Proof Of Concept for Opportunistic Security in MPLS Networks

Salil Ajgaonkar B.E

A Dissertation

Presented to the University of Dublin, Trinity College

in partial fulfilment of the requirements for the degree of

Master of Science in Computer Science(Future Network Systems)

Supervisor: Stephen Farrell

August 2018

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Salil Ajgaonkar

August 30, 2018

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Salil Ajgaonkar

August 30, 2018

Acknowledgments

First and foremost I would like to thank my thesis advisor Dr. Stephen Farrell of the School of Computer Science and Statistics at Trinity College Dublin whose guidance and expertise has given me wealth of knowledge which I shall forever carry with me.

I would also like to acknowledge Dr. Donal E. O'Mahony of the School of Computer Science and Statistics at Trinity College Dublin as the second reader of this thesis, and I am gratefully indebted to his very valuable comments on this thesis.

Finally, I would like to express my extreme gratitude to my parents without whose support I would have never been able to be the man I am today. Who constantly encouraged and pushed me to be the best I can be and have been a source of unfailing support and encouragement throughout my years of study. This accomplishment would not have been possible without them. Thank you.

Salil Ajgaonkar

University of Dublin, Trinity College
August 2018

Proof Of Concept for Opportunistic Security in MPLS Networks

Salil Ajgaonkar, Master of Science in Computer Science

University of Dublin, Trinity College, 2018

Supervisor: Stephen Farrell

Multiprotocol Label Switching (MPLS) is a high performance packet switching technology which supports packet switching for multiple protocols and access technologies. It is a switching technology that redirects data based on short label paths rather than an IP table lookup. Each Label Switched Router in the MPLS network has a table that tells the router how to handle and redirect specific types of data. This helps the router to handle data in a consistent fashion while maintaining flexibility for traffic engineering at the same time.

MPLS provides networking corporations and government high speed and reliable data transfer over geographically dispersed sites while maintaining flexibility and high bandwidth data exchange. It is very easy to scale and can be efficiently tuned to meet various service level requirements. it is primarily used to provide Virtual Private Network (VPN) services for corporations who wish to transfer private data over a public network.

Such a heavy reliance on MPLS by big companies and the general public to manage their data means that any forms of network attacks often have the potential to disrupt vital everyday operations. Attacks ranging from data theft to denial of service if successful may lead to severe damages to the service provider as well as the owner of the data.

Earlier work conducted by the Internet Engineering Task Force (IETF) have analyzed the various vulnerabilities and security threats that affect the MPLS technology and have drafted a potential resolution to some of the security issues. This work proposed the implementation of Opportunistic Security in the MPLS network using payload encryption to encrypt the underlying application data and protocols headers using symmetric key encryption between end to end or hop by hop Label Switching Routers (LSRs) on an MPLS Label Switched Path (LSP).

This thesis describes the proof of concept for this proposed solution by implementing it on an experimental controlled environment and measuring its effects on the overall functionality and features of the presently running MPLS technology, the results of which should give an idea regarding the feasibility of the solution in practical commercial network in the world.

Summary

MPLS is a network routing technology which uses circuit switching and packet switched network technologies to support data routing for multiple protocols. It is a protocol independent and highly scalable technology whose flexibility enables efficient utilization of network resources resulting in high quality of service at low cost. MPLS relies on the use of fixed bytes of data that represent as labels for routing decisions. These labels are pushed on top of a data packet at the ingress (entering) switch of an MPLS network. All the succeeding Label Switching Routers (LSRs) in the MPLS cloud lookup these labels to appropriately route the data to the correct destination. The path to the destination is pre-configured in the LSR's label lookup table rather than an IP routing table. This avoids any complex lookups or reading long network addresses and implementation of longest match algorithms. Once the MPLS packet reaches the egress (exiting) switch of the MPLS network, the MPLS label is popped off and the data packet is then transmitted as a regular original packet along the succeeding switches. The LSRs between the ingress and the egress switch of the MPLS cloud also have the capability of pushing and popping off the MPLS labels on the data packet to efficiently engineer the network traffic.

Networking corporations and service providers leverage this flexibility to scale their network services. Using the advantages of MPLS they are able to run high bandwidth applications over seamless IP based networks that connect multiple, remote site [1]. The reliance on MPLS VPNs by corporations and government agencies means that at-

tacks ranging from intercepting sensitive data to disrupting data, voice and multimedia services can significantly impact vital operations [2].

Previously in MPLS, data security in the MPLS network mainly relied on just three features:
1) Physical isolation of MPLS networks that ensured interception of MPLS traffic was not possible.
2) Higher-layer protocol security such as IPsec which had been used whenever a particular flow has determined that security was desirable [1].
3) Layer 2 protocol security like MACSec where security was implemented on a hop by hop basis between devices connected on the Ethernet. However these features have a number of vulnerabilities. The network is still vulnerable to network taps between links, misconfiguration of routers, data replication, at the same time users might not enable to implement end to end security in their applications [1].

To mitigate these vulnerabilities to some extent, the IETF drafted an Internet-draft titled "Opportunistic Security in MPLS Networks draft-ietf-mpls-opportunistic-encrypt-03" which proposed a novel idea of implementing Opportunistic Security (OS) in the currently existing MPLS implementations. The Internet draft suggested the implementation of opportunistic encryption of data payload which is held by the MPLS packet using symmetric key encryption. As the LSR's only rely on the information contained in the MPLS labels for routing, the contents of the MPLS packet are of no significance for the flow of traffic. Thus end-to-end or hop-by-hop encryption of the data payload of the MPLS packet by the LSRs may contribute to maintaining the confidentiality, integrity and authenticity of the data that flows through the MPLS network.

This dissertation describes a way to implement this proposed experimental idea in a controlled environment by simulating the flow of traffic in an MPLS network in a virtual network and comparing the difference in performance between MPLS with the OS and MPLS without the OS. This will help validate or invalidate the proposed solution experimentally and further spread more light on its impact and feasibility on real world implementation.

This experiment is carried out using an experimental network setup consisting of mininet as a virtualized network platform. On this virtual platform OpenVSwitch (OVS) virtual switches are setup which simulate the flow of network traffic through the virtual switches. The MPLS flows needed to configure the OVS switches are configured using OpenFlow Software Defined Networking (SDN) Technology. OpenFlow configures these OVS switches by installing flow rules to simulate the behavior of an MPLS switch. The Opportunistic Security feature in the MPLS is implemented by coding the symmetric key functionality into the existing OVS code base which would then be configured onto the OVS switches in the mininet network to simulate the behavior of the system with the OS. We then evaluated the performance of the system by measuring certain network performance metrics and compared them with the ones measured without the OS. This comparative analysis may help us understand the performance level of the OS in MPLS against MPLS without OS, if there are any room for improvements and its potential impact on practical implementation.

Contents

List of Tables

List of Figures

Chapter 1

State Of The Art

1.1 Multi Protocol Label Switching (MPLS)

In order to better understand the MPLS functionality let us first understand how traditional IP routing works.

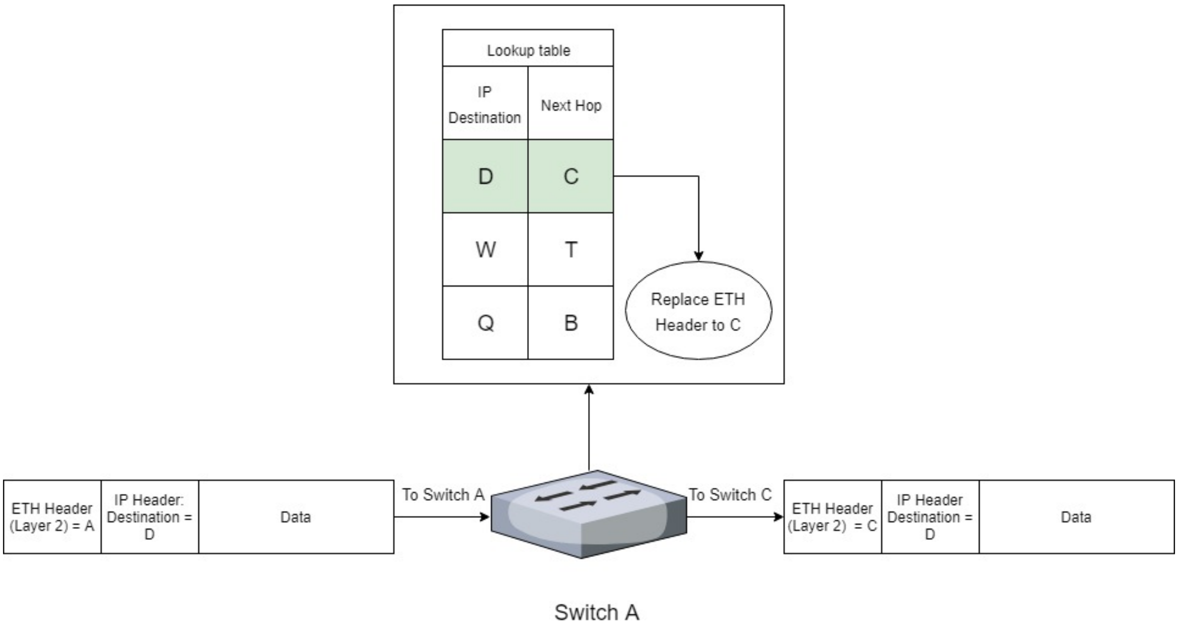
1.1.1 IP routing

IP routing is a type routing methodology that routes IP packets across an IP network. The switches[1] inside a network determine where to forward the IP packet based on the destination information stored in its IP header. The IP packet is forwarded from one switch to another in a hop-by-hop basis till it reaches its destination. Interior gateway protocols (IGPs) such as routing information protocol (RIP) and open shortest path first (OSPF), or exterior gateway protocol (EGPs) such as border gateway protocol (BGP) help route the IP packets from switch to switch [3]. Using these protocols the IP packets are routed through the network without any pre-determined path. All the IP packets with the same destination may flow through different paths to reach their destination. That is the reason why they call IP routing protocol as a connectionless protocol.

Routing protocols, such as Open Shortest Path First (OSPF), enable each router to learn the topology of the network. The routers build forwarding tables using the information provided by these routing protocols [3]. A switch references this forwarding

[1] 'Switch' in this dissertation indicates a Router capable of routing packets in the network

table when it analyzes an arriving packet to decide which is the next switch in the network that can bring the IP packet loser to its destination.



Lookup table

IP Destination	Next Hop
D	C
W	T
Q	B

Figure 1.1: IP Routing

The information regarding the IP packets destination is located in the Layer 3 header of the IP packet. The switch has to use this information to compare it with the lookup tables. The switch then replaces the Layer 2 header of the IP packet so it can be routed to the next switch as determined by the lookup. Figure 1.1 shows an IP packet with destination D arriving at switch A. Switch A analyses the IP header and finds the destination to be D. Switch A then refers to its lookup table to find the next hop for all packets destined to D. The next switch based on the IP lookup is C whose information is then stored into the ETH header (Layer 2) of the packet by switch A. Switch A then routes the packet to switch C which would follow the same procedure an eventually lead the IP packet to its destination D. All the switches follow this same process as the IP packet traverses through the network to reach its destination. The routing algorithms need not necessarily follow the shortest path logic to deliver the IP packet to its destination. Certain protocols also have the functionality to make decisions by considering the bandwidth occupied in between links to determine the best and fastest path to deliver packet.

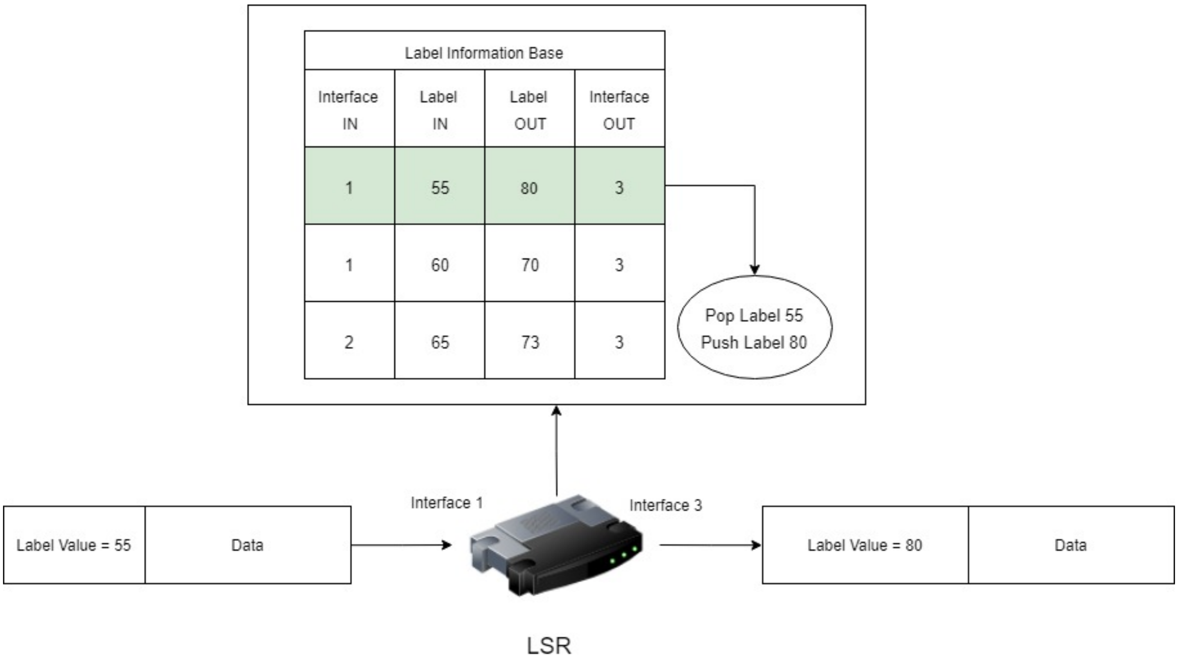
1.1.2 MPLS routing

MPLS Routing is a routing technology that leverages the benefits of both packet switching technologies as well as IP routing technologies. It is a data plane protocol most widely used in core network systems for high speed data transfer. MPLS is an IETF standard approach to integrate the best attributes of traditional layer 2 and layer 3 technologies. It defines a set of protocols and procedures that enable the fast switching capabilities of ATM and frame relay to be utilized by IP networks [3].

The main concept of MPLS routing is that instead of referring the IP header destination information for routing, the switches in the MPLS network have the functionality to push or pop certain additional routing information onto the data packets in the form of labels. The switches refer these labels to appropriately route the packets to the desired destination. The Label information mapping is based on existing IP routing technologies.

Switches having the functionality to push and pop labels together form an MPLS network. Routers in this network are specially called Label Switch Routers (LSRs). Routers at the edge of the MPLS network are responsible for communicating with the external traditional IP routers. These Routers are named as Label Edge Routers (LERs). The LERs are responsible for attaching the very first labels on top of the data packet that newly enters into the MPLS network. The label information is based on Forward Equivalency Class (FEC). Packets are then forwarded through the MPLS network, based on their associated FECs, through lable swapping by the LSRs in the core of the network, to their destination [3].

The mapping of Label values to their respective path is stored in the Label Information Base (LIB) of every LSR in the MPLS network. Each LSR builds it LIB when it is first initialized when the network is established. The LIB is similar to an IP lookup table with the difference being that instead of reading long network addresses the LSR refer these short label values which give a granular control over a packet's path in the MPLS network, which is aptly named a Label Switch Path (LSP). LIB is typically established in addition to the routing table and Forwarding Information Base (FIB) that traditional routers maintain [3].



Label Information Base			
Interface IN	Label IN	Label OUT	Interface OUT
1	55	80	3
1	60	70	3
2	65	73	3

Figure 1.2: MPLS Routing

Fig 1.2 shows a packet with a label value of 55 entering an LSR on its interface 1. The LSR then reads the label value and refers to its LIB to find out which LSP to forward this data packet. The LSR can also pop the Label value and insert a new value on the data packet if the LIB of the next LSR expects the new value. The next LSR will follow the same procedure which will eventually route the packet to the LER on the other end. This LER pops the last MPLS label forwarding the packet as it was originally onto the next traditional router which lies outside the MPLS network.

1.1.3 Components of MPLS

For MPLS to correctly route the data in its network it relies on certain special components. Some of which are as follows:

Forward Equivalency Class (FEC)

FEC is a class of packets which are routed along the same LSP in the MPLS network. A FEC can include all packets whose destination address matches a particular IP network prefix, or packets that belong to a particular application between a source and destination computer [3]. When a packet reaches an LSR, the LSR classifies the packets into an FEC based on its destination address, quality of service, the interface on which it arrived among other criteria. FECs are usually built through information learned through an IGP, such as OSPF or RIP [6].

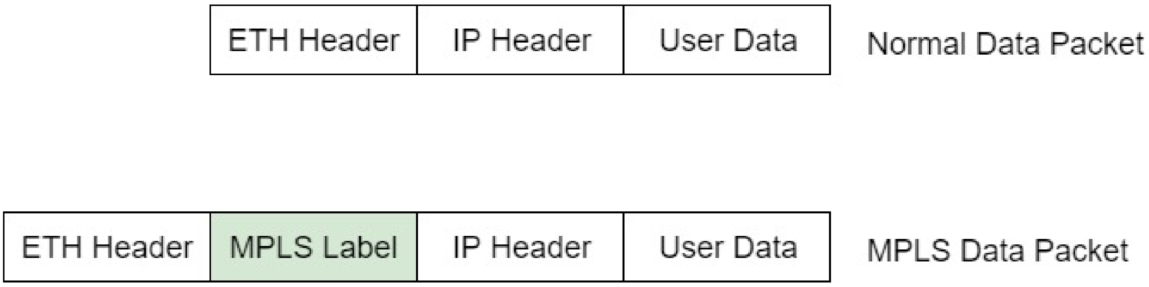


Figure 1.3: MPLS Label Comparison

Label Configuration

MPLS introduces its own header information termed as an MPLS Label or just Label. The MPLS label is a 32 bit (4 Bytes) fixed length, contiguous identifier used to denote the FEC of the packet. Just like the IP header in the IP routing mechanism, the MPLS header has all the information required to forward the data packet in the MPLS network. Figure 1.3 describes the structure of the MPLS Label. The MPLS label sits between the ETH header and the IP header. This is the reason why they say MPLS is a layer 2.5 protocol.

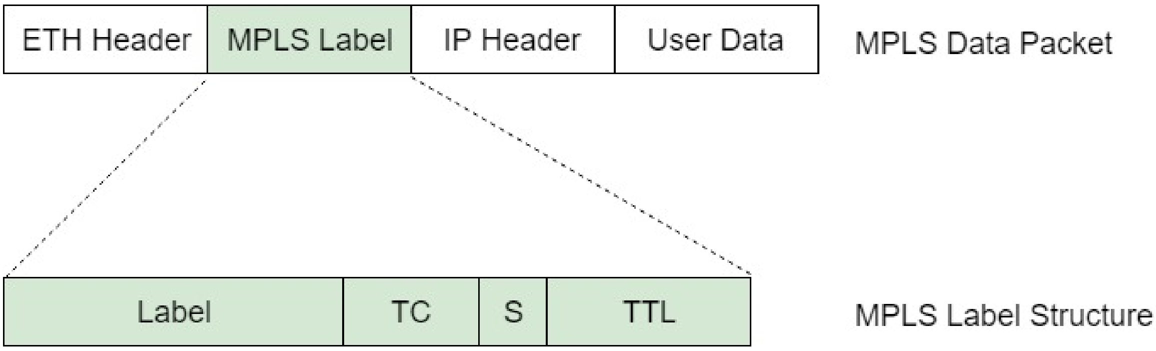


Figure 1.4: MPLS Label Structure

Figure 1.4 describes the MPLS Label structure. The MPLS Label consists of the following parts.

Label

The Label consists of an id that identifies the FEC of a particular Data packet. The LSR refer this value to determine the FEC of a data packet that arrives at its interface. This values determines the LSP that the data packet will take from the current LSR.

Traffic Class (TC)

The TC is primarily used to denote the Quality of Service implementation. This value along with the Label value can influence the LSR in selecting the LSP.

Bottom of the stack (S)

This is a simple flag that help identify whether the MPLS header is the last MPLS header in the MPLS stack. If the current label is the last one in the stack then the value is set to 1 else it is set to 0.

Time to Live (TTL)

This value denotes the number of hops the data packet has taken while traversing through the MPLS network. This value helps identify whether the packet was routed through the expected number of router hops or if it faced any errors while being routed through the network. The TTL helps prevent the packet from circulating in the network indefinitely. This value is copied from the packet header and copied back to the IP packet header when it emerges from the Label Switched Path [3].

MPLS by itself is vulnerable to a number of security threats. This is primarily because MPLS in its primary idea aims to solve the problem of high speed data delivery over large geographical network while maintaining flexibility so it can be scaled to meet business requirements. Routing and traffic engineering using MPLS is very easy and service providers often leverage this benefit to provide varying levels of Quality of Service.

With the increasing deployment of MPLS, security of data passing through the MPLS network has become a concern for corporations and the service provider alike.

Service providers are concerned with the security and confidentiality of their customerâĂŹs data while corporations using MPLS to share private data between their geographically distributed sites demand authentication and integrity as well.

1.2 Security Requirements in MPLS

Some of the General Service Provider Security Requirements are as follows:

Protection of data at the Data-Plane

Encryption of data is not provided as a basic feature in all telecommunication protocols. Protocols like IPsec have all the features of authentication, data integrity and confidentiality however it is not widely adopted by all applications. However,very few of web sites actually take on this burden of implementing IPsec, even though the software required is ubiquitous [2].

Protection from attacks on Label Distribution protocol

Label Distribution Protocol (LDP) is a protocol used by LSRs to exchange label mapping information with eachother. It used to build and maintain the LSP information in the databases of the participating LSRs. In [1] the authors have stated that attacks on Label Distribution protocol (LDP) exploit 3 weaknesses: The LDP specification, Service provider implementation and underlying infrastructure. The authors have expressed that these attacks can lead to various DOS attacks or Route modification attacks most of which may lead to violation of SLA for the Service provider. Naturally the Service provider would like protection against such attacks as a major requirement in implementing MPLS networks.

Prevent Malicious External Controllers from mis-configuring the SDN switches

SDN Switches are vulnerable to being mis-configured by external controller. An attacker can send malicious control-plane messages to the switches which can mis-configure them to send packets to a malicious switch, replicate the packets or drop the packets in general to trigger a Denial of service attack (DOS). Service providers expect some form of authentication of messages received from the control plane by the switches to make sure any control-plane messages received by the SDN are from a legitimate controller in the network.

Prevention of attacks that spoof IP addresses

Many attacks on protocols running in core involve spoofing a source IP address of a node in the core (e.g., TCP-RST attacks) [4]. If such a spoofed IP address gets accepted in the MPLS network, the MPLS switches can route sensitive packets to the spoofed address leading to data leakage. The attacker can then have the luxury to analyses and read the data at his leisure till the systems identifies the spoofed address.

Hiding Service Infrastructure

In general the service provider would like to hide his service infrastructure from the external network. An MPLS provider may make its infrastructure routers unreachable from outside users and unauthorized internal users. For example, separate address space may be used for the infrastructure loopbacks [4]. A service that is hidden to the external network has less chances of being targeted by attackers.

Protection from mis-merging of LSP

Care needs to be taken that any implementation of security procedures do not alter the the MPLS Label stacking logic which then becomes vulnerable to mis-merging of LSPs. LSP mis-merging has security implications beyond that of simply being a network defect. LSP mis-merging can happen due to a number of potential sources of failure, some of which are due to MPLS label stacking [5].

Link Authentication

Service providers would prefer to authenticate a site before linking a connection. This helps validate the site based on certain security protocols like IPsec. If the user wishes to hold the authentication credentials for access, then provider solutions require the flexibility for either direct authentication by the provider edge switch itself or interaction with a customer authentication server [4].

Security Considerations in Operations, Administration, and Maintenance messages

Operations, Administration, and Maintenance (OAM) messages are messages that are used for the internal functionality of the MPLS switches. OAM messages help in monitoring devices and implementing data transport mechanisms on a network level. They are responsible for the overall performance of the network device. On a service-oriented functionality they provide monitoring services to end users which is vital to keep a track of the performance so as to make sure the SLAs are met. The nature of OAM therefore suggests having some form of authentication, authorization, and encryption in place. This will prevent unauthorized access to MPLS-TP equipment and it will prevent third parties from learning about sensitive information about the transport network [6].

Meeting these requirements can easily fail if the system in place is vulnerable to any security threatening attacks. With the intention of securing these vulnerabilities we first need to analyze what are the different forms of attacks and how we can resolve them. A number of research has been conducted in analyzing the various security threats that affect the MPLS networks.

1.3 Security threats in MPLS

[1] has analysis a number of security threats to MPLS in VPN. In this work the authors have discussed the principal security issues in MPLS related to Network separation, Inter-provider connectivity and MPLS packet labeling. In [2] the authors have stated that security in MPLS previously relied only on the physical isolation of the MPLS network and High-layer protocol security like IPsec. The authors have also gone further as to why these features fall short of being the ideal solutions to current network attacks. In [4] the authors have elaborated on specific types of exploits that threaten the MPLS/GMPLS network. The authors have segregated the attacks in the form of attacks on the data plane, attacks on the control plane, attacks on the operation

and management plane and insider attacks. they have also recommended defensive techniques for MPLS against these forms of attacks. In [7] the author has suggested a mechanism to enhance the security in MPLS networks by using multi-path routing using a threshold secret sharing scheme. [8] has elaborated the security issues that are inherent in the MPLS architecture. The authors state that some routers may implement security procedures relying on certain headers being in fixed place relative to a certain layer header in the IP stack. Another security issue mentioned was that the MPLS routers agree on the meaning of the labels and thus work upon a chain of trust to transfer data in the network and that if packets from untrusted sources are accepted then they may get routed illegitimately.

In this section we will try to list out the various security attacks that the MPLS network is vulnerable to. The Security attacks have been categorized into 4 types: Attacks on the Data Plane, Attacks on the Control Plane, Attacks on Operational and Management of the MPLS Network and Insider attacks. MPLS OS is aimed at solving issues pertaining to threats at the data-plane level. We will hence not dwell on the security issues faced at the control-plane or Operational and management.

1.3.1 Attacks on the Data Plane

Attacks that are mainly aimed at the User's or the Service Provider's data are categorized into Data-plane security attacks. These attacks aim to either manipulate the data flowing through the MPLS network, delete the data flowing through the MPLS network, inject malicious data or just plain observe unauthorized data all with a malicious intent.

Plain IP forwarding

MPLS Switches can forward un-labeled IP packets as normal IP packets if they are configured to do so. An attacker already inside the core can exploit this information to reach other core devices compromising them. It is difficult to manage this type of attack by just traffic engineering or implementing any VPN service.

Forwarding captured packets from the core to the outside network

An attacker inside the core can capture the data sensitive packets and forward them to any destination he desires even if IP traffic is not being forwarded. This can be done by encapsulating the captured packet in the payload of a UDP packet. The source IP address can be spoofed and the destination IP address can be set to wherever the attacker may wish to forward the packet. This however is only possible if the attacker is already aware of the Labels needed to route through the LSP.

Unauthorized analysis of Packets

Packet analysis can be explained as the action of capturing data packets and analyzing its contents to understand what they contain. Private corporations and Users in general often transfer confidential and sensitive data over the internet. If this data is not sufficiently encrypted before forwarding them onto the internet then an attacker can use the wide variety of packet analyzing tools to read these data sensitive packets and use the sensitive information in them for malicious purposes. Unauthorized packet analysis can also be a first step in other attacks in which the recorded data is modified and re-inserted, or simply replayed later [4]. Unauthorized packet analysis is one of the most commonly faced security issue in any network system including and especially the MPLS network considering its wide spread use for mission critical systems and reliance of both common users as well as Multinational corporations.

One of the most recent issues with packet analysis comes from a new form of security threat termed as 'Upstream data collection'. Upstream data collection is a termed used by the National Security agency (NSA) of the United States of America to describe the act of intercepting the internet, phone and any other telecommunication device from the Internet backbone to analyze the data packets flowing through it. This has become a great concern since fibre optics operators can work with malicious government agencies to all allow interception and packet analysis for bribes [9]. The implementation of MPLS OS may help resolve this issue as encrypted packet even if intercepted may be hard to decipher.

Modification of Data Packets

If an attacker is able to manipulate the contents of the data passing through the MPLS network then he has the potential to initiate a wide range of security attacks on the network. Though not as easy as it sounds the attacker must first be aware of the internal configurations of the MPLS networks which more often than not is more difficult than the actual manipulation of packets. MPLS often has an "egg-shell" security model where it is very difficult to penetrate the internal network core, but once inside the attacker can cause a lot of damage to the service provider and the network service as a whole [1]. Some of the potential attacks that can be triggered by data manipulations are:

Route modification attacks

Once inside the network an attacker can manipulate the data flowing through the network and route them to any destination he desires, provided he has the information necessary to route the information correctly through the network. Route modification attacks enable an attacker to gain access to certain traffic (e.g., maneuver traffic through a compromised link); affect accounting (e.g., trigger automatic financial transactions among cooperating providers); or route traffic across domains (e.g., send one customerâĂŹs traffic to another customerâĂŹs network) [10].

1. Path Switching: A normal traffic flowing through the MPLS network ideally follows a fixed set path or to be more precise a Label Switched Path (LSP). The LSP is determined by the initial traffic engineering setup done by the service provider. The traffic engineering primarily relies on the label configuration of the packets. If an attacker were to modify this label information of a data packet, then he may be able to route the data packet that it was not intended to. Such a path is called a rogue path. By doing this the attacker can deceive the traffic engineering and reap benefits out of it. For example an attacker can forward his online gaming data packets via the live video conferencing path to get better speed advantage and disrupting the video conferencing quality of service.

2. Destination Switching: Just like Path switching, an attacker is also able to modify the destination of a packet if he modifies the label configurations appropriately.

This can fool the MPLS network into forwarding the packet to a rogue destination where the attacker can further conduct malicious operations on the packet which he wasn't able to do while inside the MPLS network.

3. Brute-Force Label Prediction: An attacker can deduce the LSP of an MPLS network if the destination address is known. He can target this address and pass data packets into the MPLS with an initial label value. He can then test out the Label with incremental values till he receives a reply from the destination address. The reply from the destination address can help him deduce the LSP for the destination address and thus reuse the label information to further pass data to the destination.

4. Brute Force Target Location: Similar to label prediction, the target can also try to identify what type of service lies at the end of an LSP, for example if the user were trying to identify if a web service lies at the end of an LSP then he can set the target TCP port to 80 and incrementally try out the IP address for a successful hit. This however is a very time consuming process considering the probabilities of getting the correct IP address compared to the total combination of IP addresses possible.

5. Forward Equivalency Class (FEC) Specificity Exploitation: When configuring the MPLS network Packets are configured in such a way that packets of similar type are routed through the same LSP. These packets are thus bound by the same MPLS label and routed through the same path as designated for labels of that value. Such classification of packets is termed as Forward Equivalency Class. This attack takes advantage of the âĂIJmost specificâĂİ or âĂIJlongest matchâĂİ rule applied by ingress routers to incoming IP packets. An attacker needs access to a link or a connection to an interface to establish an LDP session. The attacker identifies a target FEC and advertises label bindings for more specific FECs. LSRs that receive the label mappings distribute them throughout the network, thereby building new LSPs toward the compromised link [10].

6. Label Mapping Messages Modification: An attacker can modify the Label values of a data packet inside the MPLS network. He can thus reroute the traffic or create loops within the MPLS network. This modified message is sent on to the

next MPLS switch. When the upstream LSR receives a packet for the target FEC, it applies the incorrect label, which causes the downstream router to mistakenly recognize the packet as belonging to a different FEC. The packet is then forwarded along the desired LSP [10].

7. Address Messages Modification: Similar to label modification an attacker can redirect a data packet by spoofing the destination IP address. This attack, also known as Fabricating Address Messages, reroutes traffic or creates loops by manipulating the âĂIJleast costâĂİ mechanism used to select the next hop. Traffic can be redirected using access to a compromised link adjacent to an LSR along a selected LSP. This modified message forces the LSR to adjust its local label information base and generate a Label Request message. Thus, a new LSP is constructed that forces the targeted traffic along the compromised link [10].

8. Label Edge Router (LER) label Modification: Label Edge Router (LER) are routers situated at the end of an MPLS network. These routers are the final routers in the MPLS network that pop the final MPLS Label off the data packet and forward the data packet its original form before entering the MPLS network. This attack requires access to a link along the path between VPN sites. Redirection of packets to a different site in the same VPN requires the attacker to know the routes and labels corresponding to that site. The attack is executed by modifying the topmost label of a transit packet (before the penultimate pop) to another label. If the new label is valid at the next hop, the packet is forwarded to a different LER [1]. This new LER may or may not be connected to the destination. It could either drop the packet as a whole which could lead to denial of service if the attacker modified to many of the packets or the LER may forward the IP packet as a normal IP packet, depending on its configuration.

9. VPN label Modification: VPN label modification is similar to the LER Label Modification. In this type of attack two VPNs are involved. One is the legitimate source VPN and the other is the destination VPN which is incorrect. The attacker captures and modifies the datapacket label of the source VPN and redirects it to the incorrect VPN. When this modified data packet reaches the LER, the LER redirects it to the incorrect VPN. If the LER does not have a VRPN routing

and Forwarding (VRF) Table then the LER may forward the packet as a regular IP packet. Given knowledge of routes and VPN labels in the network core, the combination of VPN label modification with LER label modification (previous attack) enables an attacker to redirect and/or drop any traffic passing through the compromised link [1].

10. VRF table Modification: A more serious issue is if the attacker is able to modify the VRF tables themselves. This would grant the user the ability to control the traffic from the LER. The attacker can change outgoing LER labels and outgoing VPN labels to impact QoS. Moreover, the attacker can control the routes taken by ingress VPN traffic and divert it to the wrong VPNs [1].

Data Insertion attacks

Data insertion attacks are type of attacks which involve insertion of malicious traffic into the network with the intention of making a switch accept the external data as valid data and forward the same to end devices. Insertion attacks come in different forms with different intentions.

1. Insertion of malicious data traffic to Spoof and Replay: Spoofing refers to sending a user packets or inserting packets into a data stream that do not belong, with the objective of having them accepted by the recipient as legitimate [4]. Once a spoofed data is accepted an attacker constantly replays the accepted packets to incite the same response from the recipient even though the request is not authentic.

2. Insertion of malicious data using VPN Labels The attack is executed by fabricating packets labeled for the target VPN and corresponding egress LER, causing the egress LER to send the fabricated packets into the target VPN [1]. This type of attack is mostly implemented in MPLS networks providing VPN services. For this attack to be successful the attacker first needs to find a vulnerable LER which accepts external labeled packets. He should also be aware of the valid labels of the target VPN as well as the label information needed to route the packets to the target VPN.

3. Insertion of malicious data using LER Labels This attack is similar to the previous type of attack the only difference being that in this attack an attacker can ignore the insertion of the VPN label and just insert the malicious traffic with only the MPLS labels. When a packet with no VPN label reaches the egress LER, it is forwarded to a locally attached VPN site, or back through the network core to a remote VPN site, or to a core LSR. This 'bouncing' maneuver helps hide the source of an attack because packets appear to originate from the egress LER. An attacker still needs to have access to an LER that accepts external labeled data to perform this type of attack.

Denial-of-Service (DOS) Attacks

Denial of service is a type of attack where an attacker aims to make a target service unable to its users. This is carried out by disrupting the services of the service machine by bombarding it with large number of requests with the intention of overloading the system such that legitimate requests are not able to request for service. Banks, e-commerce websites and services which aim to provide high availability are particularly vulnerable to the damages caused by DOS attacks. There are plenty of ways an attacker can leverage the MPLS network to initiate a DOS attack on a service in the network.

1. Modifying the Community Attribute in LERs: In this attack the attacker modifies the VRF table that corresponds to the target VPN in an MPLS network that provides VPN services. The attack disrupts inbound and outbound traffic to and from the VPN site associated with the affected VRF table [1]. However to conduct this attack the attacker must first gain access to the LER which is connected to the target VPN.

2. Notification Messages Fabrication: An attacker who has access to a link in the MPLS network can fabricate false notification messages and release them in the MPLS network link. The attacker needs to have read and write access to this link. When an LSR receives this notification message it closes its LDP session thereby disconnecting the link. This prevents the flow of traffic through that LDP session and all packets destined to flow through that link are dropped, preventing requests from reaching the service. If an attacker has access to all the links of an

LSR then using the same form of attack he can close all the links of that LSR isolating it from the entire MPLS network.

3. Blocking Keep-Alive Messages: Similar to the previous attack another way an attacker can close LDP links is by selectively blocking the LDP keep-alive messages which are periodically sent by the LSRs to continue the link session. This causes the LSRs to time out and close the LDP session preventing any traffic from routing through the LDP. This attack is tricky in the sense that identifying the cause of timeout can distract the network engineer from finding out about the attack.

4. Address Withdraw Messages Fabrication In this type of attack an attacker target three LSRs in an LSP. LetâĂŹs assume the three LSR's involved in this example are A,B,C which are linearly connected in an LSP. The attacker fabricates an Address Withdraw message stating that address C has been withdrawn. He sends this packet to LSR A by gaining access to link A-B. LSR A withdraws the address associated with the interface for LSR C. LSR A now believes that LSR B cannot direct the traffic to C and thus forwards the traffic to C via other LSP congesting them. This congestion may eventually lead to a DOS attack.

5. Label Withdraw Messages Fabrication: This attack targets a specific LSP and requires access to a link along the target path. If the network employs label merging, then the attack also affects all upstream portions of paths merged with the target LSP [10]. The attacker fabricates a label Withdraw message and passes it along the LSP. The proceeding LSRs on receiving this message withdraw the label from their label information base. The LSP is thus destroyed and all traffic engineered to flow through this LSP now needs to be redirected along different LSP leading to congestion and eventual DOS.

6. Label Memory Exhaustion: This attack exploits LSRs with label retention mode. an attacker can flood such an LSR with Label mapping messages based on random FEC and label information. An LSR with Label retention mode will be forced to store these label information in its LIB which will eventually exhaust. The LSR is then forced to drop the older Label mappings to accommodate the new malicious Label mappings from the attacker which affects the legitimate mappings.

7. LSP Deletion. LSP deletion attack is an attack that involves the injection of malicious 'PathTear' messages from the Resource Reservation protocol that remove label bindings and resource reservations of targeted LSPs. A particularly insidious attack involves crafting a 'PathTear' message for a specific node in a targeted LSP; this message must be re-sent at least once per refresh period [11]. The receiving LSR de-allocates all resources and configurations of the LSP and stops routing packets through that LSP leading to packets drops and denial of service. This attack is able to target an LSP individually. This helps in targeting attacks to an individual target rather than disrupting the whole system to initiate a DOS attack. This helps prevent any warnings from arising in the network and the attack may go undetected [11].

1.3.2 Cross domain attacks

An MPLS Service provider has to allow a customer edge router to send data into the MPLS network which can be further routed in the MPLS network. An attacker can exploit this cross-domain interaction to initiate attacks on the MPLS network. [11] Describes two types of attacks that can happen and has termed them as cross-domain attacks.

1. Promiscuous Path Acceptance. Integrated Services (IntServ) [12] is a Quality of Service Architecture that incorporates a variety of signaling, admission, traffic management and scheduling protocols. A service provider implementing IntServ QoS in a traditional switched network typically uses Resource Reservation Protocol (RSVP) messaging. In such cases, a PE node in an adjacent MPLS network would accept packets with the Router Alert Option set as specified by the RSVP protocol [13]. Once these RSVP messages enter the MPLS network they are guaranteed to be forwarded. This attack is particularly dangerous as if an attacker is able to pass RSVP messages into the MPLS network then an attacker is basically able to perform any traffic engineering changes.

2. Pre-Labeled Traffic Acceptance. A Provide Edge node needs to be configured to accept traffic from a certain Customer Edge node that is authenticated. The PE node must use pre-platform scoping rules, otherwise the labels sent from the CE

node interface would not have bindings. Accepting pre-labeled traffic exposes an MPLS network to any number of signaling attacks from a compromised CE node [11].

1.4 Existing security tools

Given that the MPLS technology is implemented widely in the core networks, several Security tools already exists to make MPLS more secure. Each tool having its own set of features and complexities to make data transmission over the MPLS network more secure. MPLS mostly relies on external security protocols or frameworks for secure data transmission. A thorough analysis needs to be made regarding which tool to be used based on the operational requirements and feasibility of its implementation in the current scenario. Each of these tools provides a different mechanism to provide different security functions to the MPLS network. There is no one single tool that solves all the security issues plaguing the MPLS network, however proper consideration and analysis of the security requirements of the system and using the appropriate tool to mitigate them is more than enough to make MPLS secure for business requirements. Based on this idea it is best to have a suitably large number of security tools in the arsenal to fight off the possible security threats. Security tools range from a wide variety of mechanism to make MPLS secure. some of the widely used are described in the following.

1.4.1 Application Data Encryption

Encrypting application data by the application itself is the best form of security that any application can guarantee. It puts the responsibility of the security of the application data on the user and application developer itself. The security of the application data becomes independent of the security of the underlying data transport system. In this scenario the application developer has the freedom to choose any security tool that best suits for his application and requirements rather than relying on the inherent security of the underlying network system.

The issue with application data encryption is that not many developers take the necessary steps to secure their data. It is an additional cost to the development company to take the effort to implement appropriate security measures to secure their vital data as it is being passed onto the internet. Thus implementing opportunistic security in the underlying network will provide that additional safety net to prevent any security breaches just in case an application forgets to implement its own security measures.

1.4.2 Transport Layer Security (TLS)

The primary goal of TLS is to provide a secure channel between two communicating peers; the only requirement from the underlying transport is a reliable, in-order data stream [14]. One of the ways to make MPLS more secure is to encrypt the underlying data that is to be passed through the LSP. This prevents any interception attacks from analyzing any confidential data that a company or service provider may be transmitting via the MPLS network. Packet Encryption is ideally a responsibility of the application that is sending or receiving the data. Applications can use security tools like Transport Layer Security (TLS) which is an Internet standard to implement privacy and data integrity between communicating computer applications.

The issue with implementing TLS is solutions like TLS leave some metadata (such as the destination IP address) exposed as the packets transit the IP network [2]. An attacker can analyse this information and identify the source and destination IP of the packet. The attacker can thus analyse any of the route modification attacks as mentioned in the previous section leading to security breaches.

1.4.3 IP Security (IPsec)

IPsec is another tool which can be applied end-to-end or hop-by-hop applications to maintain privacy and data integrity. It is mainly used to encrypt IP packets that flow through the network. IPsec can be used to encrypt the IP packets before they are passed onto the MPLS network. It has historically placed a heavy "full-mesh" configuration burden on implementation although this is now ease with the introduction of the NULL Authentication Method in the Internet Key Exchange Protocol Version 2 allowing for opportunistic key exchange to support IPsec [1].

MPLS is a multi-protocol data forwarding technology. It's main strength lies in its ability to transfer data of different type of protocols via its MPLS network. IPsec may be a very good security measure to secure IP data in a network, however from an

MPLS implementation point of view it will only provide security if the data is of IP data type.

1.4.4 Link layer security (MACSec)

Moving down the IP stack, encryption is also possible in the Layer 2 (Link Layer). Packets can be encrypted on a hop by hop basis between two communicating routers using MACsec. MACsec encrypts Ethernet frames that transmit across ethernet network. Thus end to end security can be implemented by creating a chain of trust between all the participating routers in the network path.

The very fact that MACSec is implemented hop-by-hop in every switch makes it less ideal in an MPLS network scenario. Since it is hop-by-hop, encryption and decryption of the data packets happens in each and every switch. This can lead to higher latency in data packet transfers if there are a large number of switches in an LSP through which the data packet is flowing through.

As we see all these security tools are powerful sets to provide the required security in a given situation. Each tool has its own Pros and cons, some may need additional enhancements while most are self-sufficient in a given scenario. Most of the literatures covering MPLS security are also concerned with a certain use case in mind. Opportunistic Security in MPLS is not aimed to replace or displace any of the existing security tools currently present. It is only an addition in the arsenal of security tools aimed to make the internet more secure.

Chapter 2

Design

In this chapter we will aim to explain the overall design of the experiment to be conducted and the underlying functionality of the various technologies involved in implementing Opportunistic Security in the MPLS network. We will also discuss on the Architectural design of the system and the reason for choosing the technologies.

2.1 Opportunistic security (OS)

Historically, the approach to internet security has always been with an 'all-or-nothing' attitude. Security protocols aims at either providing the service with full security or a complete hard failure. This discouraged many service providers from implementing the latest security tools to make their network secure. Service providers have a responsibility to maintain their Service Level Agreement (SLA) and as such will often end up disabling these security tools and pass clear text through their network whenever the connection becomes too slow or customers have trouble connecting to services.

Authentication, Confidentiality and Integrity are the three pillars of network security. Confidentiality and Integrity can be brought about using encryption techniques with hashing mechanisms. Authentication on the other hand has to be carried out on a peer to peer basis. The ability to authenticate any potential peer on the Internet requires an authentication mechanism that encompasses all such peers. No IETF standard for authentication scales as needed and has been deployed widely enough to meet this requirement [15].

Browsers, web services and network devices refer to the Public Key Infrastructure (PKI) model to authenticate their peers or web services. Implementation of the PKI has its own additional costs and burdens and considering the large number of Certificate Authorities (CA) that provide this service on the internet, many service providers have trust issues with regards to the CAs. This may lead to disagreement between peers in the network that are trying to connect to each other.

Thus if authentication of nodes in a network becomes optional, where the system can run normally with plain text if authentication cannot be easily attained then the network system is more likely to implement Encryption in majority of its links. This way the approach to network security changes from 'security being default and anything less than that as degraded security' to 'No protection being default and anything more than that is an improvement'

"Opportunistic Security" (OS) is defined as the use of cleartext as the baseline communication security policy, with encryption and authentication negotiated and applied to the communication when available [15]. The aim of OS security is to implement encrypted and authenticated communication between peers whenever they are capable else only encrypted communication without authenticating the users or just clear text communication.

OS is not intended as a substitute for authenticated, encrypted communication when such communication is already mandated by policy (that is, by configuration or direct request of the application) or is otherwise required to access a particular resource. In essence, OS is employed when one might otherwise settle for cleartext .

2.1.1 OS in MPLS

As proposed in [2] the basic requirement in MPLS OS is to provide a way to encrypt data passing between two MPLS switches by doing a key exchange between them to create a session key using which the encryption can be carried out. The key exchange between the LSRs is to be carried out using the Diffie-Hellman key exchange. Using the key values agreed after the Key exchange we encrypt the flowing packets using Advanced Encryption Standard (AES) cryptographic algorithm. To enable authentication of the peers as well in the encryption it is suggested to use AES in Galois/Counter Mode (GCM).

Diffie-Hellman Key Exchange

Diffie-Hellman Key Exchange is a way of sharing secret information between 2 nodes over an insecure channel. It was developed by Martin Hellman, Whitfield Diffie and Ralph Merkle. Using this protocol the 2 nodes agree on a secret symmetric key which can be later used in encrypting the sensitive data that is to be communicated.

The Diffie-Hellman key exchange is modelled based on the Internet Key Exchange Protocol Version 2 as specified in the RFC [16].

Advanced Encryption Standard - Galois/Counter Mode (AES-GCM)

AES is a symmetric blockcipher algorithm established by the U.S. National Institute of Standards and Technology in 2001. It was developed by Vincent Rijmen and Joan Daemen who won the NIST conducted AES selection process. AES is able to handle block encryption of sizes 128, 192 and 256 bits. It is expected to provide data security for 20-30 years and is free to use for all devices.

GCM is an Authenticated Encryption with Associated Data (AEAD) mode which is a form of encryption that simultaneously provides data confidentiality, integrity and authenticity. It extends AES's encryption functionality to incorporate Authentication and Data Integrity as well. GCM is widely adopted because of its efficiency and parallel processing. Unlike cipher block chaining whose pipeline operations bottle neck its efficiency, GCM makes efficient use of its instruction pipeline to provide high speed operations using parallel processing.

The Data packet in the MPLS OS must be encrypted with AEAD-AES-GCM-128 encryption algorithm based on the specifications mentioned in the [16].

2.1.2 MPLS Packet Encryption

The MPLS packet is encrypted using AES-GCM encryption algorithm whose Key and Initialization Vector are determined by the initial Diffie-Hellman Key exchange carried out between the two participating end-to-end LSRs in the MPLS network. The encryption state of the MPLS packet is identified by the addition of a special purpose MPLS label called the MPLS Encryption Label (MEL). The bottom of the stack of the MEL is set to 1 and should be followed by a 4 byte Pseudowire control Word (CW).

As specified in the [16] the packet counter nonce used in the AES-GCM needs to be managed properly for successful Encryption and Decryption on the LSRs. The initial nonce value is derived from the HMAC-based Key Derivation Function (HKDF) output at key agreement time and the counter is incremented by one for each packet encrypted on the sending side and by one for each packet successfully decrypted on the receiver side [2].

The CW carries the low-order 16 bits of the nonce which helps the receiving LSR to identify any dropped packets or mismatch in the IV packet counter. The receiving LSR can then update the counter or resynchronize the counter to successfully decrypt the next incoming packets. The receiving LSR can raise alerts if more than 65536 packets are lost as the LSR will face a decryption failure, thus it is advisable to implement small window size to accept mismatched counters, beyond which the LSR will stop further decryption attempts to mitigate denial of service.

The AES-GCM generates a cipher packet which is slightly longer than the original packet. This is because of the additional authentication tag generated by the AES-GCM algorithm to provide authentication capabilities. The Receiving LSR will verify the authenticity of the encrypted packet as specified in [16] by referring this authentication tag.

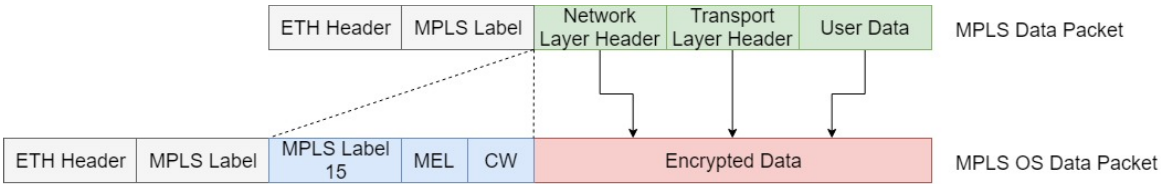


Figure 2.1: MPLS OS Packet Structure

Figure 2.1 shows the format of a normal MPLS packet with the MPLS with OS packet. In the normal MPLS packet an MPLS label with its bottom-of-stack (S) value 1 is pushed on top of the payload. Subsequent MPLS labels can be pushed over each other with values S = 0. Before transmission on the network the final Layer 2 header is pushed on top of the top MPLS label. In the MPLS OS data packet, the structure slightly changes. The Layer 2 header and the top MPLS label stays the same. The structure is then followed by the MPLS label with value 15. The value 15 informs the LSR that following this packet is a special purpose MPLS label. Then comes the MEL followed by the CW containing the low-order 16 bits of the nonce. The remainder of the packet is encrypted and contains the rest data of the packet.

MPLS Encryption Label (MEL)

The MPLS Encryption Label is a normal label stack entry carrying an extended special-purpose label with a value from the experimental range 240-255 [2]. The structure of the MEL is same as that of the traditional MPLS Label. The values of the contents of the label is what differentiates the MEL from the traditional MPLS labels.

Label

Considering this idea is still under experimentation no special purpose label value is yet assigned by the Internet Assigned Numbers Authority (IANA). The value however must be selected from the experimental range of 240-255

TC

The TC field should be set to the value of the unencrypted label stack entry directly preceding the the MEL else it should be set to 0.

S

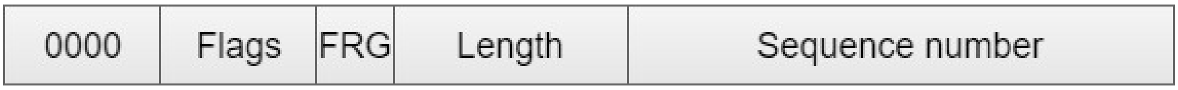
The MEL should always be the bottom of the MPLS stack and thus its value S should always be 1.

TTL

The TTL should ideally be set to 2 for end-to-end MPLS OS encryption to prevent the encrypted packets from being forwarded beyond the decrypting LSR.

Control Word (CW)

An LSR may tend to inspect the contents of an incoming packet to analyze its underlying protocol eg. check if it is IP and forward it via normal IP routing if so configured. The presence of the CW along with the MEL informs the receiving LSR that the contents of the MPLS packet is not of any standard protocol and thus cannot be inspected.



0000	Flags	FRG	Length	Sequence number

Figure 2.2: Pseudo-Wire Control Word Structure

Figure 2.2 shows the internal structure of the CW as specified in [2].

0000

The Value of the first 4 bits of the CW is set to 0000 to identify it as an MPLS Payload [17].

Flags

The Flags field is treated as a four-bit number. It contains the key-id that identifies the algorithm and key as established through configuration or dynamic key exchange [2].

Fragmentation FRG

FRG indicates Fragmentation, MPLS OS doesn't support fragmentation of the Data packets and as such the FRG should always be set to 0.

Length

Length is set to 0 and should be ignored by the receiving LSR

Sequence Number

This field contains the low-order 16 bits of the nonce that is currently being used for the agreed encryption parameters. It is indicative of the counter used in the AEAD-AES-GCM encryption which the receiving LSR can use to check it its counter is correct and if it can go ahead with the decryption.

2.2 Technologies Involved

Considering the experimental nature of this project a proper base architecture needs to be designed to cover all the requirements of the experiment. This architecture will form the blueprint on which the technologies involved will be implemented to mimic the behavior of the MPLS network with and without the OS as close to the real life behavior as possible.

The ideal controlled test bed for this project should involve a virtualized environmental system on which network routers could be set up. The building up and tearing down of the network should be quick and clean so as to prevent long loading times or incomplete setting changes in the network settings even before the experimentation could begin. The virtualized routers involved should have all the basic MPLS functionality required for proper implementation of the MPLS network. The Routers should also be able to process the data packets at the kernel level so as to get readings as close to real-life routers as possible without any latency added due to the implemented technology or userspace processing. Secondly the Router functionality should be configurable to implement the OS functionality into the switches.

Based on these requirements the following technologies were chosen to implement the MPLS OS experiment.

2.2.1 Mininet

Mininet is a technology that can create realistic virtual network on a single machine. It is able to run real kernel and application code to provide authentic network behavior in a virtualized form. It has support for various switches all set up in a Software Defined Network Architecture. Mininet can yield a more efficient use of time and resources compared to other workflows. It provides a local environment for network innovation that complements shared global infrastructure [18],

Mininet is used for a wide variety of cases such as optimization of topology designs, tutorials for various network technologies, demonstrations, Regression Suits and most importantly Prototyping [18]. Mininet has the capacity of rapid and simplified prototyping, the applicability safety, the possibility of sharing results and tools at zero cost [19].

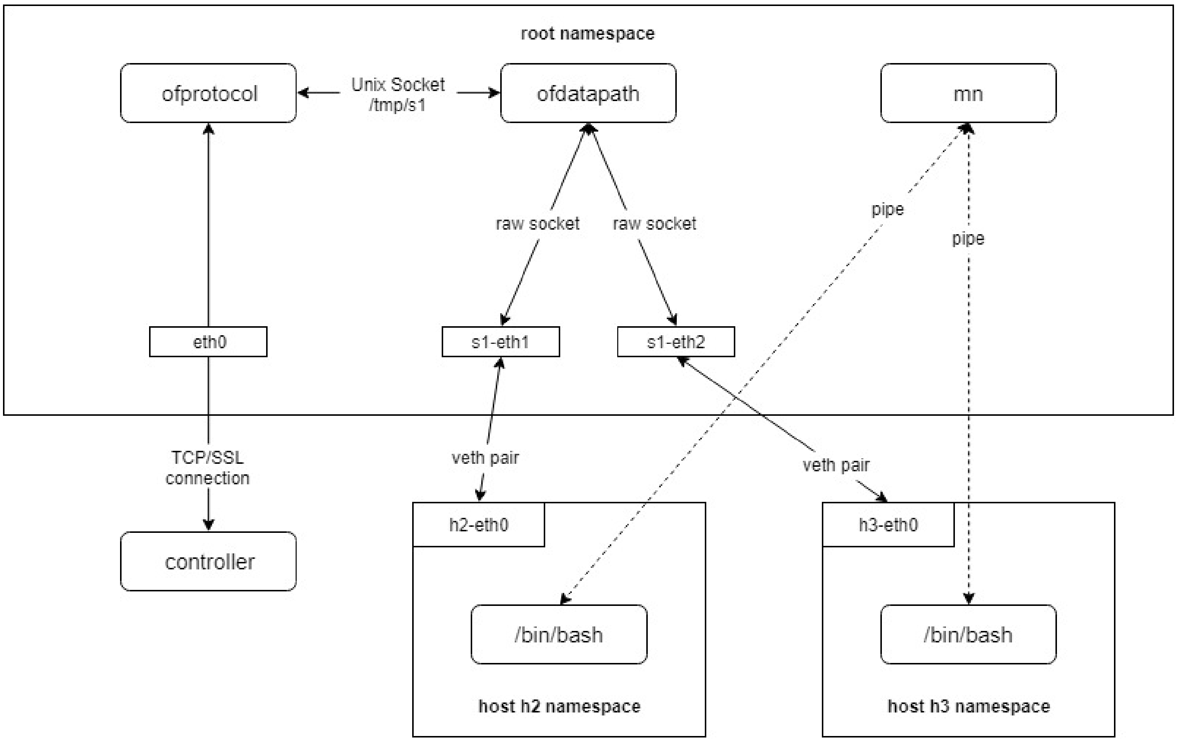


Figure 2.3: Mininet Emulation

Figure 2.3 [18] has described a basic architecture of how Mininet create a virtualized network with kernel functionality. Mininet creates a virtual network by placing host processes in network namespaces and connecting them with virtual Ethernet (veth) pairs.

2.2.2 OpenVSwitch (OVS)

OVS is a multi-layer Virtual Switch technology used to implement the functionalities of a hardware switch in a virtualized manner. It is designed for network automation on a large scale using programmatic extension, while still supporting standard management interfaces and protocols (e.g. NetFlow, sFlow, IPFIX, RSPAN, CLI, LACP, 802.1ag) [20].

OpenVSwitch works with most hypervisors and container systems, including Xen, KVM, and Docker. OpenVSwitch also works âĂIJout of the boxâĂİ on the FreeBSD and NetBSD operating systems and ports to the VMware ESXi and Microsoft Hyper-V hypervisors are underway [21]. It is designed to be flexible and portable and lies inside the hypervisor to provide connectivity between the virtual switch and the physical interface. Figure 2.4 describes the architecture of OVS.

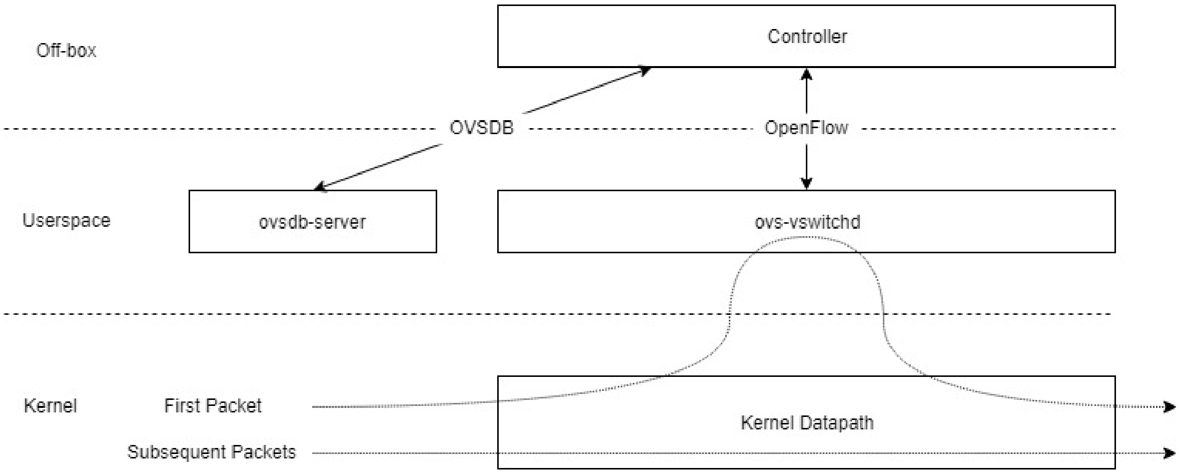


Figure 2.4: OVS Architecture

OpenVSwitch relies on 2 major components for forwarding packets in a network. The ovs-vswitchd which is a daemon that lies in the userspace of the system and the smaller datapath kernel module. The ovs-vswitchd daemon is different in different operating system environment however provides the same functionality, the data path kernel module on the other hand is written especially to function at the kernel level.

Figure 2.4 explains how the components of the OpenVSwitch interact with each other. A packet which arrives on the interface of a physical or virtual NIC is passed onto the datapath kernel module. The datapath kernel module will do either of the following things: Forward the packet based on the flow logic instructed by the ovs-vswitchd or pass it to the ovs-vswitchd to await instructions on what actions to take on such types of packets.

Kernel Forwarding

The ovs-vswitchd daemon instructs the kernel datapath module how to handles packets of specific types or ’flows’. These instructions are called ’actions’. The actions may specify a range of functionalities like modification of packets, sampling, packet dropping, re-routing etc. The kernel datapath simply follows the flow rules set by the ovs-vswitchd and executes the actions mapped to them. The kernel datapath can store these flow rules from the ovs-vswitchd daemon in its cache to act similarly on similar types of data packets. If an incoming packet does not match any of the flows stored in the kernel modules cache, i.e a cache miss, then the module forwards it to the userspace instead.

Userspace Forwarding

When the kernel module forwards a packet to the userspace due to a cache miss the ovs-vswitchd daemon determines what actions are to be taken on the data packet. The Daemon then forwards the action to the kernel Datapath to cache it and handle future similar packets.

OpenVSwitch is commonly used as an SDN switch and the main way to control forwarding is OpenFlow. It leverages the use of Open Flow protocol to add, update, and delete flows in a switch's flow table. The ovs-vswitchd daemon receives these flow rules from an SDN controller. It also has a great support for MPLS from version 2.4 onwards where you have kernel support for MPLS packets upto a maximum of 3 label stack.

2.2.3 OpenFlow

Networks have become a critical part of our day to day lives from business to research. Due to the large number of equipment and protocol installation overhead along with the reluctance of network owners to experiment with their production network so as to not affect ongoing services, Network researchers have a difficult time in experimenting with new protocols and network technologies. This is where OpenFlow comes in. OpenFlow provides an open protocol to program the flowtable in different switches and routers [22]. A network researcher is able to use OpenFlow protocol to control the flows in a flow table of all the switches inside the experimental network remotely and programmatically. This allows for easy and quick implementation of experimental setups which are ideal for research purposes.

An OpenFlow enables a switch to perform actions based on the experimental protocol. Using OpenFlow, a researcher experiments with his protocol by running it on a controller. The protocol will pick a route in the network of OpenFlow enabled switches and add the flow entry into all the switches in its path. The protocol can instruct the switches to encapsulate, drop, forward or push labels on top of packets like we do for MPLS LSRs. We can also set the flow rules to pick the type of data flows that pass through the Open flow enabled switch on which the actions are to be performed.

2.3 Architecture of Experiment

As Described in the previous chapter we will be implementing the MPLS OS system in a virtualized environment using Mininet, OpenVSwitch and OpenFlow technology. In order to simulate the MPLS OS structure and its effects on LSRs we need to design a network topology where we can analyze its overall effects in all use cases. Figure 2.5 Describes the topology of the network system we will be using to demonstrate the MPLS OS system.

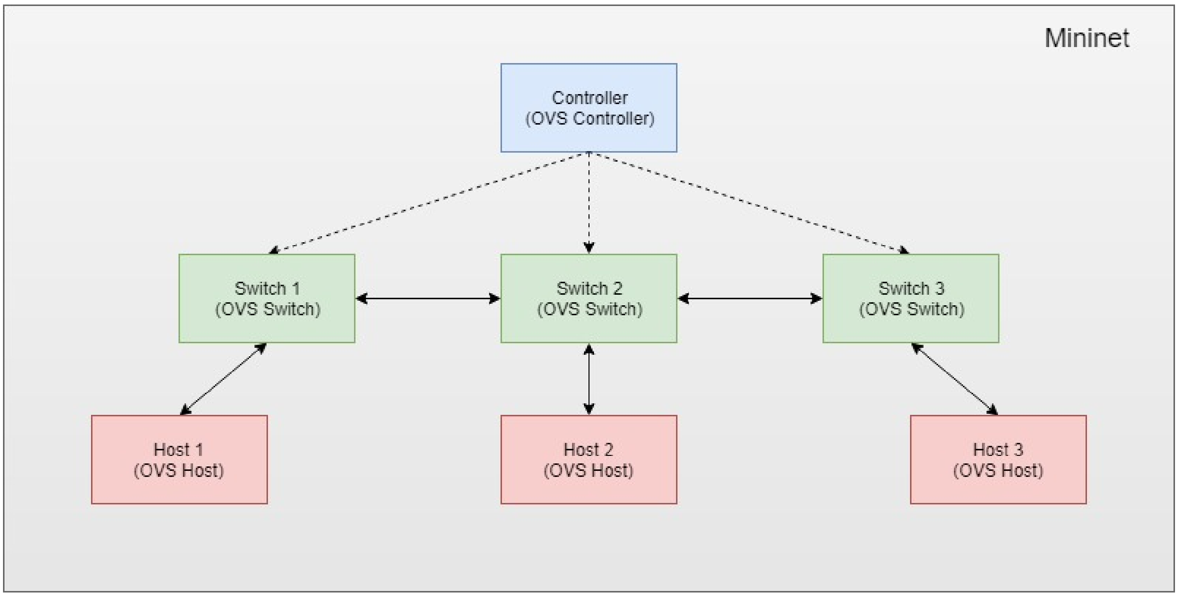


Figure 2.5: Network Topology for the Experiment

We will be using Mininet as our virtual network environment. The fast and clean network topology build up and tear down speed makes Mininet a good choice for setting up the virtual network environment. The Mininet version used in this experiment is version 2.2.1.

We will be deploying OpenVSwitch switches in the Mininet virtual network. Mininet has inbuilt support for setting up OpenVSwitches in its Virtual environment. The collaboration of both Mininet and OpenVSwitch makes both their use together in a system ideal for experimentation. The OpenVSwitch version used in this experiment is version 2.9.2

Flow rules and actions will be passed onto the OpenVSwitches using OpenFlow protocol. For this we need to setup an OVS controller which communicates the OpenFlow messages to all the OVS switches in the network. The OpenVSwitch Controller comes in built with Mininet and needs to be passed in the parameters for Mininet to setup in the network.

As shown in Figure 2.5 we will be setting up 3 switches (S1, S2, S3) in a linear topology. Each switch has its own host (H1, H2, H3) connected to it. Switch S1 has two interfaces s1-eth1 connected to host H1 and s1-eth2 connected to Switch S2 on its interface s2-eth2. Switch S2 has 3 interfaces s2-eth1 connected to host S2, s2-eth2 connected to Switch S1 on its s1-eth1 interface and s2-eth3 interface connected to Switch S3's s3-eth2 interface. Switch s3 has 2 interfaces, s3-eth1 connected to Host H3 and s3-eth2 connected to switch S2 on its s2-eth3 interface. Controller C0 will communicate the MPLS flow rules to all the three switches.

The Expected behavior of the system in normal MPLS process will be as follows:

1. Host 1 will ping an ICMP packet to Host 3

2. The packet will be first forwarded to Switch S1 at its s1-eth1 interface

3. S1, based on the flow rules that we will pass via the controller, will push an MPLS label on top of the packet and forward the packet along s1-eth2 interface.

4. Based on the linkage, S2 will receive the data packet. S2 will read the MPLS header value and forward it along its s2-eth3 interface.

5. Based on the linkage, S3 will receive the data packet. S3 will then pop the MPLS header and forward the normal ICMP packet to H3 via its s3-eth1 interface.

6. H3 will reply to the ICMP packet following the reverse path. the only difference being that now S3 will push the MPLS label and S1 will pop it at the end

The Expected behavior of the system in MPLS OS process will be as follows:

1. Host 1 will ping an ICMP packet to Host 3

2. The packet will be first forwarded to Switch S1 at its s1-eth1 interface

3. S1, based on the flow rules that we passed via the controller, will first Encrypt the payload using AES-GCM encryption algorithm the keys and IV of which, for the scope of this project, are currently hard coded. After encryption the CW with the low-order 16 bits of the nonce is pushed on top of the Data packet. Then the MEL is pushed on top. A normal MPLS with value 15 is then pushed on top of the MEL completing the MPLS OS data packet which is then forwarded onto the s1-eth2 interface

4. Based on the linkage, S2 will receive the OS data packet. On parsing the Data packet S2 should stop any further inspection or hash checking functions on the underlying encrypted data. S2 should just read the MPLS header value and forward it along its s2-eth3 interface.

5. Based on the linkage, S3 will receive the data packet. S3 will then pop the MPLS header with the value 15. It will then read the MEL value and pop it, expecting the next header to be the CW. S3 will the compare its low-order 16 bits of the nonce value with the one present in the CW. If it is a match then the counters are correct and s3 can proceed with the decryption. If not then S3 will update its counter to continue decrypting the next oncoming packets. Once successfully decrypted S3 will then forward the normal ICMP packet to H3 via its s3-eth1 interface.

6. H3 will reply to the ICMP packet following the reverse path with encryption on S3 and Decryption on S1.

The Diffie-Hellman Key exchange required to initialize the Symmetric Encryption Key and the Initialization vector for the AES-GCM encryption Algorithm is not implemented in this experiment and as such these values are currently hard coded into the system and can be later replaced with further development on that part. This experiment currently only demonstrates the encryption functionality of the OS and how the implementation of the OS Encryption affects the performance of the overall system.

Chapter 3

Experiment

In the previous chapter we described the design details that we will be using to implement the experiment. In this Chapter we go into the implementation details of the project.

3.1 Infrastructure setup

We will start of by first setting up the virtual network with the OVS Switches running the traditional MPLS System. To do this we first set up the mininet virtual network with the OpenVSwitch switches and hosts as described in Figure 2.5. Once the network system is in place we add the MPLS flows to the switches using OpenFlow. And finally we run a ping test to pass the ICMP packets through the MPLS switches to test the system.

3.1.1 Setting up Mininet and OpenVSwitch

The Mininet tool package is available in the Advanced Package Tool (APT) of all Linux Distributions. It can be easily installed on the system using the package name 'mn'. The Mininet package has a dependency on OVS packages and thus are installed along with the mininet package by the APT command.

The Network topology can be set up by calling the command:

sudo mn –topo=linear,3 –switch=ovsk –controller=ovsc

The parameter meanings are as follows:

1. –topo=linear,3
   This parameter creates a topology of 3 switches each with their own host connected in a linear fashion.

2. –switch=ovsk
   This parameter determines the type of switch that will be created in the mininet network. The value 'ovsk' indicates that OVS switches that run in kernel space are to be created

3. –controller=ovsc
   This parameter determines the type of controller that will be created in the mininet network. this controller is responsible for forwarding the flow rules and actions to all the switches in the network. the value 'ovsc' indicates that an OVS controller is to be created to communicate with the OVS switches.

3.2 Testing the MPLS System

3.2.1 Addition of the MPLS flows

Once the network infrastructure is set up we then add the MPLS Flows to the switches via the OVS controller. This is done passing the flow rules and actions to the Openflow Switch Manager (ovs-ofctl).

The OpenFlow Switch management command structure is as follows:

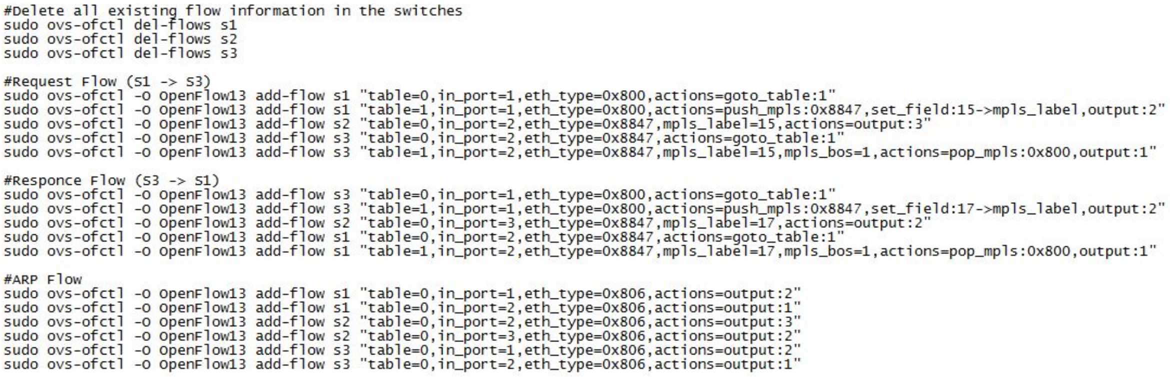
ovs-ofctl -O OpenFlow13 add-flow s1 "table=0,in_port=1,eth_type=0x800,actions=goto_table:1"

1. -O OpenFlow13
   Indicates the OpenFlow Version to use. The value 'OpenFlow13' indicates OpenFlow Version 1.3.

2. add-flow s1
   Indicates the action to be taken on the switches. We can add flows, delete flows modify flows etc using the OpenFlow Protocol. The value 's1' indicates the switch on which the flow rules are to be added.

3. "table=0,in_port=1,eth_type=0x800,actions=goto_table:1"
   Is the flow rule where the flow is to be added in table '0' of the Switch, to packets arriving at port '1' which is mapped to interface s1-eth1, which are of eth type 0x800 (ICMP Packets). The 'actions' value indicates the type of actions to take on such packets.

The Complete Flow configuration for MPLS is as follows:



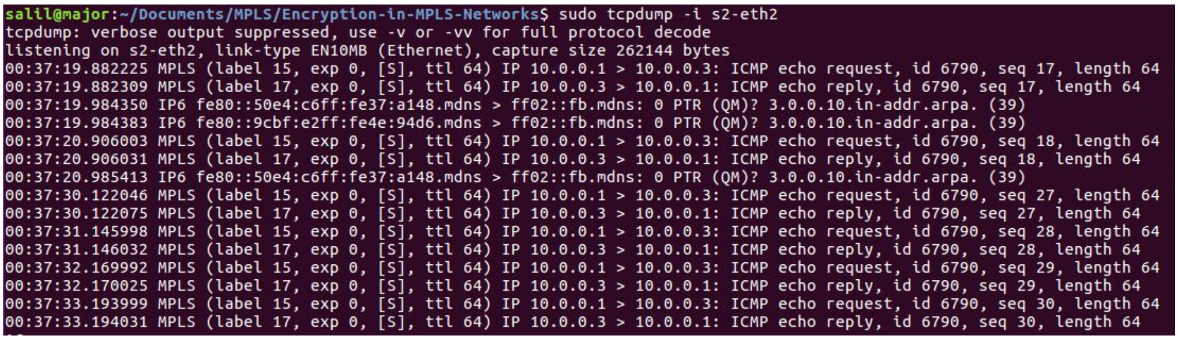
```
#Delete all existing flow information in the switches
sudo ovs-ofctl del-flows s1
sudo ovs-ofctl del-flows s2
sudo ovs-ofctl del-flows s3

#Request Flow (S1 -> S3)
sudo ovs-ofctl -O OpenFlow13 add-flow s1 "table=0,in_port=1,eth_type=0x800,actions=goto_table:1"
sudo ovs-ofctl -O OpenFlow13 add-flow s1 "table=1,in_port=1,eth_type=0x800,actions=push_mpls:0x8847,set_field:15->mpls_label,output:2"
sudo ovs-ofctl -O OpenFlow13 add-flow s2 "table=0,in_port=2,eth_type=0x8847,mpls_label=15,actions=output:3"
sudo ovs-ofctl -O OpenFlow13 add-flow s3 "table=0,in_port=2,eth_type=0x8847,actions=goto_table:1"
sudo ovs-ofctl -O OpenFlow13 add-flow s3 "table=1,in_port=2,eth_type=0x8847,mpls_label=15,mpls_bos=1,actions=pop_mpls:0x800,output:1"

#Responce Flow (S3 -> S1)
sudo ovs-ofctl -O OpenFlow13 add-flow s3 "table=0,in_port=1,eth_type=0x800,actions=goto_table:1"
sudo ovs-ofctl -O OpenFlow13 add-flow s3 "table=1,in_port=1,eth_type=0x800,actions=push_mpls:0x8847,set_field:17->mpls_label,output:2"
sudo ovs-ofctl -O OpenFlow13 add-flow s2 "table=0,in_port=3,eth_type=0x8847,mpls_label=17,actions=output:2"
sudo ovs-ofctl -O OpenFlow13 add-flow s1 "table=0,in_port=2,eth_type=0x8847,actions=goto_table:1"
sudo ovs-ofctl -O OpenFlow13 add-flow s1 "table=1,in_port=2,eth_type=0x8847,mpls_label=17,mpls_bos=1,actions=pop_mpls:0x800,output:1"

#ARP Flow
sudo ovs-ofctl -O OpenFlow13 add-flow s1 "table=0,in_port=1,eth_type=0x806,actions=output:2"
sudo ovs-ofctl -O OpenFlow13 add-flow s1 "table=0,in_port=2,eth_type=0x806,actions=output:1"
sudo ovs-ofctl -O OpenFlow13 add-flow s2 "table=0,in_port=2,eth_type=0x806,actions=output:3"
sudo ovs-ofctl -O OpenFlow13 add-flow s2 "table=0,in_port=3,eth_type=0x806,actions=output:2"
sudo ovs-ofctl -O OpenFlow13 add-flow s3 "table=0,in_port=1,eth_type=0x806,actions=output:2"
sudo ovs-ofctl -O OpenFlow13 add-flow s3 "table=0,in_port=2,eth_type=0x806,actions=output:1"
```

Figure 3.1: OpenFlow Flow Scripts for traditional MPLS

3.2.2 Testing Communication for MPLS

After adding the Flow Rules we can initiate the ping test from h1 to h3 by calling the ping command in the Mininet CLI. This would create an ICMP packet at Host H1 which would be forwarded to Switch S1. Switch S1 based on the Flow Rules as described in Figure 3.1 will forward the packet to its table '1' where it will push an MPLS label of eth value 0x8847, set its label value to 15 and forward it to switch S2. Switch S2 will redirect all packets of eth type 0x8847 and MPLS Label value 15 to Switch S3. S3 based on its flow rules will first forward the data packet to its table 1 where it will check if the MPLS label has value 15 and is a bottom of the stack label, pop the MPLS label if it is and later forward the packet to Host H3. H3 then responds to the ICMP request by sending the ICMP response to switch S3 after which the data packets is subjected to the flow rules as described in the 'Response Flow' part of the diagram.

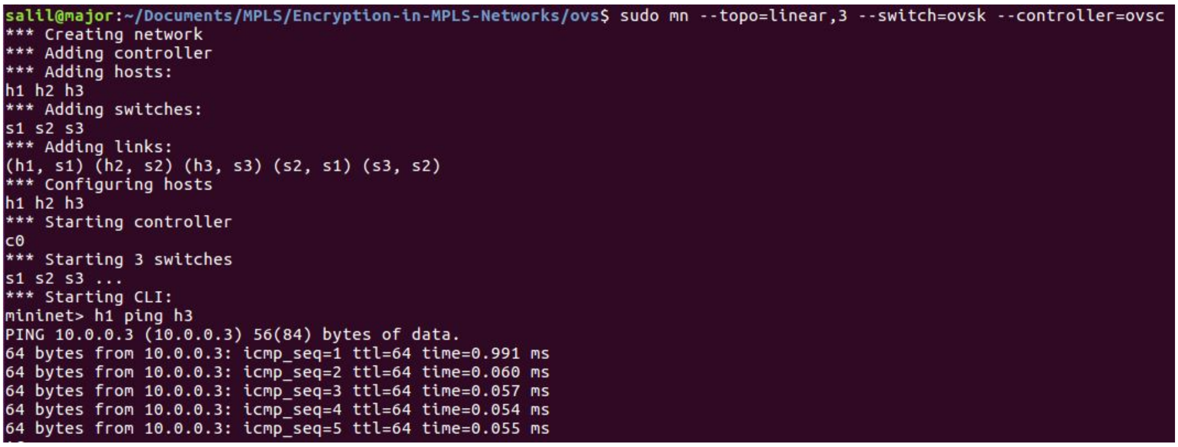
To test if the packets are being routed correctly using MPLS labels we can inspect the packets at switch 2. This is done by calling the command 'tcpdump' on one of the interfaces of switch 2. Refer diagram



```
salil@major:~/Documents/MPLS/Encryption-in-MPLS-Networks$ sudo tcpdump -i s2-eth2
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on s2-eth2, link-type EN10MB (Ethernet), capture size 262144 bytes
00:37:19.882225 MPLS (label 15, exp 0, [S], ttl 64) IP 10.0.0.1 > 10.0.0.3: ICMP echo request, id 6790, seq 17, length 64
00:37:19.882309 MPLS (label 17, exp 0, [S], ttl 64) IP 10.0.0.3 > 10.0.0.1: ICMP echo reply, id 6790, seq 17, length 64
00:37:19.984350 IP6 fe80::50e4:c6ff:fe37:a148.mdns > ff02::fb.mdns: 0 PTR (QM)? 3.0.0.10.in-addr.arpa. (39)
00:37:19.984383 IP6 fe80::9cbf:e2ff:fe4e:94d6.mdns > ff02::fb.mdns: 0 PTR (QM)? 3.0.0.10.in-addr.arpa. (39)
00:37:20.906003 MPLS (label 15, exp 0, [S], ttl 64) IP 10.0.0.1 > 10.0.0.3: ICMP echo request, id 6790, seq 18, length 64
00:37:20.906031 MPLS (label 17, exp 0, [S], ttl 64) IP 10.0.0.3 > 10.0.0.1: ICMP echo reply, id 6790, seq 18, length 64
00:37:20.985413 IP6 fe80::50e4:c6ff:fe37:a148.mdns > ff02::fb.mdns: 0 PTR (QM)? 3.0.0.10.in-addr.arpa. (39)
00:37:30.122046 MPLS (label 15, exp 0, [S], ttl 64) IP 10.0.0.1 > 10.0.0.3: ICMP echo request, id 6790, seq 27, length 64
00:37:30.122075 MPLS (label 17, exp 0, [S], ttl 64) IP 10.0.0.3 > 10.0.0.1: ICMP echo reply, id 6790, seq 27, length 64
00:37:31.145998 MPLS (label 15, exp 0, [S], ttl 64) IP 10.0.0.1 > 10.0.0.3: ICMP echo request, id 6790, seq 28, length 64
00:37:31.146032 MPLS (label 17, exp 0, [S], ttl 64) IP 10.0.0.3 > 10.0.0.1: ICMP echo reply, id 6790, seq 28, length 64
00:37:32.169992 MPLS (label 15, exp 0, [S], ttl 64) IP 10.0.0.1 > 10.0.0.3: ICMP echo request, id 6790, seq 29, length 64
00:37:32.170025 MPLS (label 17, exp 0, [S], ttl 64) IP 10.0.0.3 > 10.0.0.1: ICMP echo reply, id 6790, seq 29, length 64
00:37:33.193999 MPLS (label 15, exp 0, [S], ttl 64) IP 10.0.0.1 > 10.0.0.3: ICMP echo request, id 6790, seq 30, length 64
00:37:33.194031 MPLS (label 17, exp 0, [S], ttl 64) IP 10.0.0.3 > 10.0.0.1: ICMP echo reply, id 6790, seq 30, length 64
```

Figure 3.2: 'tcpdump' response for MPLS network traffic

The output displayed by the ping command in diagram indicates that the switches are pushing, inspecting and popping the MPLS labels correctly without any issues as shown in Figure 3.3.



```
salil@major:~/Documents/MPLS/Encryption-in-MPLS-Networks/ovs$ sudo mn --topo=linear,3 --switch=ovsk --controller=ovsc
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (s2, s1) (s3, s2)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
mininet> h1 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=0.991 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.060 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.057 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.054 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=0.055 ms
```

Figure 3.3: 'ping' response for MPLS Network Traffic

3.3 Implementation of OS Encryption functionality

The next part of the experiment involves the implementation of the OS Encryption functionality. Based on the OS designed as mentioned in chapter 2, the four additional functionality involved in the MPLS OS is the addition and removal of the CW and the Encryption and Decryption of the MPLS Payload. These functionalities should be performed by the OVS Switch involved in the network experiment and as such should be implemented in the OVS code base.

After a thorough study of the OpenVSwitch codebase, we found out that there is no such inbuilt provision in the OVS codebase for the required Os functionality. We can neither implement the CW functionality nor the Encryption/Decryption of the MPLS payload with the currently existing features of OVS. The functionality was thus needed to be implemented manually into the OVS codebase and then converted into a kernel module for the Switches to implement the same at the kernel level.

For the sake of development classification we will segregate the OS functionality into 2 categories:

3.3.1 Changes at the Ingress (Entering) Switch

When the ICMP request packet from H1 enters S1, before pushing the MPLS header on the data packet, S1 first encrypts the payload data using AES-GCM encryption algorithm. The key and IV required for the encryption is currently hardcoded in the code as their values are dependent on the Key Exchange which is not implemented in this project. The AEAD-AES-GCM encryption is carried out using the Linux Kernel Crypto API[1]. After encryption the pseudo wire CW is pushed on top of the encrypted data followed by pushing the MEL on top of it.

Due to certain implementation aspects of the Linux Kernel Crypto AEAD encryption API, instead of encrypting the Payload data first and then adding the CW we will first add the CW and then encrypt the underlying Payload. The details of this aspect will be discussed further in this section.

[1] Kernel Crypto API is a cryptographic framework in the Linux Kernel for various kernel implementations dealing with cryptography

Insertion of CW

We first create the CW data structure in the OVS codebase and assign the nonce value to the CW sequence attribute.

In OVS datapath and Linux kernel in general, all data packets are parsed into a fundamental data structure called a socket buffer (skb)[2]. The skb is a series of contiguous memory location storing the data packet information in the form of bytes. The structure of the skb is shown in Figure 3.4.

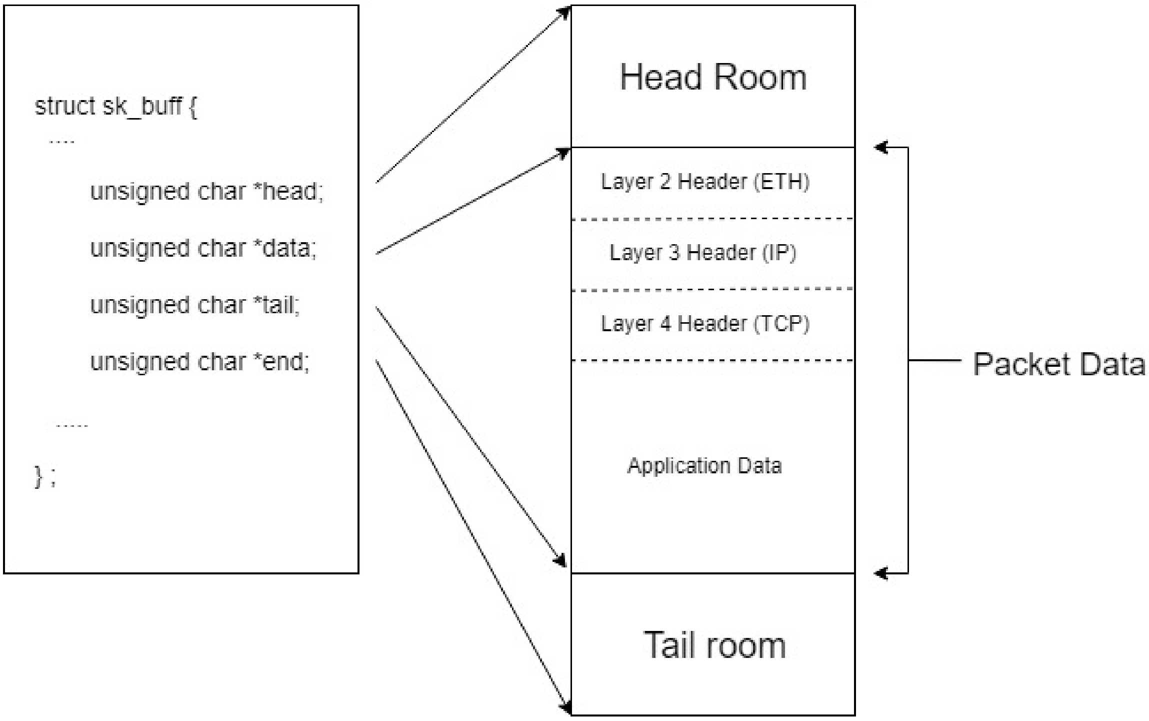


Figure 3.4: SKB Data Structure

Any changes that involve manipulating the data packet have to be done by manipulating the data bits of the skb data structure. A number of data pointers help map the various data and header locations in the skb structure. The linux kernel has in-built APIs to manipulate these contents. It is highly recommended to use these API's as the various pointer in the skb data structure are heavily interdependent on each other. Us-

[2] The socket buffer, or "SKB", is the most fundamental data structure in the Linux networking code. Every packet sent or received is handled using this data structure.

ing these APIs we manipulate the data packet and insert our CW between the payload and the MAC header of the datapacket. This is done by calling the following series of Kernel APIs. Refer Table 3.1 which depicts the implementation of the skb kernel APIs to insert the CW into the data packet.

Sr.no	Command	Function
1	skb_cow_head	Checks if space, the size of the CW header, is available in the headroom for inserting the CW.
2	skb_push	Allocates the space for the Data Structure.
3	memmove	Moves the MAC header to the newly allocated space so that the CW can be inserted between the MAC header and the underlying payload.
4	skb_reset_mac_header	Resets the MAC header pointer to the new MAC header position.
5	skb_mac_header(skb) + skb->mac_len	Points to the start of the CW memory space where the CW data bits can be filled.

Table 3.1: Steps to insert CW in Data packet

Encryption of Data using Linux Crypto API

The Linux Kernel Crypto API is a rich set of cryptographic ciphers and data transformation API used for cryptographic operations at the kernel level. To understand and properly use the Crypto API functions one needs to understand their underlying architecture specifications and the functional flow of these APIs.

The Kernel Crypto API refers to all encryption algorithms as ’transformations’. The encryption is carried out by 2 major components, The transformation object, also called ’Cipher Handles’ and the request handle. The transformation object contains all the settings and configurations of the given encryption type to use. The request handler as the name suggest handles the encryption request.

Sr.no	Command	Function
1	Initialize 'crypto_aead' and 'aead_request'	Initialize the AEAD transformation Object and its Request Handler
2	crypto_alloc_aead	Allocate the AEAD Cipher Handle
3	aead_request_alloc	Allocate the AEAD Request Handle
4	aead_request_set_callback	Set the Asynchronous callback function that will be executed once the encryption is successfully completed
5	crypto_aead_setkey	Set the encryption key in the cipher handle
6	aead_request_set_crypt	Set the data buffers where the encryption process will happen. The data that is to be encrypted is concatenated with the allocated data at the start and then passed as whole as plain text
7	aead_request_set_ad	Set the associated data information that will be used to generate the authentication tag for authentication of data at the receiving end
8	crypto_aead_encrypt	Start the encryption process

Table 3.2: Steps to Encrypt the Data Payload

Table 3.2 describes the flow of API calls to initiate the encryption of the packet data.

Once the payload data has been encrypted S1 can then further push the MEL label on top of the packet followed by the Special Purpose Label Packet and another normal MPLS packet for routing.

3.3.2 Changes at the Egress (Exiting) Switch

Once the encrypted MPLS OS packet reaches the Egress Switch S3, S3 pops the top MPLS label along with the MEL. It then inspects nonce value in the CW header. If the nonce counter value matches with the counter value of switch S3 then the switch moves ahead with the decryption.

Decryption of Data using Linux Crypto API

Decryption of the MPLS payload follows the same procedure as the one during Encryption process with a slight change in the input values to the API's. During decryption, the cipher text is passed in the place of plain text into the API's. The cipher text is a concatenated combination of the associated data, ciphertext and authentication tag. Also in the crypto_aead_encrypt function the flag is set to decrypt. The decryption process authenticates the packet using the authentication tag. The process can send out an alert if the authentication tag fails to authenticate. The output of the decryption gives us a concatenated combination of the associated data and the plain text.

Removal of CW

Just as how we used the in-built SKB manipulation APIs to insert the CW in the Ingress Switch, we will use the same approach while removing the CW from the Data packet. Refer Table 3.3 which depicts the implementation of the skb kernel APIs to remove the CW from the data packet.

Sr.no	Command	Function
1	skb_postpull_rcsum	Recalculate the checksum after pulling the MEL from the data packet
2	memmove	Moves the MAC header to occupy the place where the CW resides
3	__skb_pull	Pulls the CW size equivalent of data from the top of the data structure which was redundant as the MAC header shifted down.
4	skb_reset_mac_header	Resets the MAC header pointer to the new MAC header position.
5	skb_set_network_header	Sets the network header pointer to the new network header position.

Table 3.3: Steps to remove the CW from the Data packet

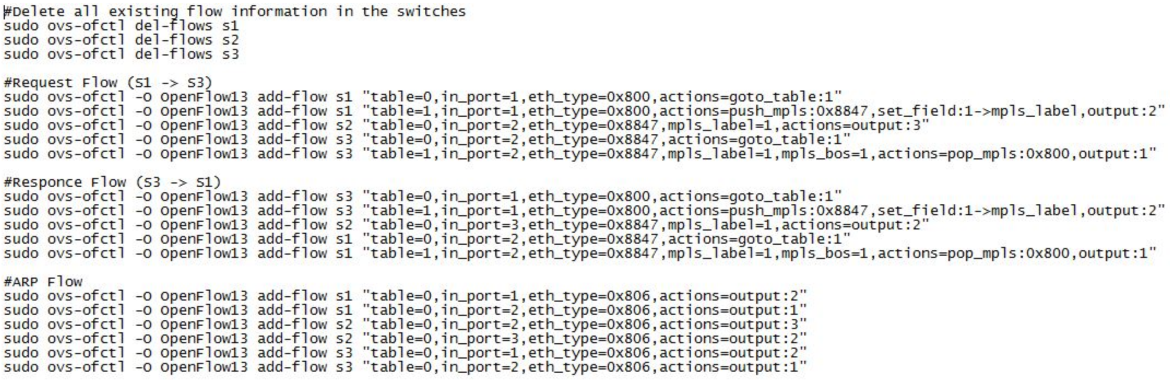
3.4 Testing the MPLS OS System

Once the OS functionality has been implemented in the OVS code-base we need to deploy it into out experimental system for testing. This is done by first building the code base to create its binaries. The OVS code has a Make files that has all the file building sequences scripted into it. Using the Linux 'make' command you can compile the system. Once the System has compiled successfully you call the Linux function 'make install' to install all the binaries into the appropriate locations in the system. We then remove the currently loaded OVS module by calling 'rmmod openvswitch' function so we can load the new module containing the MPLS OS functionality. We then run the 'make modules_ install' function to build the kernel module libraries and install them the the Kernel modules directory. Finally we load the newly installed openvswitch kernel module by calling 'modprobe openvswitch'.

3.4.1 Addition of the MPLS OS flows

The MPLS Flows for OS are similar to the ones mentioned for traditional MPLS. The only difference being the addition of the Special Purpose Label with value 15, the MEL and the CW. However the addition of these headers is not a part of the OpenFlow Flow. The addition of these headers depends on the success of Key Exchange between the Ingress and Egress LSR's before the OS can begin and thus the LSRs must be programmed to take that decision programmatically. Since we are not implementing the Key exchange in this experiment and have hard coded the key for experimental purposes. We are allowed to instruct the LSRs to force push the MEL of value 1 followed by the Special purpose MPLS Label with value 15 and another MPLS Label for routing purposes. Addition of CW is programmed internally in the OS functionality. During Experimentation however, it was observed that the Egress switch would drop these packets. The MPLS OS packets would route through switch S2 but S3 would drop these packets before forwarding to Host H3. Analysis into the issue led to the discovery of issues in OVS with respect to contiguous popping off of 2 or more Labels (MPLS and even VLAN). It was observed that at the kernel level OVS is only able to push and pop one Label. 2 or more contiguous labeled packets are routed to the Userspace for efficient handling. However for some reason the Userspace fails to do so. This issue is apparently prevalent in OVS and concerns were been raised back in 2016. However no information was found regarding its solution. More details can be found in this mailing list [23] [24] As a result a work-around was implemented which would not affect the overall aim of the project. Instead of pushing the MEL, the special purpose MPLS label with value 15 and the normal MPLS Label, we decided to push just one MPLS Label of label value 1. This Label will act as a functional combination of all the

3 labels in our experiment and would not interfere with the OS Encryption as a whole. So based on this work-around the Flow configuration for MPLS with OS is as follows:



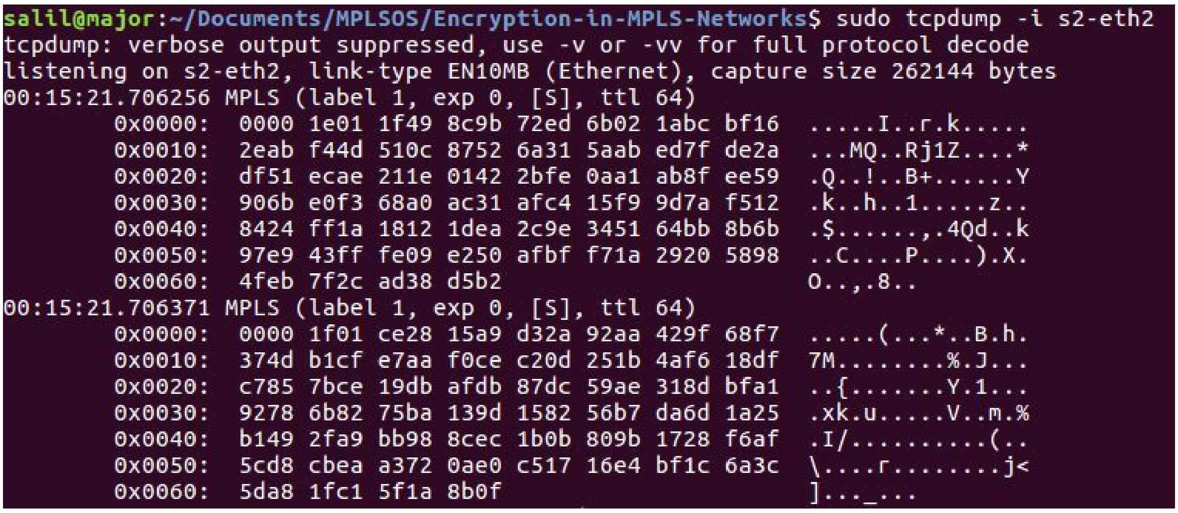
```
#Delete all existing flow information in the switches
sudo ovs-ofctl del-flows s1
sudo ovs-ofctl del-flows s2
sudo ovs-ofctl del-flows s3

#Request Flow (S1 -> S3)
sudo ovs-ofctl -O OpenFlow13 add-flow s1 "table=0,in_port=1,eth_type=0x800,actions=goto_table:1"
sudo ovs-ofctl -O OpenFlow13 add-flow s1 "table=1,in_port=1,eth_type=0x800,actions=push_mpls:0x8847,set_field:1->mpls_label,output:2"
sudo ovs-ofctl -O OpenFlow13 add-flow s2 "table=0,in_port=2,eth_type=0x8847,mpls_label=1,actions=output:3"
sudo ovs-ofctl -O OpenFlow13 add-flow s3 "table=0,in_port=2,eth_type=0x8847,actions=goto_table:1"
sudo ovs-ofctl -O OpenFlow13 add-flow s3 "table=1,in_port=2,eth_type=0x8847,mpls_label=1,mpls_bos=1,actions=pop_mpls:0x800,output:1"

#Responce Flow (S3 -> S1)
sudo ovs-ofctl -O OpenFlow13 add-flow s3 "table=0,in_port=1,eth_type=0x800,actions=goto_table:1"
sudo ovs-ofctl -O OpenFlow13 add-flow s3 "table=1,in_port=1,eth_type=0x800,actions=push_mpls:0x8847,set_field:1->mpls_label,output:2"
sudo ovs-ofctl -O OpenFlow13 add-flow s2 "table=0,in_port=3,eth_type=0x8847,mpls_label=1,actions=output:2"
sudo ovs-ofctl -O OpenFlow13 add-flow s1 "table=0,in_port=2,eth_type=0x8847,actions=goto_table:1"
sudo ovs-ofctl -O OpenFlow13 add-flow s1 "table=1,in_port=2,eth_type=0x8847,mpls_label=1,mpls_bos=1,actions=pop_mpls:0x800,output:1"

#ARP Flow
sudo ovs-ofctl -O OpenFlow13 add-flow s1 "table=0,in_port=1,eth_type=0x806,actions=output:2"
sudo ovs-ofctl -O OpenFlow13 add-flow s1 "table=0,in_port=2,eth_type=0x806,actions=output:1"
sudo ovs-ofctl -O OpenFlow13 add-flow s2 "table=0,in_port=2,eth_type=0x806,actions=output:3"
sudo ovs-ofctl -O OpenFlow13 add-flow s2 "table=0,in_port=3,eth_type=0x806,actions=output:2"
sudo ovs-ofctl -O OpenFlow13 add-flow s3 "table=0,in_port=1,eth_type=0x806,actions=output:2"
sudo ovs-ofctl -O OpenFlow13 add-flow s3 "table=0,in_port=2,eth_type=0x806,actions=output:1"
```

Figure 3.5: OpenFlow Flow Scripts for MPLS with OS

3.4.2 Testing Communication for MPLS with OS

The behavior of the MPLS System with the OS is similar to the behavior without the OS. The only difference to note is that in the MPLS OS Switch S1 will be encrypting the data packet and inserting the CW along with the MPLS Label with value 1 before forwarding it to Switch S2. S2 must not be able to sniff the internal data of the encrypted packet and must be able to forward it only based on the MPLS Label value. Once the MPLS OS packet reaches Switch S3, S3 should decrypt the packet, remove the CW and pop the MPLS Label before forwarding the original ICMP packet to host H3. Host H3 will respond to the packet with the response ICMP packet which follows the reverse path with encryption on S3 and decryption on S1.

To test if the packets are being encrypted correctly using the MPLS OS, we can inspect the packets at one of S2's interface. This is done by calling the command 'tcpdump' on one of the interfaces of switch 2. Refer diagram

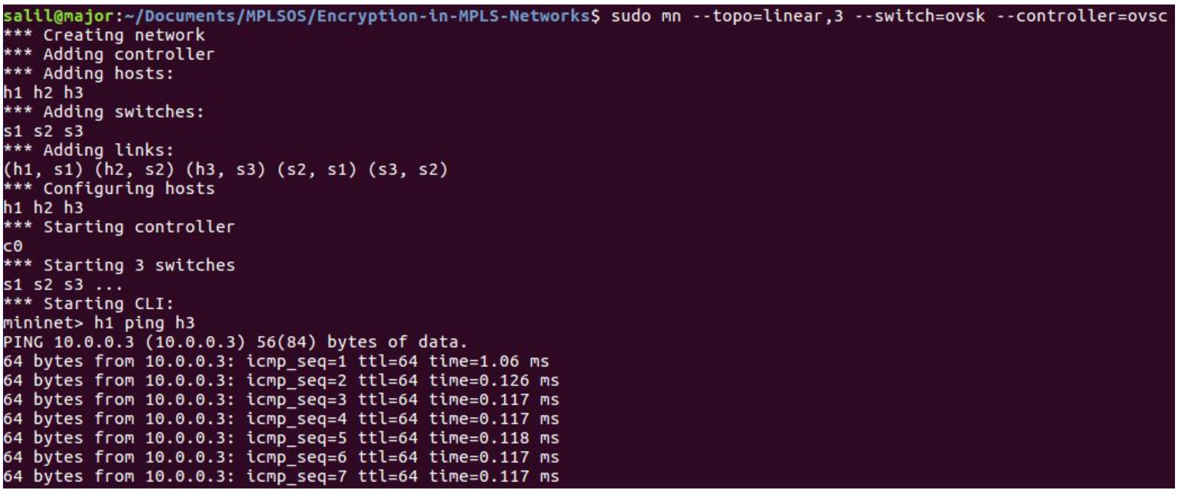


```
salil@major:~/Documents/MPLSOS/Encryption-in-MPLS-Networks$ sudo tcpdump -i s2-eth2
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on s2-eth2, link-type EN10MB (Ethernet), capture size 262144 bytes
00:15:21.706256 MPLS (label 1, exp 0, [S], ttl 64)
        0x0000:  0000 1e01 1f49 8c9b 72ed 6b02 1abc bf16  .....I..r.k.....
        0x0010:  2eab f44d 510c 8752 6a31 5aab ed7f de2a  ...MQ..Rj1Z....*
        0x0020:  df51 ecae 211e 0142 2bfe 0aa1 ab8f ee59  .Q..!..B+......Y
        0x0030:  906b e0f3 68a0 ac31 afc4 15f9 9d7a f512  .k..h..1.....z..
        0x0040:  8424 ff1a 1812 1dea 2c9e 3451 64bb 8b6b  .$......,.4Qd..k
        0x0050:  97e9 43ff fe09 e250 afbf f71a 2920 5898  ..C....P....).X.
        0x0060:  4feb 7f2c ad38 d5b2                      O..,.8..
00:15:21.706371 MPLS (label 1, exp 0, [S], ttl 64)
        0x0000:  0000 1f01 ce28 15a9 d32a 92aa 429f 68f7  .....(...*..B.h.
        0x0010:  374d b1cf e7aa f0ce c20d 251b 4af6 18df  7M........%.J...
        0x0020:  c785 7bce 19db afdb 87dc 59ae 318d bfa1  ..{.......Y.1...
        0x0030:  9278 6b82 75ba 139d 1582 56b7 da6d 1a25  .xk.u.....V..m.%
        0x0040:  b149 2fa9 bb98 8cec 1b0b 809b 1728 f6af  .I/..........(..
        0x0050:  5cd8 cbea a372 0ae0 c517 16e4 bf1c 6a3c  \....r........j<
        0x0060:  5da8 1fc1 5f1a 8b0f                      ]..._...
```

Figure 3.6: 'tcpdump' response for MPLS OS

As we can see in Figure 3.6 tcpdump tries to sniff the contents of the data packets and is unable to interpret its contents. The only understandable part being the MPLS header with value 1. The rest of the contents is dumped as it is in hexadecimal byte form. S2 cannot interpret that the contents of the IP packet are infact ICMP as we saw during the communication testing of traditional MPLS system in section 3.2.2.

The output displayed by the ping command in Figure 3.7 displays the proper response being received for each ICMP request packet. This means that the receiving switch is able to correctly decrypt the packet at its end and get the original ICMP packet as was sent by the sender.



```
salil@major:~/Documents/MPLSOS/Encryption-in-MPLS-Networks$ sudo mn --topo=linear,3 --switch=ovsk --controller=ovsc
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (s2, s1) (s3, s2)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
mininet> h1 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=1.06 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.126 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.117 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.117 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=0.118 ms
64 bytes from 10.0.0.3: icmp_seq=6 ttl=64 time=0.117 ms
64 bytes from 10.0.0.3: icmp_seq=7 ttl=64 time=0.117 ms
```

Figure 3.7: 'ping' response for MPLS OS

Chapter 4

Results and Evaluation

To evaluate the effect of OS on the overall performance of the network, finding out the correct metrics to measure the performance is very important. The ideal metric are those that correctly reflect the inherent complexity involved in performing the OS Encryption. There are plenty of metrics used to measure the performance of a network. However not all of them are relevant in terms of what we aim to achieve. For example number of packets dropped is a good metric to test out new network protocols, however in our scenario the number of packet drops is not relevant since that attribute is not affected by the implementation of OS Encryption. The MPLS OS switch may drop all the MPLS OS packets if it fails to encrypt or decrypt the packets, however that is a case where the protocol failed as a whole and not a performance issue. Such metrics will be ignored in the evaluation of this project. The metrics that we find are relevant in our scenario are explained in the following sections.

4.1 Latency

Latency is a term used to define the time delay between data communication over a network. Packets flowing through a network take time to reach their destination. The difference between the time a packet is sent from the sender and the time it reaches the receiver is termed as delay. In network communication delay is measured in terms of Round Trip Time (RTT). Due to the difference between the clock time of two nodes in a network it is difficult to measure the exact time a packet take to reach from point A to point B. In network the time it takes for a sender to send a node to a receiver and the receiver to send an acknowledgement back to the user is termed as a round trip time. In this case the user keeps track of the time it sent the data and the time it received an acknowledgement of the data. For our experiment we measured the effects of OS on the latency of the network system by comparing it to the one without the OS. The Encryption decryption and addition of special purpose labels is bound to introduce some additional computation time and as such would affect the time required to send and receive the packets. We measured the latency of the system by measuring the Round Trip Time (RTT) of 1000 ICMP ping requests and responses and compare its average, RTT times. It is expected that the RTT for MPLS OS will be more than the traditional MPLS routing. Refer Figure 4.1 for the obtained results.

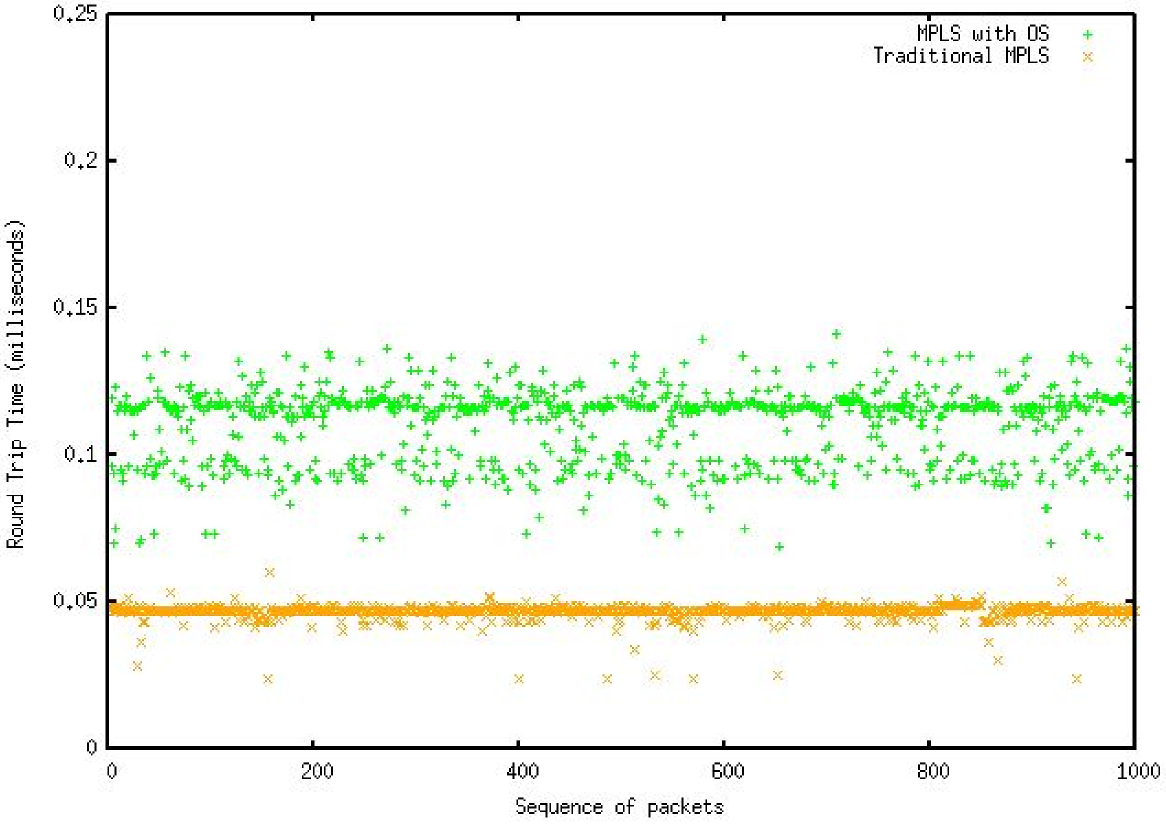


Figure 4.1: Round Trip Time of Traditional MPLS and MPLS with OS

As stated in Figure 4.1 we can see that the average RTT for MPLS OS is infact slightly more than the average RTT of the traditional MPLS network. IT was also observer that a small number of packets cluster together at a RTT slightly less that 0.1 millisecond and deviate from the majority of the MPLS with OS packets in the graph. It is difficult to find out why a small number of packets face the same lower RTT than majority of the packet. Further study may be needed to find out. A difference of about 0.07 milliseconds was observed between the RTT of MPLS with OS packets against traditional MPLS packets. Though the difference is not that significant we can confirm that the Encryption and Decryption process does in fact add more computational time to the overall time required for data transfer.

4.2 Effects of Size of Packets

Based from our results in the previous section we can assume that the amount of RTT taken by a packet may be proportional to the amount of data that needs to be encrypted and decrypted. More the data more time required to encrypt and decrypt it. It would be interesting to measure how the increase in packet size might affect the time taken to encrypt and decrypt and thus affect the RTT.

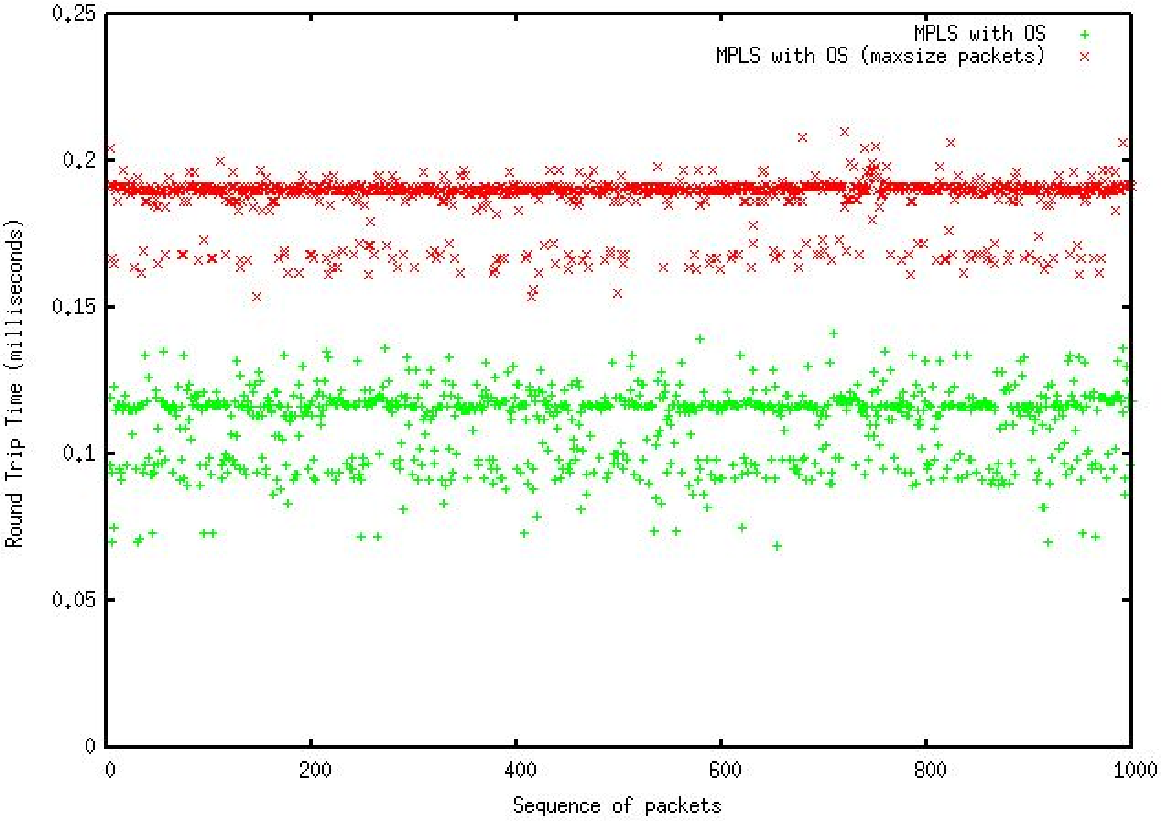


Figure 4.2: Round Trip Time of MPLS OS with 56 byte ping packet vs MPLS with OS with Maximum Packet Size 1500 bytes

In Figure 4.2 we see that the latency of MPLS OS increases by an additional 0.7 milliseconds approx if the packet size was increased to to the Maximum Transfer Unit MTU as compared to a ping packet of default size of 56 bytes plus 20 bytes of MPLS OS headers (MEL, CW, additional Cipher text). If we compare the MPLS OS results with traditional MPLS packet with the packet size reaching the MTU we can see that the MPLS with max packet size still has lesser latency than MPLS with OS with a default 56 byte size packets. Refer figure 4.3 for the details.

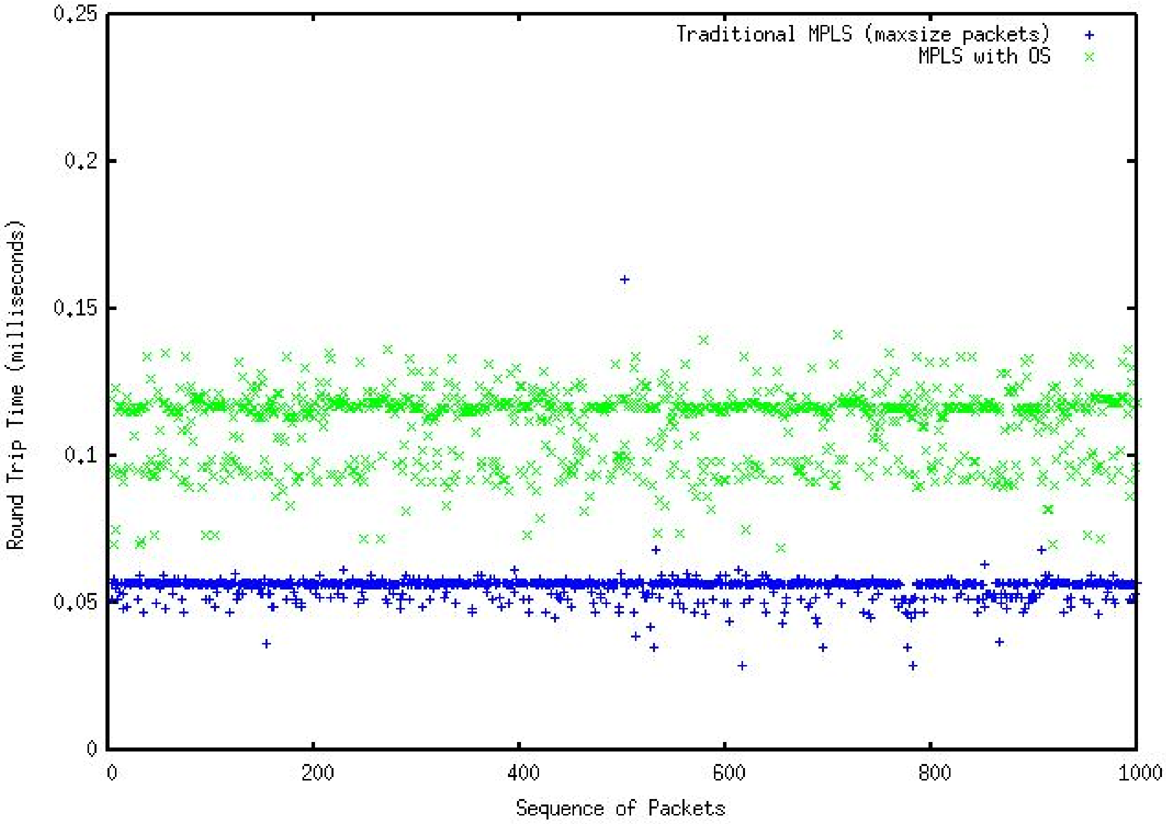


Figure 4.3: Round Trip Time of Traditional MPLS with max size packets vs MPLS with OS with regular size packets

Figure 4.4 summarises the average latency in all the 4 systems.

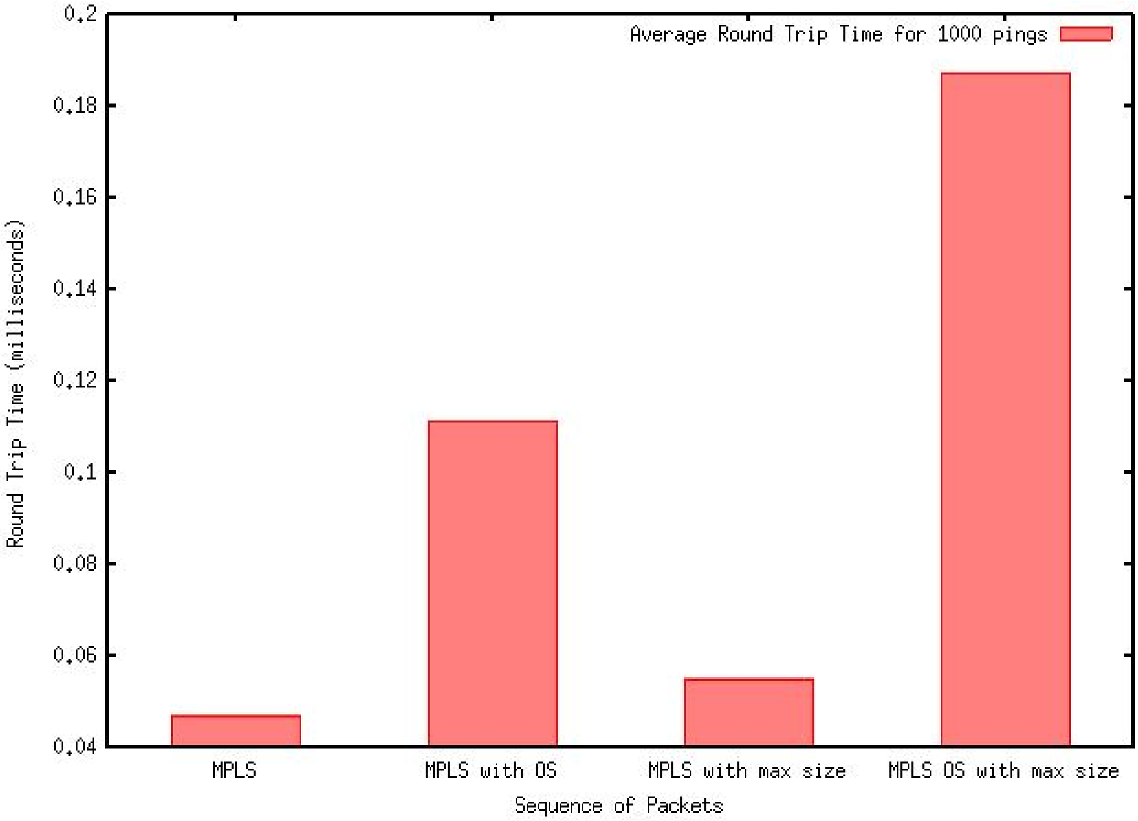


Figure 4.4: Average Latency in all the four Systems

4.3 Packet Size Limitation

The Maximum Transmission Unit (MTU) is the size of the largest packet that can be communicated into a network. Packets whose size exceeds the MTU undergo IP fragmentation where the packet is broken down into small chunks that can be transmitted into the network. The default MTU for Ethernet is set to 1500 bytes of data, exceeding which if the packets are IP packets, IP fragmentation takes place. MPLS do not undergo IP fragmentation if the packet size increases beyond the 1500 bytes allowed on the Ethernet. As we observed, MPLS adds an additional of 4 bytes of data per label pushed on the data packet. Thus care needs to be taken not to max out the packet size if it expected to flow down an MPLS network. This concern is more intensified in an MPLS OS network. OS adds an additional overhead data to the data packet comprising of the CW, the MEL and the special purpose MPLS label as well as the 16 bytes larger Cipher text data. This means that the data packet that is going to flow through an MPLS network implementing OS has to compensate for the additional data overheads by making sure the application data payload is small in size so as to not exceed the MTU. This also states that the amount of data that can be transferred over an MPLS OS network in a given time frame will always be less than data transferred in traditional MPLS network. This case can be best tested using throughput/bandwidth measuring tools like 'iperf'. We tried performing this test on the OS system, however it was observed that 'iperf' would crash the system when it was run, freezing the virtual box in which the test was being performed. Further analysis and debugging will be needed to figure out the issue.

Chapter 5

Future Work

Implementation of Key Exchange

Another important part of the MPLS OS not covered in this project is the initial Key Exchange between the LSRs implementing the MPLS OS. Before Encryption can begin the two participating LSR's need to agree upon an encryption key. Rather than using a key transport mechanism the aim of OS is to implement a 'key agreement' mechanism that can securely be used to agree upon a key value.

This can be done using a Diffie-Hellman Key exchange. The LSR's should also be able to support a key exchange in the middle of data transmission and gracefully shift to the newly agreed key without any disruption in the flow of packets due to mismatched keys. Thus the LSR's should have the capability to handle at least 2 keys for a given LSP. After the agreement of the new key the encrypting LSR should start encrypting the new packets with the newly agreed upon key. Similarly at the decrypting LSR, once it starts receiving packets encrypted with the new key then the LSR should discard the old key. It is recommended that the key on an LSP be changed at least once every day or every 10 raised to 6 packets whichever is sooner, and MUST change keys before encrypting 2 raised to 64 packets [2].

Implementation of Monitoring tools

Measuring the performance metrics and monitoring the behavior of OpenVSwitch at the Kernel module level is a difficult task. There is no easy to use tool that helps in monitoring OpenVSwitch without the tool adding it own overheads at the kernel level.

[25] have suggested certain monitoring designs and tested them on OpenVSwitch by tooling it with monitoring capabilities.Using a combination of Flow Capture Schema (FCAP) and Sketch based monitoring (SMON) designs the authors were able to monitor OpenVSwitch internally with little to no overhead introduced by the tools. It would be worthwhile to investigate its implementation for further closely monitoring MPLS OS as the insight gained by data gather from these tools might prove invaluable.

Implementation on bare metal switches

An ideal scenario to evaluate the performance of OS is to implement it on production networks using production level bare metal switches. The behavior of the switch as it responds to the variety of traffic that flows through it as it implements OS would be interesting to observe and may provide valuable insight into its performance.

This however may prove difficult to implement as the cost of experimentation may be too high and its effects may impair service level agreements of currently running data flows.

Chapter 6

Conclusion

In the current age of high speed networks carrying terabytes of data, security and privacy of data is often overlooked. Mostly due to lack of concern or lack of awareness regarding the various security threats that affect the network systems. The reliance of service providers and corporations on the MPLS network means that MPLS network is a lucrative target for attackers and thus making MPLS as secure as possible is of prime concern. This is what Opportunistic Security in MPLS network tries to accomplish by encrypting data whenever it is possible to do so so as to not affect the current on going data transfer process.

However its impact on the performance of the system should also be taken into consideration. Any solution that weighs down the current functionality of the system is not ideal and will most likely not be implemented. Based on the experiment conducted in this project we observed that the implementation of the Encryption part of the MPLS OS introduces some slight overhead in the performance of the system. However this overhead is not that significant compared to the overall performance of the system which mostly remains at the same level as compared to the traditional MPLS system.

Bibliography

[1] Denise Grayson, Daniel Guernsey, Jonathan Butts, Michael Spainhower, and Sujeet Shenoi, “Analysis of security threats to mpls virtual private networks”, *International Journal of Critical Infrastructure Protection*, vol. 2, no. 4, pp. 146–153, 2009.

[2] S. Farrell A. Farrel, “Opportunistic Security in MPLS Networks draft-ietf-mpls-opportunistic-encrypt-03”, nternet-Draft, March 2017.

[3] RN Pise, SA Kulkarni, and RV Pawar, “Packet forwarding with multiprotocol label switching.”, in *IEC (Prague)*, 2005, pp. 277–281.

[4] L. Fang, “Security Framework for MPLS and GMPLS Networks”, RFC 5920, July 2010.

[5] G. Swallow T. Nadeau, M. Morrow, “Operations and Management (OAM) Requirements for Multi-Protocol Label Switched (MPLS) Networks”, RFC 4377, February 2006.

[6] M. Betts M. Vigoureux, D. Ward, “Requirements for Operations, Administration, and Maintenance (OAM) in MPLS Transport Networks”, RFC 5860, May 2010.

[7] Sahel Alouneh, Anjali Agarwal, and Abdeslam En-Nouaary, “A novel path protection scheme for mpls networks using multi-path routing”, *Computer Networks*, vol. 53, no. 9, pp. 1530–1545, 2009.

[8] G. Fedorkow Y. Rekhter D. Farinacci A. Conta E. Rosen, D. Tappan, “MPLS Label Stack Encoding”, RFC 3032, January 2001.

[9] Zack Whittaker, "Nsa is so overwhelmed with data, it's no longer effective, says whistleblower", 2016.

[10] Daniel Guernsey, Aaron Engel, Jonathan Butts, and Sujeet Shenoi, "Security analysis of the mpls label distribution protocol", in *International Conference on Critical Infrastructure Protection*. Springer, 2010, pp. 127–139.

[11] Michael Spainhower, Jonathan Butts, Daniel Guernsey, and Sujeet Shenoi, "Security analysis of rsvp-te signaling in mpls networks", *International Journal of Critical Infrastructure Protection*, vol. 1, pp. 68–74, 2008.

[12] Bruce S Davie and Yakov Rekhter, *MPLS: technology and applications*, Morgan Kaufmann Publishers Inc., 2000.

[13] Robert Braden, Lixia Zhang, Steve Berson, Shai Herzog, and Sugih Jamin, "Resource reservation protocol (rsvp)–version 1 functional specification", Tech. Rep., 1997.

[14] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3 ", Tech. Rep.

[15] V. Dukhovni, "Opportunistic Security: Some Protection Most of the Time Abstract", RFC 7435, December 2014.

[16] Y. Nir P. Eronen T. Kivinen C. Kaufman, P. Hoffman, "Internet Key Exchange Protocol Version 2 (IKEv2)", RFC 7296, October 2014.

[17] L. Martini D. McPherson S. Bryant, G. Swallow, "Pseudowire Emulation Edge-to-Edge (PWE3) Control Word for Use over an MPLS PSN", Tech. Rep. 4385, February 2006.

[18] Bob Lantz, Brandon Heller, and Nick McKeown, "A network in a laptop: rapid prototyping for software-defined networks", in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, 2010, p. 19.

[19] Rogério Leão Santos De Oliveira, Ailton Akira Shinoda, Christiane Marie Schweitzer, and Ligia Rodrigues Prete, "Using mininet for emulation and prototyp-

ing software-defined networks", in *Communications and Computing (COLCOM), 2014 IEEE Colombian Conference on*. IEEE, 2014, pp. 1–6.

[20] "Production quality, multilayer open virtual switch", 2016.

[21] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan J Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, et al., "The design and implementation of open vswitch.", in *NSDI*, 2015, vol. 15, pp. 117–130.

[22] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner, "Openflow: enabling innovation in campus networks", *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[23] "[ovs-discuss] reg pushing two mpls labels", 2014.

[24] "[ovs-discuss] error when popping mpls tag from vlan tagged packet", 2016.

[25] Zili Zha, An Wang, Yang Guo, Doug Montgomery, and Songqing Chen, "Instrumenting open vswitch with monitoring capabilities: Designs and challenges", in *Proceedings of the Symposium on SDN Research*. ACM, 2018, p. 16.

The Codebase of this entire experiment and results found are uploaded on a github repository with the link:
https://github.com/SalilAj/Encryption-in-MPLS-Networks