gbps connections. It is used to transport data frames to the machines in a network based on their Ethernet addresses, also known as hardware addresses or Media Access Control (MAC) addresses. This essentially monitors the incoming data from the network by the node. A different address is allocated to every interface of a networking component. For example, a host with three physical Ethernet interfaces will have three different MAC address, one for each interface. The important feature of Ethernet addresses which authorizes this design is that all the addresses are unique. This guarantees that the Ethernet frame is transferred to the right interface without the necessity of any other information. An Ethernet Address is six bytes or 48 bits(e.g. 00:16:8C:a2:94:06).

Broadcasting denotes sending the Ethernet frame/Packet to every end-device on the local network. The domain of such frame/packet is bounded to a certain broadcast domain. For example, the boundary between distinct domains is marked

by routers. Broadcast addressing is not favored by all protocols and is fundamentally bounded to local area networks, as it does not impact the effectiveness of the network like in the case of wide area networks. Whenever a host wants to broadcast it will set all bits of the destination Ethernet address to one which hands an address of ff:ff:ff:ff:ff:ff. In this case every end device in the Ethernet broadcast domain is forced to get these frames. Broadcast is utilized when certain information is sent to all nodes, or if the destination hardware address is obscure. Dynamic assignments such as Dynamic Host Configuration Protocol (DHCP) and Address Resolution Protocol (ARP) are not possible without the use of Broadcast addressing.

Internet Protocol (IP) is the basic data-link protocol which is in charge of routing and shifting packets across the network gateways on the internet. This is executed using the IP address which is also known as logical address, which is assigned to every end device on the network. Addresses assignment can be static (Manual) or dynamic (using DHCP). The two popular versions of the IP address Protocol are IPv4 and IPv6. The vast majority of networked devices support IP version 4 (IPv4) defined in RFC-791. IPv4 provides a 32 bit address field for each of the source and destination of a packet. IP version 6 (IPv6) defined in RFC-2460, provides a 128 bit source and destination address fields. IPv4 32 bits address long is partitioned into 4 octets. Each octet is 8 bits and symbolizing a decimal number from 0 to 255 (e.g. 192.168.2.5).

1.2 ARP

Address Resolution Protocol (ARP) was first presented in RFC 826 [5] in 1982 and is still in use. It is a networking protocol which is utilized to dynamically discover the network layer address of a host in a network using its data link address. In any regular network, this mapping would be between the IP address (Network layer) and the Ethernet address (data link layer). This protocol has an

important role in linking the data-link and network layers together and permitting them to work together.

In any network, it is common for hosts to be allocated a new IP Address when they migrate or change their location, when their dynamic address lease time expires. The source host who initiated the communication will use ARP to detect the new IP to MAC address mapping for the destination host. For example, host 1 (source) with IP address 192.168.1.1 and Ethernet address MAC: aa:aa:aa:aa:aa:aa wants to reach host 2 (destination) with IP address IP 192.168.1.2 and MAC: bb:bb:bb:bb:bb:bb. Host 1 will broadcast an ARP query packet with its own IP and MAC address and Host 2's IP Address on its NIC. Since this message is a broadcast message, all hosts in the subnet will get it but only host 2 will reply because he owns the target IP address. Host 2, with IP address 192.168.1.2, will send a unicast ARP reply to Host 1 with its own MAC address. After Host 1 receives the frame, he will record the mapping of host 2 and starts sending the data to host 2. ARP packets do not have IP headers and hence ARP can analyze addresses only within the same local network, and the destination IP address has to be from the same subnet as the source IP address.

In order to decrease ARP broadcast messages for the same destination IP address, all nodes preserve an ARP table on each device, which is used whenever a mapping is needed. ARP request is only broadcasted if there is no matched entry in the table. The table basically works like a cache each one of the entries will expire if its not used for a certain amount of time.

ARP acts well in small networks since the number of broadcast messages in such can be limited, and ARP messages consumes a very small fraction of the available bandwidth. Nevertheless, broadcasting messages in large local networks (>10000 host) would not only eat more bandwidth, but would also stress hosts to process the extra unnecessary broadcast messages. This problem recognizes ARP as unscalable . A study at Carnegie Mellon University [6] scaled the number of ARP requests and replies which reached a host in a network of 2456 hosts. They study claimed that the host received 1150 ARPs per second and it averaged 89 ARPs

per second which adds up to about 45 kbps of traffic which can raise if the number of hosts increase.

1.3 IPv4 Packet Header

IPv4 packet header consists of application information, which contains usage and source and destination IP addresses. IPv4 packet headers include 20 bytes of data and 8 octets long.

A packet is a network communication data unit containing fixed or variable lengths. However, a single packet has three parts: header, body and trailer.A 20-byte header contains 13 multipurpose fields (Figure 1.1).

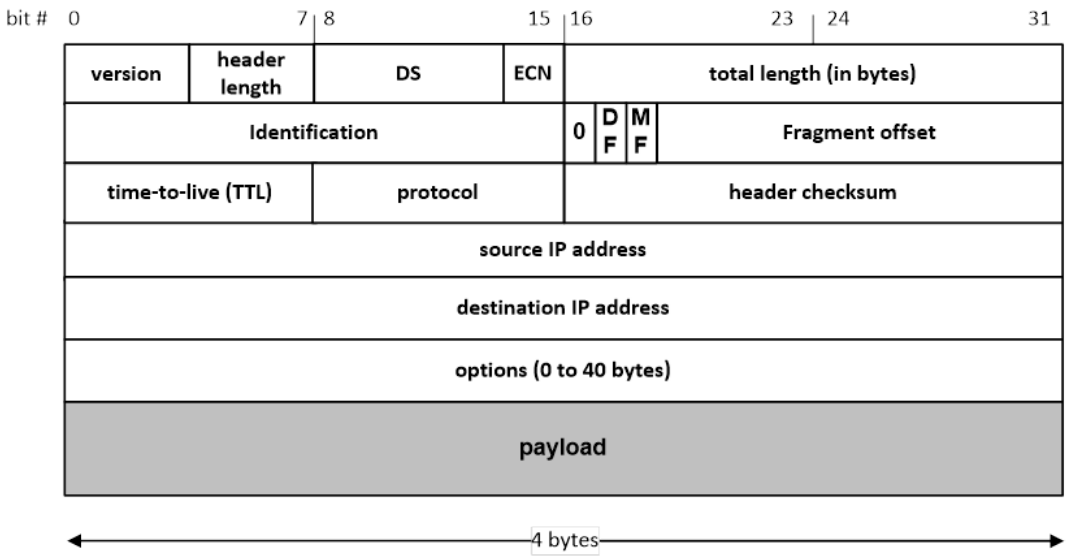

Figure 1.1 : IPv4 Header Format

The following are specific header field descriptions:

Version: it is the Internet header format (4 bits).

Internet header length (IHL): stores IP header length information(4 bits).

Type of service (ToS): provides network service parameters(8 bits).

Datagram size: contains combined data and header length(16 bits).

Identification: contains a specific number for primary data identification(16 bits).

Flags: Three flags (3 bits).

Fragmentation offset: This is a fragment identification via offset value(13 bits).

Time to Live (TTL): Total number of routers allowing packet pass-through(8 bits).

Protocol: Header transport packet information(8 bits).

Header checksum: Checks communication errors(16 bits).

Source address: Source IP address(32 bits).

Destination address: Destination IP address(32 bits).

Options: Its used for additional information.

1.4 IPv6 Header

The new IPv6 header is actually much simpler than IPv4 header. The IPv6 header has only 40 bytes, 32 of which are used for IPv6 addresses and the lasting 8 bytes by 6 additional fields. Unlike IPv4, IPv6 headers do not have any optional elements(Figure 1.2).

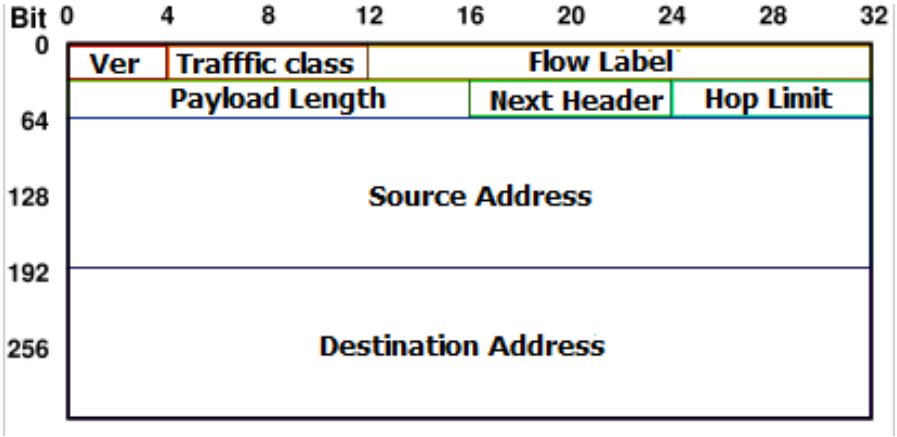

Figure 1.2 : IPv6 header Format

RFC 2460 defines the following IPv6 header fields:

Version : Same meaning from IPv4 to IPv6(4 bits).

Priority : Allows an application to specify the type of traffic(bits).

Flow Label :This will label a set of packets that belong to the same flow(24 bits).

Payload Length : Used to determine the length of the entire packet (16 bits).

Next Header : Indicates either the first extension header (if present) or the protocol in the upper layer PDU (such as TCP, UDP, or ICMPv6(8 bits).

Hop Limit: Identifies the number of network segments(8bits).

Source Address :Source IPv6 address of the packet(128 bits).

Destination Address :Destination IPv6 address of the packet(128 bits).

Extension Headers : IPv6 specification defines 6 extension headers

1.5 CAM Table

A CAM (Content Addressable Memory) table[7] is a fundamental component in the operation of the Ethernet switch. Ethernet switches link multiple computers on a single network, almost the same way like hubs. Unlike other networking devices, a switch contains a CAM table. The CAM table lets information routed through the switch to be sent to a single computer on the network, and not to all hosts connected to the switch. CAM tables can store a limited number of addresses for specific MAC ports. A CAM is a memory device that implements a lookup-table function in a single clock cycle using dedicated comparison circuitry. CAM is a hardware search engine that is much faster than algorithmic approaches for search-intensive applications. CAMs are formed of conventional semiconductor memory (usually SRAM) with added comparison circuitry that enables a search operation to complete in a single clock cycle. The two most common search-intensive tasks that use CAMs are packet forwarding and packet classification. In Internet routers, CAMs are among the most expensive circuit elements in a network device.

1.6 Claims

We invented a new technique to embed network connectivity information into the IPv6 address space through re-assignment of the address fields.

In order to justify the re-assigned address fields, we have implemented different test strategies. This thesis will document the invention and testing methods along with the measurement results.

I aimed to implement ESPM in conjunction with POF controller and POF switches, and compare its efficiency against OpenFLow learning switch. More specifically, we proposed to embed network topology information (e.g., switch ports, port number, and MAC address) into the IPv6 address of a host. In the next chapter we will continue discussing the background information about the experiment, while in the later chapters we will examine the implementation and evaluation of this system.

Chapter 2 Background

2.1 Technologies Used

Research was done to decide upon the suitable technologies, Controllers, Switches and monitoring software for the Implementation. ESPM implementation was intended to be a prototype, and not a deployable system.

2.1.1 OpenFlow and Beacon

OpenFlow [8], a programmable network control plane framework, suits in as the perfect candidate for my needs, as it permits, among other advantages, reconfiguration of the action of an OpenFlow-compatible Ethernet switch (referred to as the OpenFlow switch) and experimentation with network protocols. OpenFlow keeps the packet forwarding function in the switch, but shifts the high-level routing decisions out of the switch to an independent software controller which typically runs on a server, thereby authorizing the programmer to easily operate traffic in the software. The switch connects with the controller over a secure channel using the Open-Flow protocol (see Figure 2.1).

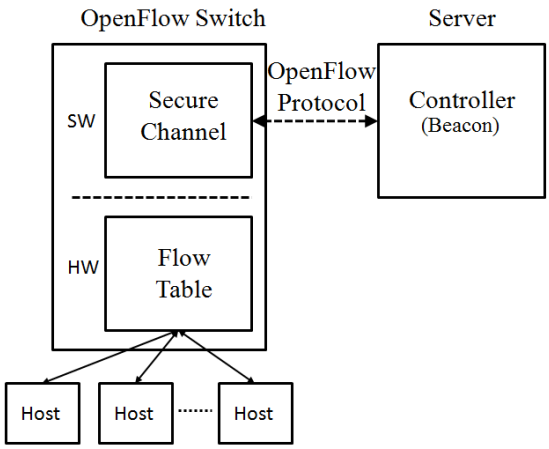

Figure 2.1: OpenFlow switch and controller

The OpenFlow protocol preserves a table in the memory of each switch which includes flow entries along with the coupled actions. A flow entry frames some of the fields from the headers of the data-link, the network, and the transport layers (see Figure 2.2), which are utilized to resemble the ingress packets. For example, a TCP connection or all packets from a particular MAC address would be seen as flows. Defined precisely, a flow is a set of packets whose headers give an exact match on any number of these fields in the flow entry, and/or a partial match on IP Source and Destination Addresses . An action can be any of the following:

- Forward packets through a specified port.
- Encapsulate the packet in an OpenFlow header and send it to the controller.
- Packet drop.

When an OpenFlow switch gets a packet, it inspects the header of the packet. If there exists a flow entry corresponding to the packet header, then the switch performs the linked action. In the case of no matching flow, the packet is encapsulated and forwarded to the OpenFlow controller. The controller, after inspecting the packet header based on the flow entries, decides and executes the best action for the packet. It may also forwards an OpenFlow packet including this information back to the switch for it to add to its flow entry table for the future packets of the same flow.

Using OpenFlow in the network to implement new network protocol is beneficial. This is because OpenFlow enables the programmer to deal

In Port	VLAN ID	Ethernet			IP			TCP	
		Src Addr	Dest Addr	Type	Src Addr	Dest Addr	Proto	Src	Dest

Figure 2.2: The header fields matched in an OpenFlow switch

With high level concepts of protocol implementation instead of having him try to run low-level hardware implementation when not required. OpenFlow protocol is widely supported and hence wraps a broader range of deployable hardware. OpenFlow is employed extensively by the network research community. It is easy to participate with them as most researchers are convenient dealing with the languages utilized for OpenFlow development (C++, JAVA and Python). In addition, OpenFlow abstracts away unneeded details and supports maintain the focus on the important features of the design.

A decrease in performance is expected when using OpenFlow because of the additional load of processing the packet in software. However, this decrease occurs only for the first packet of the flow, which needs to be sent to the controller in order to find the matched action for the remaining flow packets. All the future packets from the same flow would be handled directly by the switches hardware lookup table. This hit would impact the performance of our system when it comes to measuring the query latency, the time taken before a querying host receives a reply.

Major vendors such as Hewlett-Packard and NEC, are part of the OpenFlow and SDN (Software Defined Networking) project and make switches and routers which are capable of running OpenFlow protocol. Linux machines with multiple interfaces can be configured to operate as soft switches which support OpenFlow by using applications such as Open vSwitch [9].Beacon is a programmable OpenFlow controller which is used to control the behavior of OpenFlow switches

and routers by adding flow entries in their hardware tables for new flows .Beacon is a Java-based OpenFlow controller platform. It is very easily developed using the Eclipse Integrated Development Environment which runs on any operating system (OS).

2.2 Protocol Oblivious Forwarding (POF)

The problem with current SDN is that the placement of each protocol field, such as the IPv4 source address, is studied by the code that is preloaded into the devices according to protocol format by the device vendor. If one new protocol wants to be supported, the code must be modified.

The current OpenFlow-based SDN permits the programmability of network devices by downloading flows into devices from the controller, but automatic support of new protocols is an area that can be further developed. The programmability of current SDN can only affect the present protocols. If one service based on a new protocol wants to be deployed, the operator has to ask the device vendor to change the code of the devices to support the new service. This will drive to a long deployment cycle for new services based on new protocols.

POF indicates any protocol field with the following structure. The metadata is considered as one special protocol header that can be configured by the controller. Any metadata field is also denoted in the same way as below.

field {
type;
offset;
length;
};

The type" is used to indicate the field type, for which the value 0 means the field is a packet data field and the value 1 means it is a metadata field. The "offset" is

the field's start position relative to the current protocol head. The following example shows the MAC protocol header format.

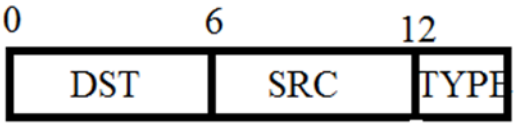

Figure 2.3: MAC address header format

There are three fields: dst, src, and type. They are denoted as follows:

dst: {0, 0, 48}; /*packet field, offset is 0bit, length is 48bit*/

src: {0, 48, 48}; /*packet field, offset is 48bit, length is 48bit*/

type: {0, 96, 16}; /*packet field, offset is 96bit, length is 16bit*/

It is easy to see that any existing or new protocols can be denoted in the similar way.

The Three classic forwarding instructions/actions in POF (See Figure 2.4) :

Goto-Table: instructs micro-code to match one table with keys extracted from the packet data or the metadata.

Set-Field: changes the value of a protocol field in a packet header.

Write-Metadata-From-Packet: copies the value of one packet field to the metadata memory.

POF uses multiple flow tables for packet processing to:

- Access control: forward/drop/send upward a packet.
- Output packet to designated port(s): unicast/multicast.
- Set/Modify the current protocol header.
- Copy the current protocol field to the metadata.
- Set the packet's committed access rate.

POF handles packet headers layer by layer. Each layer has one or more relative flow tables for parsing the next protocol, setting/modifying the current protocol fields, adding/deleting protocol headers to/from the packet, or copying the protocol header to the metadata. The flow tables dedicated to one layer cannot

handle any other preceding or rear protocol headers. Metadata can be used to hold the previous protocol headers.

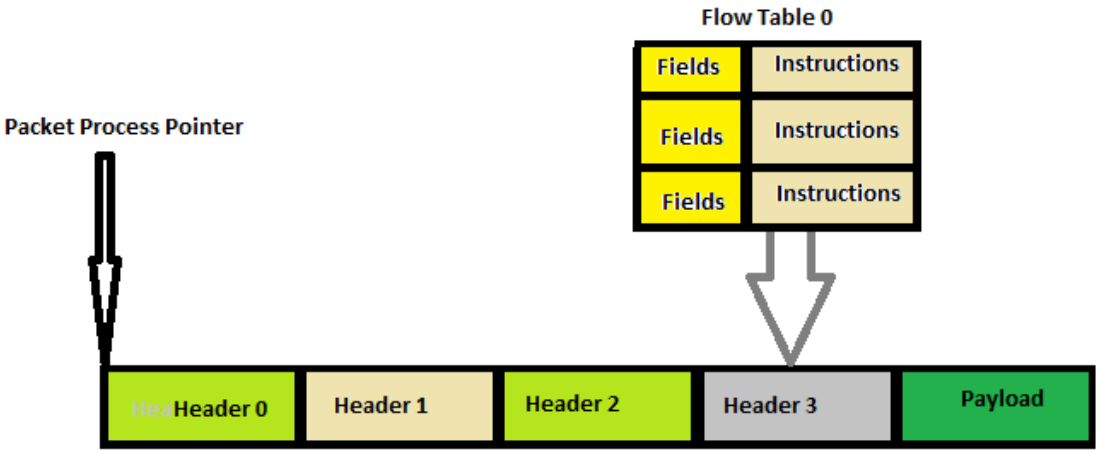

Figure 2.4: Forwarding process with POF

Chapter 3 ESPM Architecture

3.1 Location-Based Internet Addressing- A History

The Precursor to the Internet, ARPANET (Advanced Research Projects Agency Network) was the world's first operational packet-switching network [10], established in 1969 and it was implementing NCP (Network Control Protocol) to authorize users to access their computer at remote locations or to send and transmit emails and transfer files. ARPANET addresses were composed of 8 bits (6 bits for the site and 2 bits for the host). As civilian use for the network increased, the addressing system could not meet the requirements of the growing networks. Therefore, the addressing system was replaced by an integrated addressing system with centralized addressing capabilities, namely, the Internet Protocol (IP).

TCP/IP was created for connections among networks rather than devices. Since the origin of packet forwarding, the address was used to demonstrate the geographical location of the device (in the early days of addresses, the first 6 bits reflected the location).

In comparison with the original addressing schemes, ESPM proposes to add more information such as the embedded routing into the address. In this respect, such information will allow frame forwarding to be performed at line rate (with minimal table lookup) and with less overhead on the network devices and links.

3.2 Overview of the Approach

ESPM adopts the approach of dynamic host configuration protocol (DHCP) [11] and offers several advantages over regular dynamic addressing protocols. For example, it equips the addressing platform with data forwarding while dynamic addressing protocol just provides the logical information about the subnets used in the network. After defining this new design (ESPM), we will discuss the benefits of ESPM regarding forwarding table usage and decreasing broadcast packets on the network. In addition, we will examine the pseudo-code used in the ESPM operation and explore the basic operation and architecture of ESPM, such as broadcast/multicast and host mobility.

3.3 ESPM design

In ESPM (Embedding Switch number, Port number and MAC Address), as soon as a host is connected to a switch port, a new IPv6 address is assigned according to switch ID, port ID, and the host MAC address. Each switch is controlled by controller with a controller ID. The controller automatically and naturally determines the address of the host based on its location in the routing path: controller to switch, switch to port, and then port to host MAC. In this respect, an ESPM address (Figure 3.1) consists of (the assigned bit space here is given as an example):

- Site Prefix: 40 bits
- Controller ID: 10 bits
- Switch ID: 20 bits
- Port ID: 10 bits
- MAC Address: 48 bits

Similarly, a switch-to-switch connection (otherwise known as a trunk port) also gets an IPv6 address assignment through their corresponding controllers. Trunk

ports will be identified through their ESPM address that will be different than the regular ESPM host address by placing an all “1”s MAC address, i.e., FFFF:FFFF:FFFF. The controller and switch ID embedded in the trunk address will assist in providing the path information between different switches or controllers.

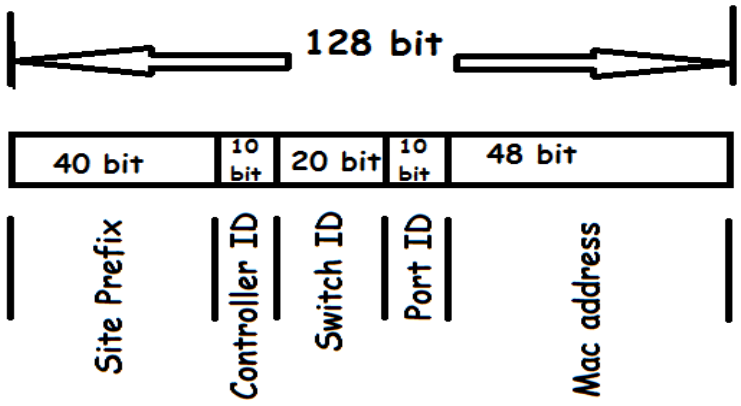

Figure 3.1: Example fields in the IPv6 represent site ID, controller ID, switch ID, port ID, and the MAC address.

3.4 Routing Protocol

ASSUMPTIONS: A centralized entity is envisioned to be assigning controller IDs. Every switch in an ESPM system will contact the controller they belong to (Local Controller) to retrieve a switch ID. In case the destination address is in a different controller ID domain, the applicable trunk port will be used to forward the packet based on flow settings in the switch. If no flow entry exists for such a controller domain, the switch will forward the packet to the controller. Within a site, we expect controllers to have full knowledge of the topology and trunk ports on the switches. However, the trunk port definition for ESPM holds true for even multi-site connections where edge in a site may connect to another edge switch at another site and their port addresses will reflect the site ID differences.

More specifically, the local controller will learn and create a neighbor table. Such a table can be visualized (Figure 3.2) and will be used to record and map the

ports of every switch to its adjacent and physically connected switches. In the neighbor table below we assumed that all switches are connected to the first 3 ports.

Switch ID	Port 1	Port 2	Port 3
Switch 1	Switch 2	Switch 3	Switch1(Controller 2)
Switch 2	Switch 1	Switch 2(Controller 2)	Switch 3
Switch 3	Switch 1	Switch2	Switch3(Controller 2)

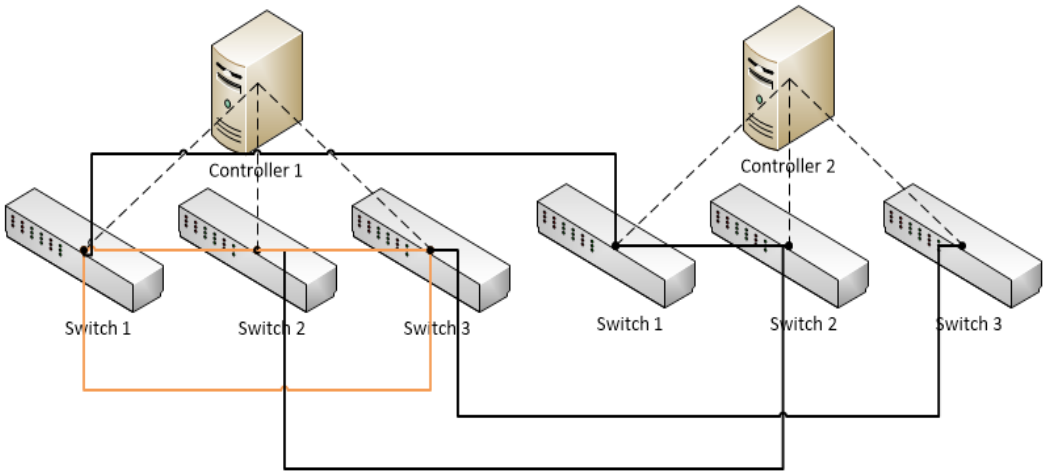

Figure 3.2: Network Topology of the neighbor table.

Using the bit field assignments in (Figure 3.1)

- 1099 billion site prefixes
- 1024 controllers on a single enterprise/site.
- 1.04 million switches for each controller ID (broadcast domain).
- 1024 ports for each switch ID.

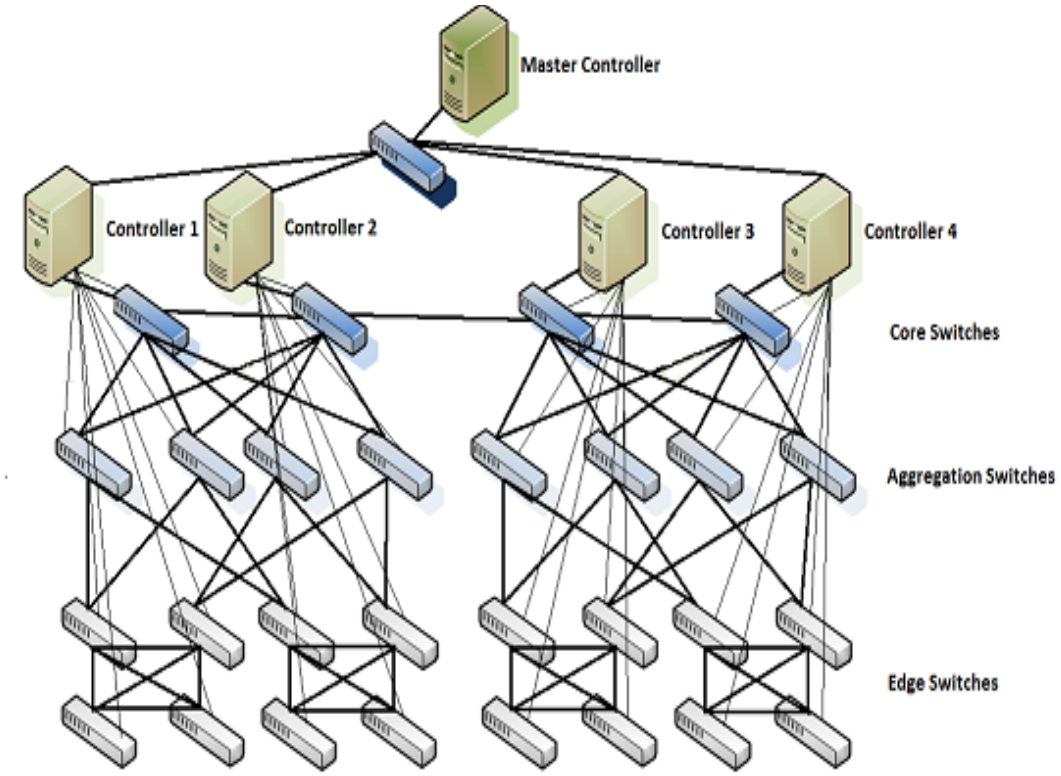

Figure 3.3: Three-Tier Network

Current routing protocols run a distributed algorithm and associated protocol to determine the optimal paths between end hosts. There is a hierarchy of route determination and exchange mechanisms. OSPFv2 [12] and IS-IS are the main link-state routing protocols with finer granularity algorithms such as Dijkstra algorithm [13] to find the shortest path in a hierarchical network. In ESPM, frames are routed through the network (Figure 3.3) to remote hosts by inspecting only the controller ID with the switch ID. Once the controller ID is established to be in the same domain with the source host, the switch ID is inspected next (Figure 3.4). If the controller ID and/or switch ID refers to a different controller or switch than where the host resides. The switch has to resort to the logically connected controller. The controller is going to check the neighbor table and Dijkstra algorithm to calculate and find the shortest path. The controller will push a flow with the best output port to forward this packet towards, so it can reach a switch in the destination controller domain.

A host may initiate a packet transfer with a specific destination address. The first hop switch will check its TCAM table for an entry to process the packet. If the switch did not find an entry, it will inspect the controller, switch and Port IDs of the destination ESPM address and if any of the controller ID or switch ID is different than the inspecting switch, it will direct the host's packet header to the controller to determine what flow to use. Controller will dissect this destination address to come up with the appropriate output port the packet should take at this particular switch by pushing a flow definition for the stream of upcoming packets. The port number and MAC address are inspected only at the destination host's home switch. An ESPM switch can be treated as a layer 2/layer 3 device, connects to hosts that acquire addresses starting with its switch ID, and delivers frames to other switches by passing them to appropriate neighboring switches [14].

Pseudo-code of the ESPM forwarding:

Start

Switch receives a packet from a host on one of its ports.
IF TCAM table can forward the destination packet
  THEN Send to destination port number
ENDIF
While controller ID != switch controller ID
Check local controller for next switch hop
    Add new TCAM entry and send to next switch.
        IF TCAM forwards the packet
        THEN Send to destination port number
        ENDIF
While destination switch ID != Switch ID
    Check local controller for next switch hop
    Add a new TCAM entry and send to next switch.
At next switch: forward to the port number embedded in the destination address
Check and verify MAC address for the host

End

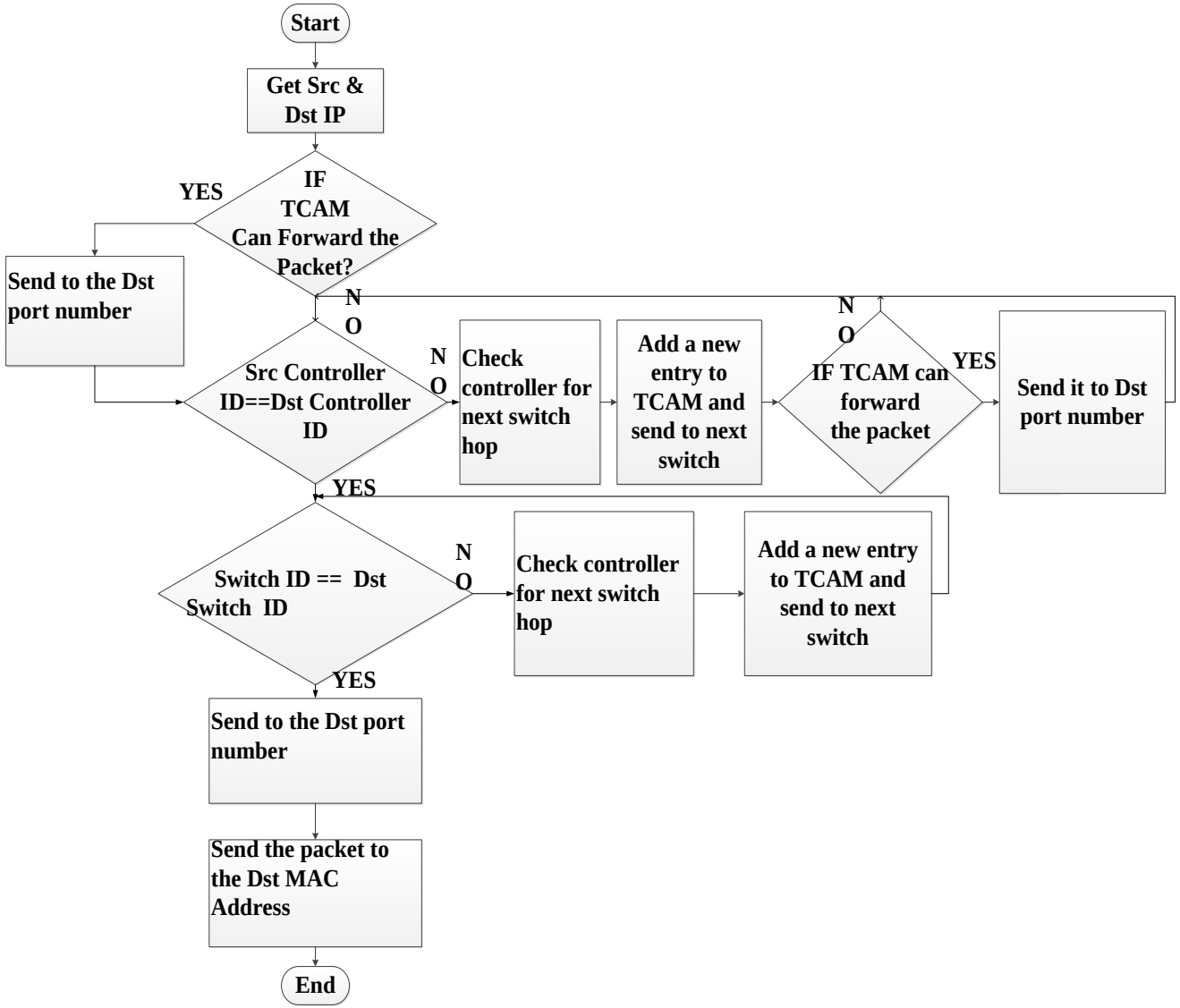

Figure 3.4: Flow chart for ESPM routing/forwarding

Static entries never expire and hence are never removed from the table. All mappings learnt from the network are defaulted to dynamic. This feature helps reduce broadcast traffic for machines with fixed IP addresses.

When an entry is about to expire, the table issues a unicast ARP request, which is directed to the mapping owner to verify if the mapping is still correct. The entry is marked stale until the table hears a reply from the host or the stale entry times out. Any requests for the stale mapping would be added to the Request Table. This is done to avoid additional broadcast requests for the entry which is being verified. If there is a reply, the stale entry is fully restored back into the table and all outstanding requests are satisfied. If the stale entry times out, then it is very likely that that the mapping no longer exists and the entry should be evicted from the table. The eviction is done by a table-cleaning thread which checks the timestamp for dynamic entries only. This thread runs once every half cache time so as to ensure that the entries are removed from the table within the next half cache time after expiration.

3.5 Multicast and Broadcast

IPv6 supports the use of multicast addresses where information or services can be enabled for a multicast group, i.e., group of interfaces. In ESPM, we will take the addresses that start with ff00 and use that as the multicast range for groups. This range (ff00::00 to ff00: ffff:ffff:ffff:ffff:ffff:ffff:ffff) will give us 16.7 million groups on every enterprise. Since multicast group members can be any device in the enterprise network, ESPM maps multicast groups to a special server that will manage multicast groups on the site. Controllers will forward all multicast join requests to the server, where the multicast groups will be created. When a device wants to send a multicast message to its multicast group, the message will always be forwarded up to the multicast-server and the multicast server will contact the controllers to disseminate the message to multicast group members. To broadcast a message to all members on the network, the multicast group FF02::1 is used,

and every host on the network join that group by default. To send a broadcast message, the destination address is simply set to ff02::1.

In case a broadcast message needs to be sent to:

• All ports on a switch: replace the port ID with FF

• All switches on controller: replace switch ID with FF

• All switches

on a site: replace controller ID with FF

3.6 Lookup Manager Services

A directory service, Lookup Manager, runs on the Master controller and in conjunction with the switches and controller. It handles the database for all assigned addresses on the Site. It adds the assigned addresses and whenever a host get removed from the switch port, the switch will contact the Lookup Manager and removes the host from the database. Lookup Manager will map the local Mac Address to a public-use Mac Address (uniquely and randomly assigned 48 bits address that will replace the last 48 bits in the Assigned address) for the hosts that connect with other sites and the internet.

The Look-up Manager will map the traffic in both ways. It will also replace the Public-use Mac Address with Local Mac Address for the incoming traffic from outside to the hosts inside the site.

3.6.1 Security Measures

The security and privacy implications of embedding hardware addresses in IPv6 address have been known and understood for some time now and the IETF deprecates the use of hardware addresses in IPv6 Interface Identifiers. To reduce and limit the attacks on the hosts using our addressing scheme, we placed Lookup Manager services to runs and coordinates with the core switches on each site. The

Lookup Manager will map and replace the local MAC Address to a Public-use MAC Address for the incoming and outgoing traffic. The Look-up Manager will strip the hardware MAC Address field from the IPv6 address for the hosts contacting the Internet and replace it with the Public-use Mac Address and vice-versa for the traffic in the opposite direction.

To weaken all possibilities of internal data center structure deduction during attacks, the Site ID, Controller switch ID and Public-use MAC Address will be a random, non-consecutive numbers. Only the port number will be given in a consecutive way according the switch's port number(e.g 1-24 or 1-48).

3.7 Mobility and Hosts Migration

A consequence of introducing location-based hierarchy into IPv6 addresses is the need to explicitly handle host mobility [15]. In a traditional Ethernet, hosts can migrate between switches as the switches will learn the host’s new location as soon as it sends a frame. With ESPM, if a host relocates to a new switch its IPv6 address changes and any of the CAM entries corresponding to the migrated host become incorrect; frames will continue to be sent to the host’s old location for a while.

Our strategy to deal with ESPM host migration is that on the arrival of the migrated host to the new switch, it will ask for an IPv6 address and the assigned IP address will be saved in the Lookup manager. The lookup manager will discover that the same physical MAC address acquiring two IPv6 addresses. To deal with this case, the Lookup Manager will send a message to the old switch to redirect incoming traffic to the new switch until the CAM entry that corresponds to that host expires, then it will remove the old IPv6 address from its database.

Chapter 4 Implementation

4.1 Connectivity use cases (Emulation)

4.1.1 POF with ESPM(Emulation)

We have emulated ESPM implementation by utilizing the OpenFlow match fields that correspond to the destination IP address, input port, and MAC address (Figure 4.1). If ESPM is fully implemented, the OpenFlow protocol would match the destination IPv6 address fields (with ESPM correspondence) to make a decision on the forwarding of the flows. In our implementation, we designed a network that contains a switch (OVS), a controller (NOX[16]), and two hosts (Case1). We assigned each host an IPv6 address within the ESPM scheme. Host A (MAC address: 1111.1111.1111) belongs to site ID=10, controller ID=1 and connects to switch ID=1 on port 1. Therefore, if we add the binary bits of all the five fields, we get the ESPM address for host A (Figure 4.2). Host B (MAC address:2222.2222.2222) connects to port 2 on the same switch and therefore, acquires the ESPM address ::4000:402:2222:2222:2222.

Assumption: In describing the flow of packets on the network, we excluded switch registration to the local controller and the keep-alive packets(Figure 4.3)

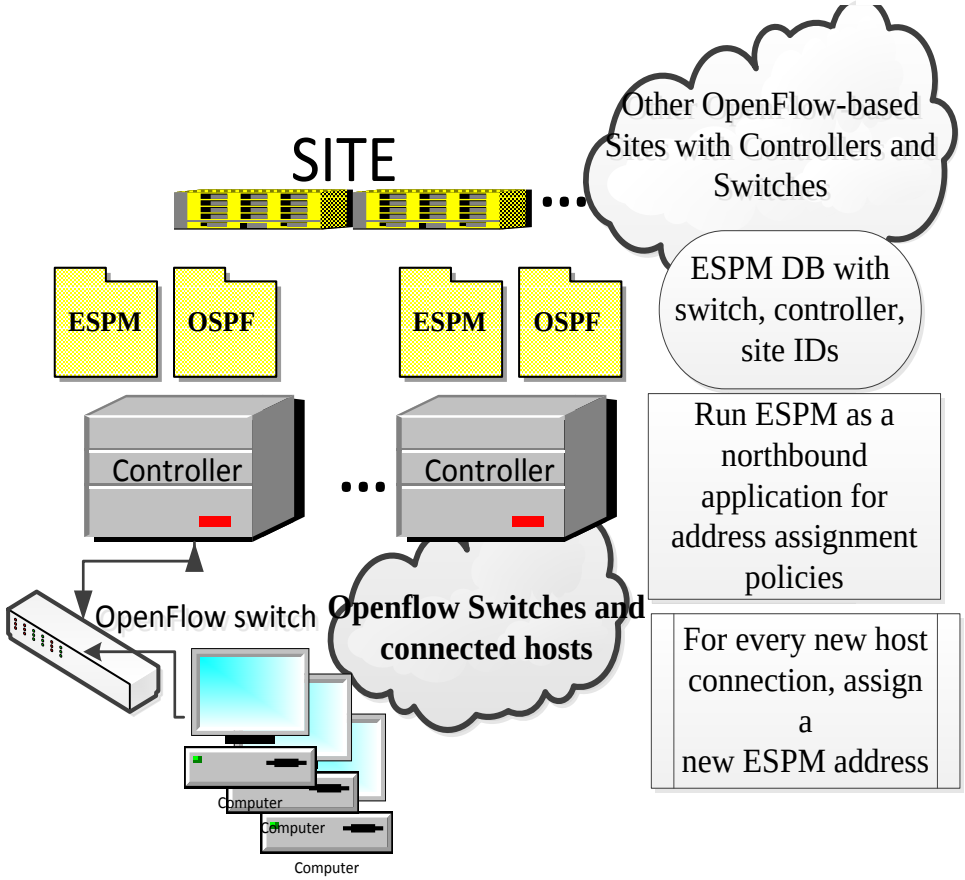

Figure 4.1: Network architecture implementation for ESPM-based addressing scheme.

Bootstrap protocol runs on each host when connected to the switch. The Bootp/DHCP "discover" packet is forwarded to the controller automatically because the switch will be pre-programmed with a flow entry for such packets by the controller. DHCP packet from the switch invokes the ESPM application in the controller. Controller will reply to the switch with the ESPM address assigned to the host. Switch will forward a "DHCP OFFER" packet to the host. DHCP process continues with the host sending a confirmation "DHCP request" with the assigned IP address to the switch. All such DHCP messages are pre-programmed to be sent to the controller by the switch. Therefore, the controller will send a "DHCP ACK" back to the switch. The switch will send the DHCP ACK packet back to the host completing the DHCP-based ESPM address assignment.

Host	Site ID	Controller ID	Switch ID	Port No	MAC Address (Hex)	ESPM Address(16-bit block Hex)
Host A	10	1	1	1	1111.1111.1111	::A00:4000:401:1111:1111:1111
Host B	10	1	1	2	2222.2222.2222	::A00:4000:402:2222:2222:2222

Figure 4.2 : ESPM addresses used in Case 1 and Case 2.

In our experiment, host A pings host B and destination IPv6 field inside the packet header will be analyzed by the ESPM code. Since the destination host acquires the same Site ID, Controller ID, Switch ID, the switch will find the port number that is embedded inside the ESPM address and forward the packet to that port.

No ARP messages are exchanged. In fact, the controller does not even need to insert a flow definition. However, the switch operates under ESPM principles with pattern matching of destination IP address fields: site, controller, switch IDs, and if all match, the port ID.

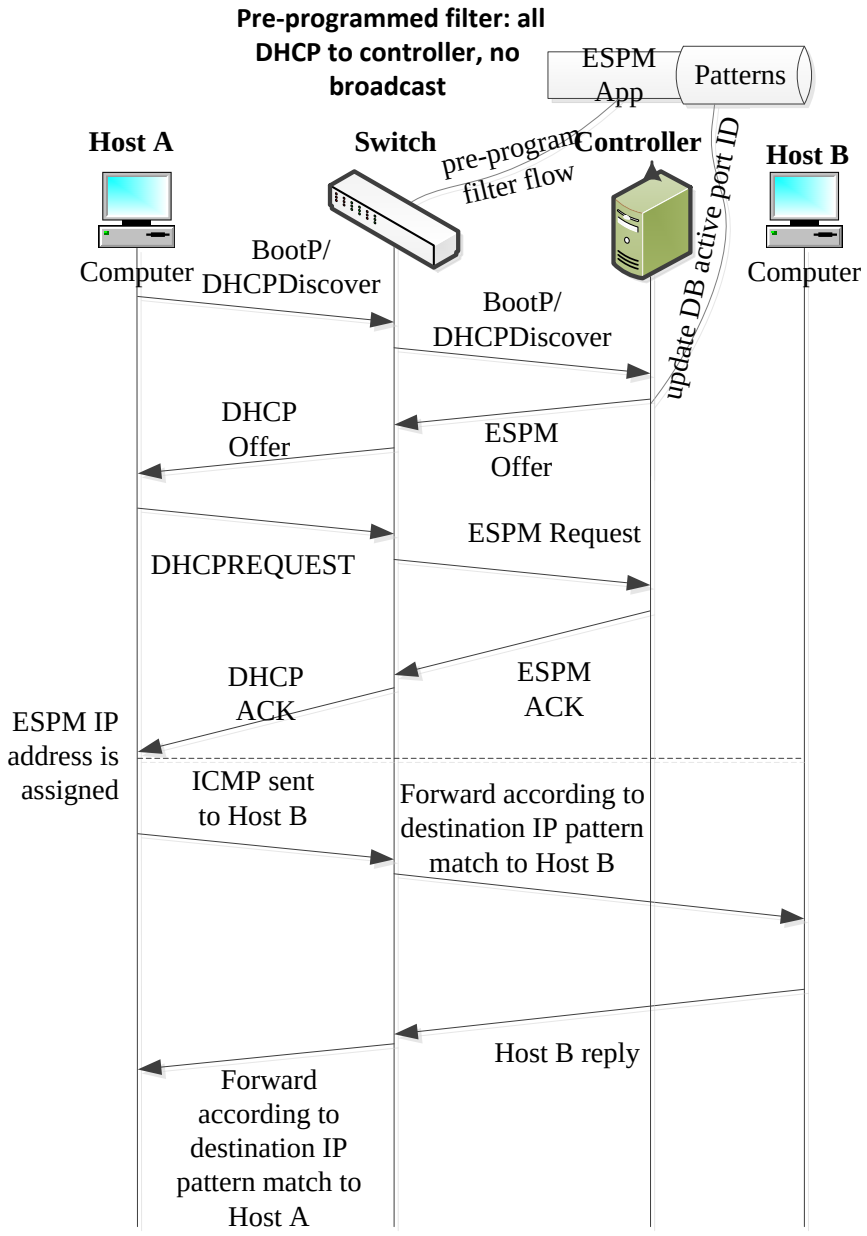

Figure 4.3: Case 1 timeline of events for address assignment and communication.

Result: An overhead of 8 packets per host has been exchanged before a ping message exchange was possible between host A and Host B. The total packets exchanged were 20. However, an application for ESPM has to be hosted to filter broadcasts from switches and maintain a data store for all IP address assignments, controller and switch IDs, and any active ports on switches.

4.1.2 OpenFlow without ESPM (Emulation)

In OpenFlow architecture, the switch learns the binding of MAC addresses to ports. Switch uses flooding of all other ports whenever destination address is not in the binding table. After the user turns host A power on, host A sends a broadcast request (DHCPDISCOVER), searching for a DHCP server to answer(Figure 4.4). The switch broadcast that to the controller. The controller, based on availability and usage policies ,finds an appropriate address (if any) to give to host A.Then host A sends DHCP request to reserve the IPv6 address, and gets back a DHCP ACK as a response for assigning IPv6 address to host A, and host B obtains the an IPv6 address in an identical fashion. After host A pings host B, on wireshark we found that host A sent an ARP request to the broadcast MAC ad-dress asking for the MAC Address of B. After receiving the first ARP, the switch has no flows yet and does not know how to forward this frame. It encapsulates the frame in an Open-Flow packet and sends it to the controller (packet IN). The controller orders the switch to send the frame out of all ports (except for the originating) (Packet OUT). As the switch floods the frame, it is received by host B. Host B will replies back to the switch, the switch asks the controller for a flow, using an OpenFlow control packet.

The controller knows where the destination MAC address exists on the network and orders the switch to create a flow. This flow can be used by the switch if any following frame with the same features should arrive, finally the frame is delivered to host A. Now that host A is familiar with the MAC address of host B, it will be able to send ICMP request.

Host A sends an ICMP request and the switch forwards it to the controller who is aware now of host B location, so it creates a flow with the specific traffic. ICMP reply happens in the same manner as ICMP request. On wireshark we found that

28 packet (except switch registration to the controller and keep alive packets) where exchanged to achieve ping between hosts.

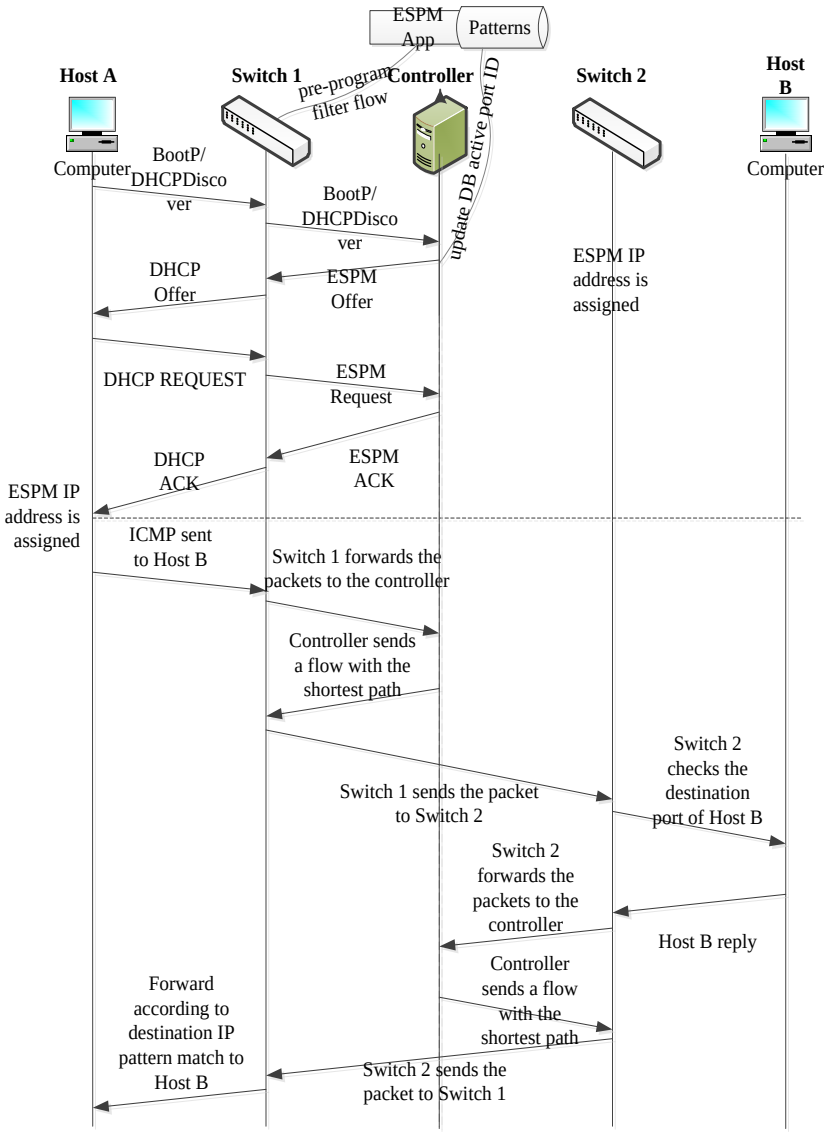

Figure 4.4: Timeline of events for address assignment and communication in one Switch—one Controller Network.

The controller knows where the destination MAC address exists on the network and orders the switch to create a flow. This flow can be used by the switch if any following frame with the same features should arrive, finally the frame is delivered to host A. Now that host A is familiar with the MAC address of host B, it will be able to send ICMP request.

Host A sends an ICMP request and the switch forwards it to the controller who is aware now of host B location, so it creates a flow with the specific traffic. ICMP reply happens in the same manner as ICMP request. On wireshark we found that 28 packet (except switch registration to the controller and keep alive packets) where exchanged to achieve ping between hosts.

4.2 Test-bed Used

The aim of the implementation was to examine the behavior of ESPM and compare its performance against that of OpenFlow forwarding procedure. The experiments conducted measured the effectiveness of ESPM in decreasing control messages in the network and hence eliminating the unnecessary broadcast packet processing at the hosts.

Since the purpose of the implementation and evaluation is to compare between two systems, we had to create two independent test-beds and compare their behavior. The First test-bed was used to evaluate the ESPM method, while the second one was used to evaluate OpenFLow forwarding procedure.

4.2.1 POF Test-bed

We chose to use physical machines instead of virtual machines to build the POF network topology. The network topology designed for this experiment (Figure 4.1) includes four desktops (POF Controller—POF Switch—Host1—Host2).

The machines used for the experiment were running Ubuntu 12.4 with one network card and one wireless network card installed. For the POF Switch we had to replace the one port network card with four ports network card. On the POF Switch, the four interfaces were used to implement the ESPM system. After installing Ubuntu 12.4 on the machines, I started cabling the machines according to the topology, since the desktops need to connect directly; I connected them with crossover cables. The next step to achieve basic connectivity between

neighbor devices was to assign IP addresses on the network cards from the same subnet.

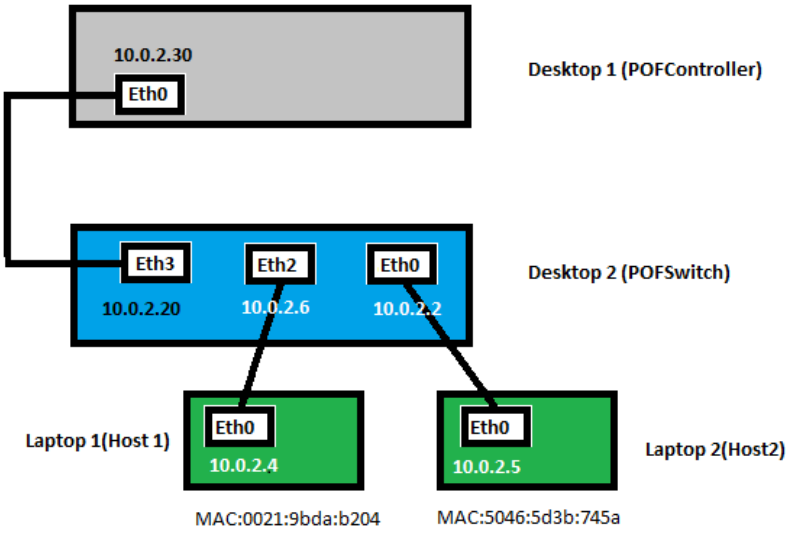

Figure 4.5: POF test-bed

I started preparing the POF Controller machine by installing Java 6 JDK, JRE and Eclipse. Then I downloaded the POF controller software and built it using Eclipse and launched the POF controller with a GUI panel.

On the POF Switch, after downloading the POF Switch software and unpacking the software tar file, I built the POF Switch manually with the "./configure" and "make" commands[17]. Then I started and established a connection with the POF Controller using the command "pofswitch –I 10.0.2.30" where 10.0.2.30 is the POF Controller IP address.

For POF test-bed, the basic operation procedure is as below[18]:

1. Launch the POF Controller.
2. Launch the POF Switch (e.g. a software switch or a POF-enabled router).
3. Hello handshake process starts automatically.
4. Create new packet types in GUI panel on the POF Controller.

5. Create new tables in GUI panel.

6. Create new table entries in GUI panel.

7. The Switch side will get the OpenFlow Messages from the controller and creates the tables, entries, etc., as directed. When the switch data path receives any packets, the switch will process the packets based on the configured processing flow.

So, after achieving a successful connection between the POF Switch and POF controller, I started creating my packet header as shown in (Figure 4.6).

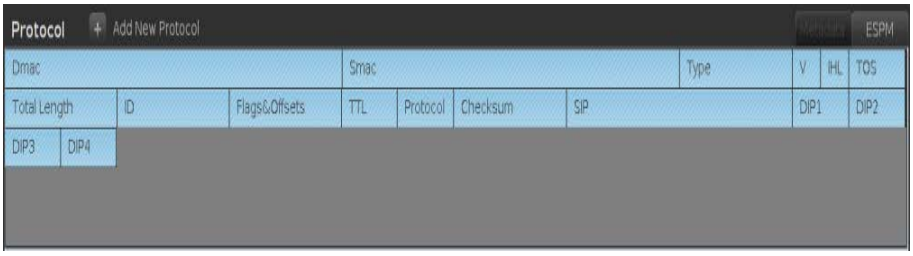

Figure 4.6: Creating Packet header in POF Controller

After creating the Packet header, I started creating the flow table as shown in (figure 4.7). Since the switch port number is embedded in the Hosts IP address, the flow table should take the last octet of the destination IP address which represent the switch port number(DIP_4 field) as a find key. According to the find key value, the instructions “APPLY_ACTIONS” will be to set the “Output Port ID” as the find key value.

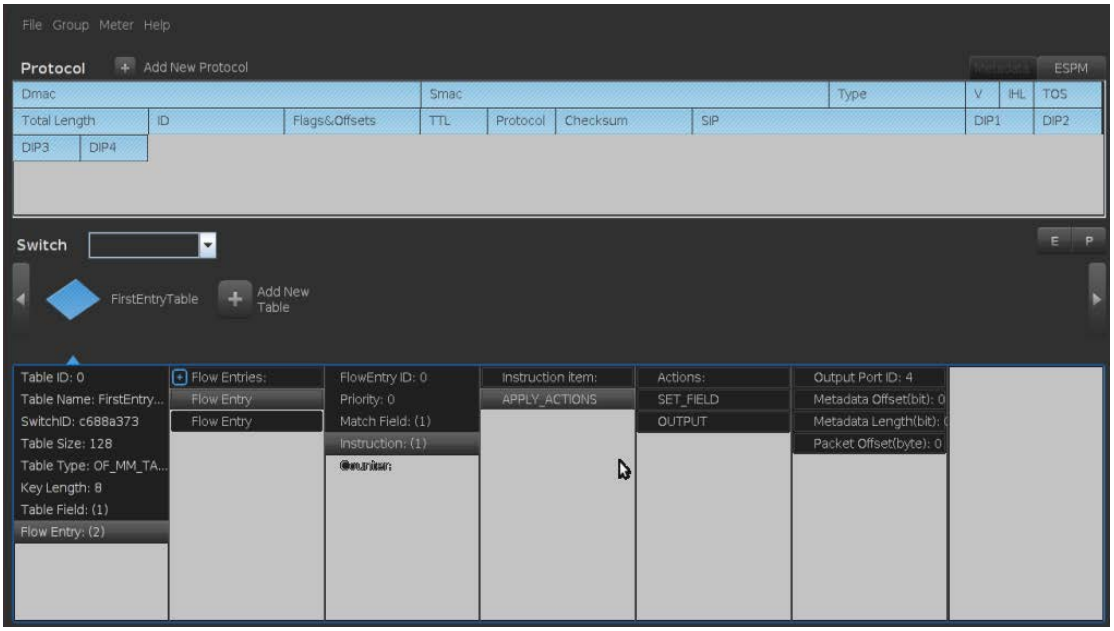

Figure 4.7: Creating a flow Table.

After creating the flow table, I submitted the instructions successfully to the switch so it can process the traffic that will be generate later.

4.2.2 OpenFlow based test-bed

Due to a limited number of available test machines, we implemented the OpenFlow test-bed on VirtualBox using Mininet and Beacon OF Controller. Mininet is a network emulator which creates a network of virtual hosts, switches, controllers, and links. Mininet [19] hosts run standard Linux network software, and its switches support OpenFlow for highly flexible custom routing and Software-Defined Networking. After downloading and installing mininet and Eclipse on VirtualBox, I started Mininet using the command “sudo mn --topo single,2 --switch ovsk --controller remote” to create the topology shown in (Figure 4.8) and built the Beacon controller in Eclipse so it can connect to the Virtual switch in Mininet.

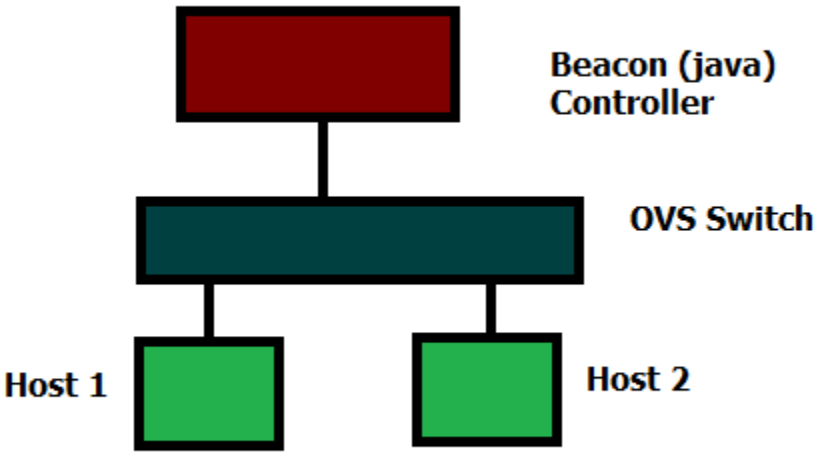

Figure 4.8: OpenFlow Learning Switch Topology

So far in the OpenFlow test-bed, we created a simple OF Controller—OF Switch-Host1 –Host2 topology were the OF Switch is acting as a L2 learning switch. A L2 learning switch learns the mapping between MAC addresses and ports by watching packets. If the switch has already seen a particular destination, it can send to exactly one port; otherwise it must flood the packet out all ports, like a hub. We verified the learning switch behavior by achieving successful ping between Host1 and Host2.

4.3 Implementing ESPM on GENI

GENI (Global Environment for Network Innovations) is a facility concept being discussed by the United States computing community with NSF (National Science Foundation) funding. The purpose of GENI is to increase experimental research in computer networking and distributed systems, and to speed the transition of this research into products and services that will upgrade the economic competitiveness of the United States. GENI planning efforts are established around several focus areas, covering facility architecture, the backbone network,

distributed services, wireless/mobile/sensor sub networks. GENI is being actively used for network research and education [20].

ProtoGENI is GPO-funded prototype implementation and deployment of GENI, drove by the Flux research group at the University of Utah, and largely based on our Emulab software. ProtoGENI is the Control Framework for GENI Cluster C, the largest set of integrated projects in GENI. We implemented our experiment (ESPM) on Proto GENI by starting with the following steps:

- Getting Emulab account
- Generate SSL Certificate
- Registering a slice name
- Allocating computer resources

Although ProtoGeni allows you to allocate PCs and VMs from its resources, but for our experiment we need PCs, so we can remotely connect to the Ubuntu desktop installed on them. We had some issues in allocating PCs resources from ProtoGeni, Only Utah genirack had available resources at that time, so we used four PCs to create our topology (Figure 4.9), and installed the same operation system and software as the one we are using for the physical testbed.
I am using VNC viewer software to connect to the GUI interface of the machines, and Putty software to connect to the CLI platform of the machines.

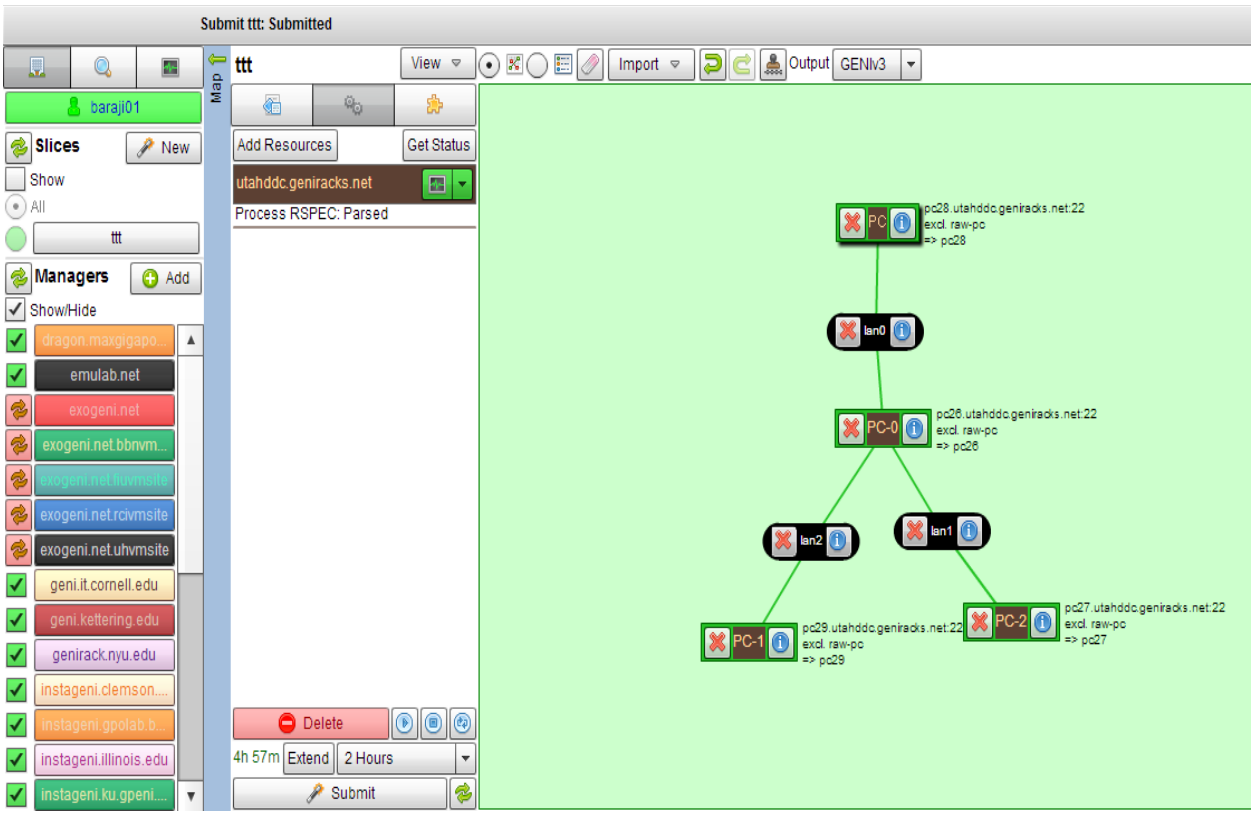

Figure 4.9: ESPM topology implemented on ProtoGeni

4.4 Testing

Testing was a definite phase of the development procedure so as to secure that the system is working reliably and behaving as expected. Testing frameworks were deployed to capture and filter the traffic, counts packets, and present it in real time.

4.4.1 Testing Strategy

To test the complete ESPM system and compare its behavior to OpenFlow learning switch, I executed a test that is going to trigger real ICMP traffic from the ping command to propagate along both networks while the link between Host2 and the Switch (POF Switch in ESPM and OVS switch in OpenFlow

Learning switch) were going up and down every 30 second. I wrote a script in shell to perform this connect/disconnect action on the link. The behavior of the traffic while we are sending continuous ICMP packets was examined using the Wireshark network protocol analyzer. Wireshark allowed us to track the exact path of each packet including OpenFlow traffic, in the network by monitoring the flow of packets at each network interface. The next step after tracing the packets was to capture and filter them. Our concern was the Arp, ICMP and Broadcast packets since in our emulation experiment we came to the result that Arp and broadcast packets will be eliminated from the total traffic exchanged during ICMP process. We presented the filtered traffic in a real-time graph that compare the number of ARP, ICMP and Broadcast packet [21].

Chapter 5: Evaluation

5.1 Experiments Result

The primary objective of ESPM is to cut off unwanted ARP traffic and the amount of broadcast traffic in the network. This is because ESPM eliminate the need for broadcast; it will find the MAC address of the destination host by extracting it from the destination address. This eventually helps to decrease unnecessary ARP processing at the hosts. The amount of benefit that the network can draw out of ESPM is proportional to the size of the network.

In both experiments, we are filtering for ARP, ICMP and Broadcast Packets. Such OpenFLow traffic as Echo Request and Reply packets which are sent to maintain the connection between the controller and the switch alive. Echo requests consist of arbitrary data that are sent only once every 15 seconds, which the switch simply wants to echo back. These packets are unicast messages that are handled by the switches and not the hosts, and they do not load the network.

5.1.1 Results and Graphs

In both experiments, we lunched ping from Host1 to Host2 while running the shell script to connect/disconnect the link between Host2 and the Switch. Wireshark was used to monitor the flow of these ARP, ICMP and Broadcast packets in the network and screenshots were taken from each experiment on Host1 network interface, the interface between the Switch and the Controller and Host2 network interface. Screenshots of these Wireshark capture are enclosed in Appendix A.

The results from both tests are summarized in the graphs in Figure: 5.1 and Figure: 5.2

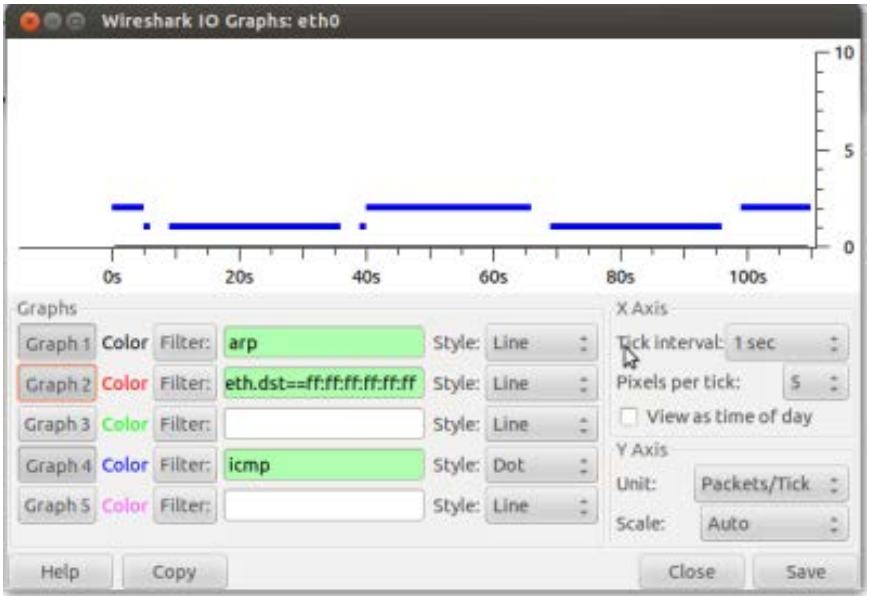

Figure 5.1: Arp, ICMP and Broadcast packets graphical representation in ESPM experiment.

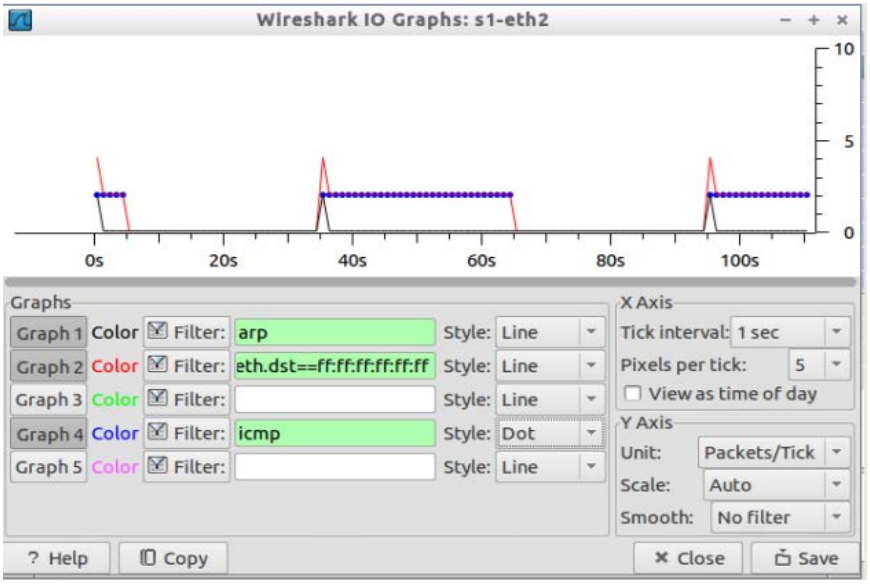

Figure 5.2: Arp, ICMP and Broadcast packets graphical representation in OF Learning Switch experiment.

In both graph, the primary axis, depicts the number of ARP, ICMP and Broadcast packets sent or received by the target host over time.

5.1.2 Reduction in Packet

With this implementation, we attempted a comparison of ESPM forwarding behavior to Ethernet forwarding manner in OpenFlow. From both implemented testbed, we produced real traffic using ping to send ICMP packets. We confirmed that packets were being sent to the exact host by using Wireshark software. We examined the packets over the network before and after enabling ESPM to find the benefits of the addressing scheme. The switch forwarding table which maps the host's MAC address to the host's port number can be removed when we use the ESPM algorithm to process the packets on one switch two hosts network. The chart below (Figure 5.3) show the decrease in total packets transferred to achieve ping between host 1 and host 2 by 28.1%.

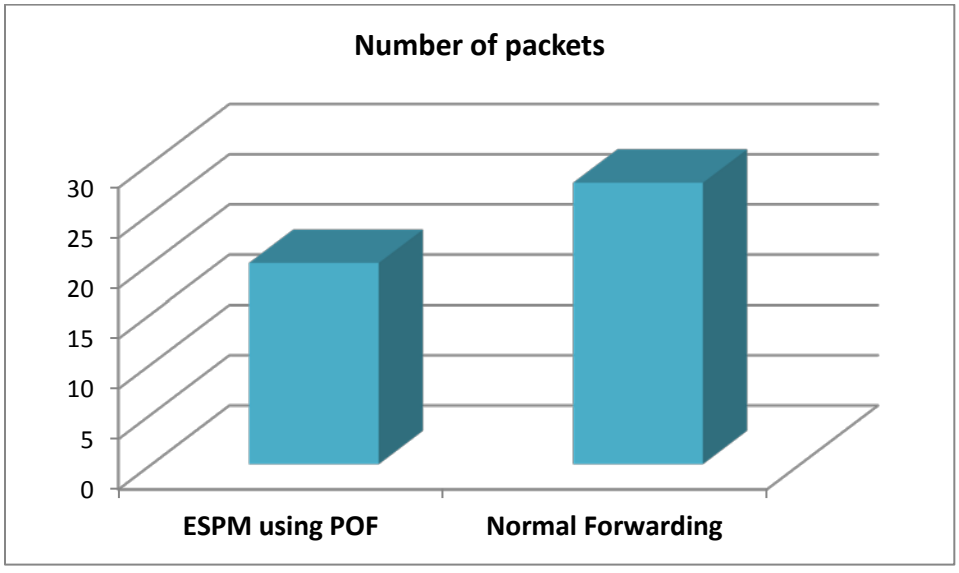

Figure 5.3: packets exchanged between host1 and host2 during ping process in ESPM and OpenFlow learning switch forwarding.

In ESPM, when a packet arriving at the switch ingress port will be checked for its controller/switch IDs and when they match, there is no flow/action pair needed to forward. The port ID embedded into the destination IP address will suffice to forward the packet. The communication with a controller on flow entry is also saved. Figure 5.3 displays the complete packet count during a ping exchange from host 1 to host 2.

5.1.3 CAM/TCAM savings

The switch will check if the TCAM table can forward the packet. If not, the switch will process the incoming packet by inspecting if the destination host is on the same controller and switch IDs, so it can forward the packet to the destination without adding a new entry to the forwarding table (CAM table). If the destination host is not on the same switch, the controller will add a flow to the switch, with the specified match/action fields. The inserted flow will be represented in the TCAM table as a new entry. ESPM can lead to major decrease in the size of the forwarding table, by removing the need for a CAM table and relying on ESPM algorithm and TCAM table, to process the incoming packets.

5.1.4 Summary of Results

Through the implementation and evaluation of ESPM, we have successfully collected positive proof demonstrating that further research into ESPM is a worthwhile endeavor. The evaluation displayed significant reduction in the total number of management messages across the network. This reduction develops the overall performance of the network.

Chapter 6: Conclusion

6.1 Future Work

We implemented ESPM architecture on physical machines and on GENI testbed using the SDN experimentation capabilities and ran experiments to test its effectiveness and draw an analogy between its performance and conventional forwarding at layer 2. We loaded up the responsibility of parsing of header fields of IPv6 packets to a northbound application in order to setup reactive and proactive flows for IPv6 networks. Both environments allowed us to test our location-based approach with programmable portability and showed significant improvement as predicted, but there are still areas that need more research as:

- Scalability of the ESPM with many domains.
- Mobility constraints on the addressing scheme (to be emulated using VM migrations between networks).
- Connectivity and discovery considerations for new ESPM nodes.
- **DHCP in ESPM**: In the existing implementation, the IP addresses are either manually given by the administrator or dynamically allocated by a separate DHCP server. Look-up Manager Server teaches the host's Controller ID, Switch ID a, Port number and MAC address by listening to incoming DHCP requests from the host during boot up process. This implies that the Lookup manager can assign IP addresses and save them in its database so we can improve IP conflict detection.

List of references

[1] V. Cerf, and R. Kahn, "A Protocol for Packet Network intercommunication", IEEE Transactions Communications, Vol. Corn-22, No. 5, May1974 pp. 637-648.

[2] S.Deering, and R.Hinden,"Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998 http://tools.ietf.org/html/rfc2460.

[3]H.Song "Protocol-Oblivious Forwarding: Unleash the Power of SDN through a Future-Proof Forwarding Plane ", Sigcomm 2013, http://conferences.sigcomm.org/sigcomm/2013/papers/hotsdn/p127.pdf.

[4] R. M. Metcalfe and D. R. Boggs, "Ethernet: distributed packet switching for local computer networks," ACM Communications, vol. 19, no. 7, pp. 395–404, 1976.

[5] D.Plummer, "Ethernet Address Resolution Protocol: Orconverting network protocol addresses to 48.bit Ethernet address for transmission on Ethernet hardware", STD 37, RFC 826, November 1982.

[6] A. Myers, E. Ng, and H. Zhang, Rethinking the Service Model: Scaling Ethernet to a Million Nodes, in ACM SIGCOMM Workshop on Hot Topics in Networking, Nov. 2004.

[7] K. Pagiamtzis and A. Sheikholeslami, "Content-Addressable Memory (CAM) circuits and architectures: a tutorial and survey," IEEE Journal of Solid-State Circuits, vol. 41, pp. 712–727, 2006.

[8] N. McKeown, T. Anderson, H. Balakrishnan,G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," ACM SIGCOMM Computer Communication Review, vol. 38, April 2008.

[9] B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, and S. Shenker, "Extending networking into the virtualization layer," Proc. HotNets, Oct. 2009.

[10] DARPA INTERNET PROGRAM "INTERNET PROTOCOL", RFC 791, September 1981.

[11] R. Droms, "Dynamic Host Configuration Protocol,"RFC 2131 (Draft Standard), Mar. 1997, updated by RFCs 3396, 4361.

[12] J. Moy, "OSPF Version 2", RFC 2328 (Standard), http://www.ietf.org/rfc/rfc2328.txt, Apr. 1998.

[13] E. Dijkstra, "A note on two problems in connexion with graphs," Numerische Mathematik, 1959.

[14] D. Wagner-Hall" NetFPGA Implementation of MOOSE", May 2010. http://www.cl.cam.ac.uk/~mas90/MOOSE/dwh-diss.pdf.

[15] C. Perkins, "IP Mobility Support for IPv4,"RFC 3344 (Proposed Standard), Aug. 2002, updated by RFC 4721. [Online]. Available:http://www.ietf.org/rfc/rfc3344.txt

[16] The NOX Team, "Developing in NOX." noxrepo.org/noxwiki/index.php/ Developing_in_NOX. Retrieved 03-30-2011.

[17] Y. Jingzhou, W. Xiaozhong "POF Switch Introlduction" http://www.poforwarding.org/document/POFSwitch_Introduction.pdf.

[18] Jian Song, Zahi Chai "POF Controller Introduction" http://www.poforwarding.org/document/POFController_Introduction.pdf.

[19] The Mininet Team " Introduction to Mininet" https://github.com/mininet/mininet/wiki/Introduction-to-Mininet.

[20] http://en.wikipedia.org/wiki/Global_Environment_for_Network_Innovations

[21] I. Aggarwal "Implementation and Evaluation of ELK, an ARP scalability enhancement", Corpus Christi College, May 2011.

Appendix A

Wireshark Screenshots

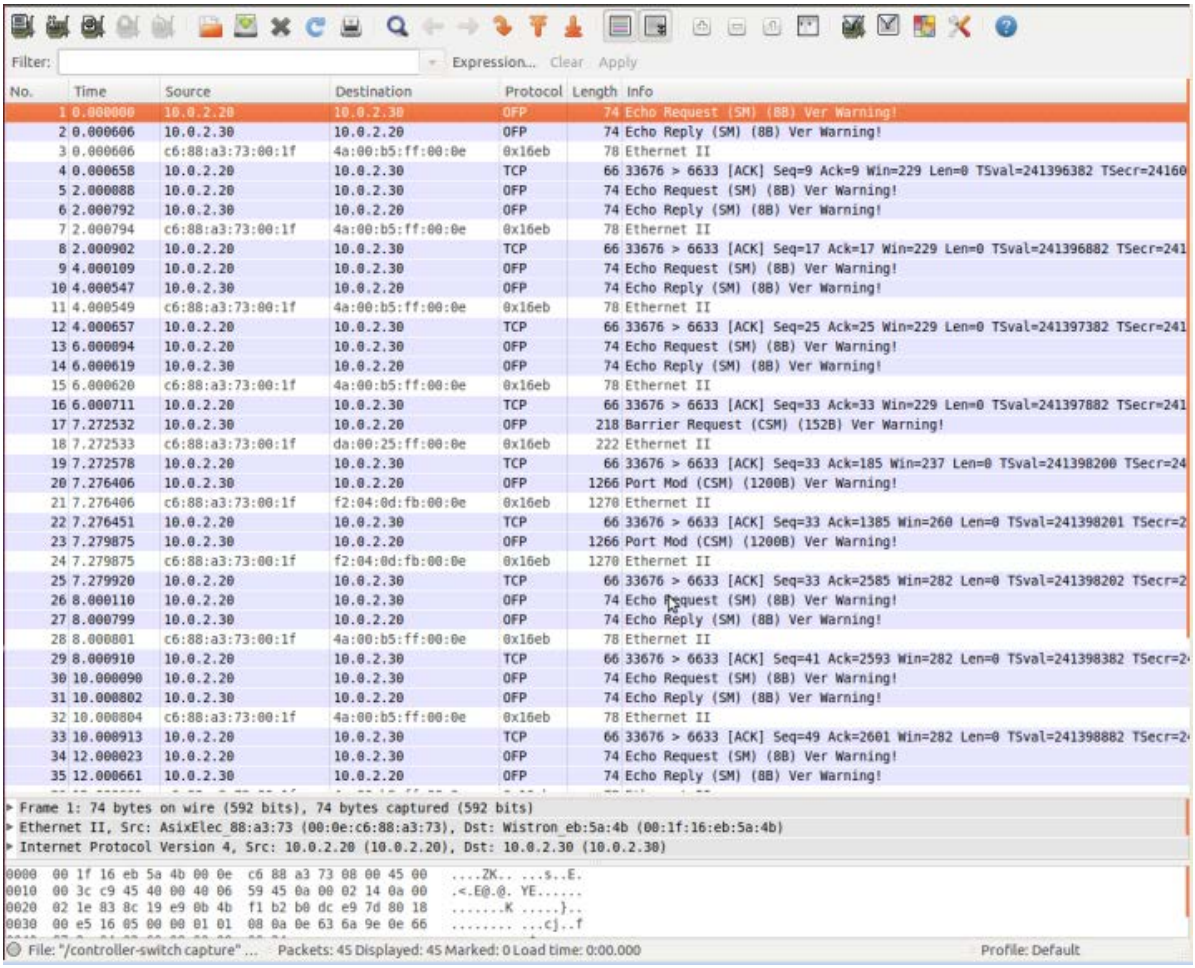

Figure A.1: Screenshot of Wireshark packet capture for the packets transferred between the POF Controller and POF Switch.

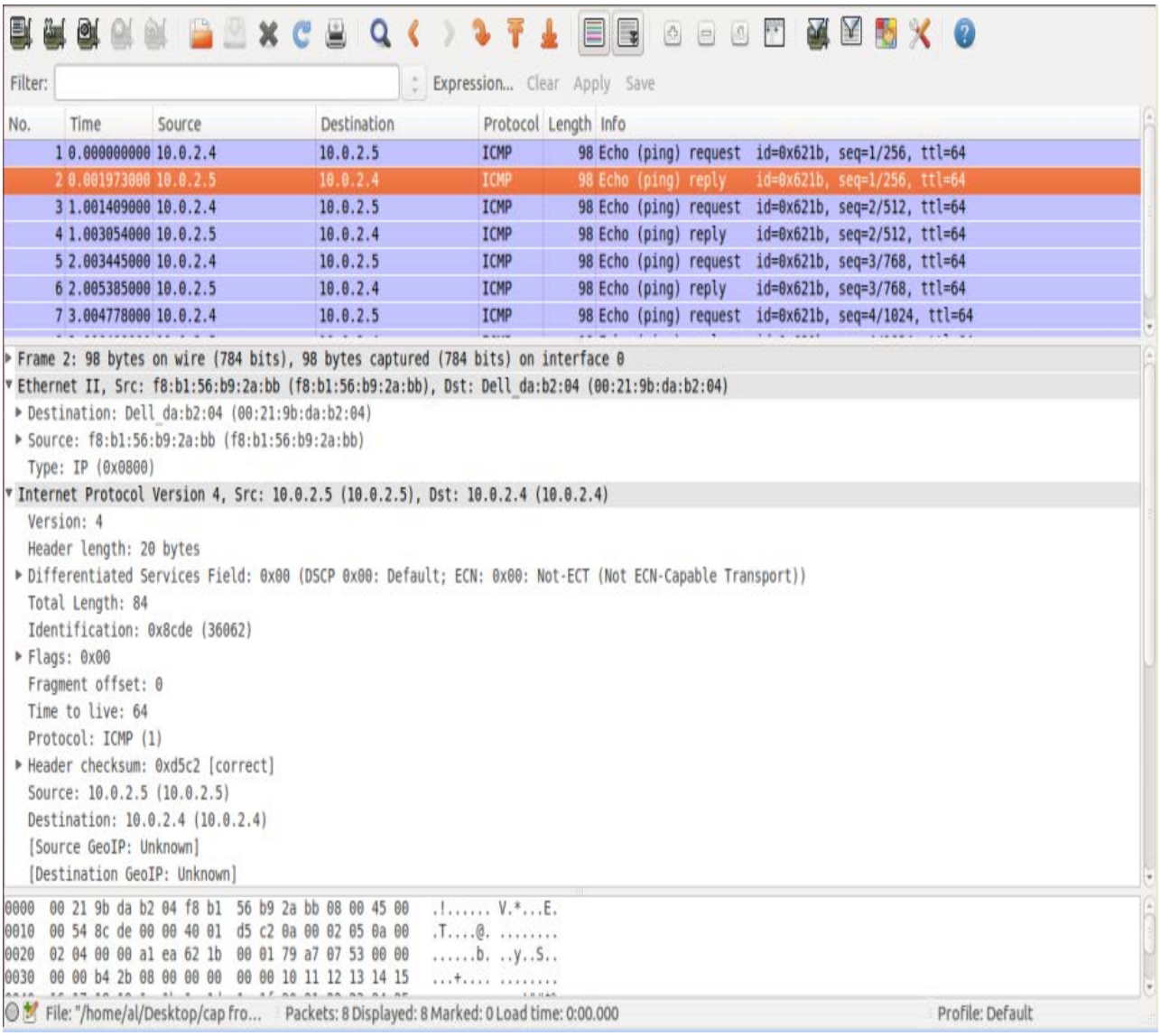

Figure A.2: Screenshot of Wireshark packet capture for the packets sent and received by host1.