Research on Performance and Security Improvements by Software Defined Network

ビン サフリ, ノル マスリ

https://doi.org/10.15017/1785422

KYUSHU UNIVERSITY

DOCTORAL THESIS

Research on Performance and Security Improvements by Software Defined Network

Nor Masri SAHRI

2016

Kyushu University, Graduate School of Information Science and Electrical Engineering, Department of Advanced Information Technology, PhD Course

I hereby declare that this thesis entitled "Research on Performance and Security Improvements by Software Defined Network " is the result of my own research except as cited in the references. This dissertation has not been accepted for any degree and is not concurrently submitted in candidature of any other degree.

Signature :

Student : Nor Masri SAHRI

Date : April 2016

Supervisor : Professor Koji OKAMURA

To my family; parents, wife (Norul Huda) and childrens (Alya, Dania, Iman and Isao). I would not have done this without you. Thank you for your endless love, encouragement, and support.

Abstract

Over the past decade, Internet and networking principles mostly remain unchanged. Network devices such as switches and routers are being developed by many business entities and the devices usually use copyrighted operating system and software. By using different types and brands of network equipment force an organization to employ a specialist on every brand. Different configuration of different systems also increases the possibility of network misconfiguration which is highly dangerous situation for the organization itself. Thus, there is a need for a new technology to make networks more scalable, dynamic and to allow easier management of network devices from different vendors. Thus, there is an urgency for a new innovation to make arranges more adaptable, element and to permit less demanding administration of system gadgets from various vendors. These needs could be fulfilled by programmable networks like Software Defined Networking (SDN).

SDN is currently ongoing developing technology that tracts consideration because of its worldview, that parts the control plane from information sending plane and because of the high programmability enable function, it is believed by many that it could replace traditional networking. The whole idea is to reduce the distribution complexity of data plane devices, e. g., network switches, which is allow the forwarding plane to forward the network traffics according to a set of actions that was defined by the specific software module in the control plane.

By having SDN, the differences in proprietary operating systems or interfaces of network devices is eliminated and makes the network administration works less burden and independent from data plane vendors. SDN also enables rapid development of network applications from developers that can be part of the single control plane which can be managed from a centralized interface and have access to all connected devices.

The control plane act as the network operating system which has the ability to control the states of all connected network. Data plane, in the other hand, is to receive the instructions from the control plane and simply just forward the data packets. Currently, a standard data plane

abstraction such as Openflow, enable the use of any type of data plane devices that available since all of the connected network devices can be manage by a common open source protocol.

OpenFlow protocol is a clean slate project introduced by Stanford University back in year 2008. OpenFlow was implemented as the first open standard interface SDN architecture. In Openflow, the data path and the high level routing decision are made from two different devices, which is the OpenFlow-enabled switch and controller, respectively. The central controller provides the switches with the operational rules instructions, which is pushed by the controller to the switch as individual or group flow entries via a secured channel between them using OpenFlow protocol. Furthermore, Openflow actions function can be performed on network flows that can be forward and drop packets or manipulating the packet header informations.

The adjustability provided by SDN and Openflow create a sudden explosive of new research area and applications such as application aware network, network flow virtualization, real time QoS support, helping Internet of Things (IoT) to reduce the strains of data generated by IoT devices, new ways to deteck DDoS attacks any many mores. Researchers believed that a large number of new network applications will be introduced which will enhance the current network operations.

Firstly, this thesis proposed a fast and efficient failover mechanism for redirecting traffic flows to optimal path when there is a single link failure problem. Setting pre-established alternative paths will result in a faster switchover compared to establishing new alternative paths on-demand. This approach however may lead to the use of sub-optimal alternative backup path. The combination of both fast rerouting with predefined backup path approach and optimal path computation would be the best solution for service restoration. Using the ability of the Openflow enabled switches to report link failure to the central controller, the proposed method allows the controller to do the real time path computation and while the controller perform the path calculations, the incoming traffics in the current switches was already rerouted to the predefined backup path that was installed previously by the controller.

Secondly, this thesis proposed an adaptive sampling strategy that is able to provide essential traffic statistics for more accurate anomaly detection in SDN. Extracting important features for the application to analyse the anomaly detection problem, introduce significant

overhead on the way of switch handling. Furthermore, high volumes of network traffic introduce notable issues that affect the performance and anomaly detection accuracy. Taking advantage of centralized control plane of Software Defined Networking (SDN), the task to handle the flow information is much more simplified programmatically. The accuracy of the measured flow statistic plays important role in anomaly detection. While the use of sampling is capable to lessen the scalability problem of traffic monitoring, the insufficiency of sampled flow statistic may have led to inaccurate detection rate of anomaly. This thesis sampling mechanism utilizes the clustering analysis, which is used to classify the attack in the network to determine the severity of monitored traffic. By manipulating the type of service of incoming packet together, these two important parameter formulate the sampling mechanism algorithm. This thesis shows experimentally that by putting higher polling frequency on detected anomalous flow, it to detect network attacks much more accurate.

Lastly, this thesis proposed *CAuth*, a novel mechanism that autonomously block the spoofing DNS query packet while authenticate the legitimate query. As DNS packet are mostly UDP-based, make it as a perfect platform for hackers to launch a well-known type of distributed denial of service (DDoS). The purpose of this attack is to saturate the DNS server availability and resources with "unwanted" DNS query traffic. This type of attack utilizes a large number of botnet and usually perform spoofing on the IP address of the targeted victim. While it is difficult to identify which one is legitimate or attack traffic, this thesis takes a different approach for spoofing detection and mitigation strategies to protect the DNS server by utilizing Software Defined Networking (SDN). By manipulating Openflow control message, this thesis designs a collaborative approach for DNS usage authentication between client and server network. Whenever a server controller receives query packet, it will send an authentication packet back to the client network and later the client controller also replies via authentication packet back to the server controller. The server controller will only forward the query to the respective server if it receives the replied authentication packet from the client. Most notably, the mechanism designed with no changes in existing DNS application and Openflow protocol. From the evaluation, *CAuth* instantly manage to block spoofing query packet while authenticate the legitimate query as soon as the mechanism started.

Acknowledgements

All praise is due to Allah almighty God for all the graces, and blessings bestowed upon my family and me during my study in Kyushu University and living in Fukuoka, Japan. The process of earning a doctorate and writing a dissertation is long and arduous. It is certainly not done singlehandedly and beyond question, without guidance, loves and sacrifices.

First and foremost, I would like to thank my wife (Norul) and family (Alya, Dania, Iman and Isao) for putting up with an absentee husband and father during this process. Norul has been unfailingly supportive – and has borne the burdens which have fallen on her shoulder as I spent my most of the time and energy pursuing goals that took me away from her and the family.

I would certainly be remiss to not mention and sincerely thank my Professor, supervisor and mentor Professor Koji OKAMURA, whom without his help, wisdom, guidance, expertise and encouragement, this research would not have happened.

I also would like to show my deep gratitude to my loving parents, who have been my first teachers as they taught me about life, and have always been encouraging me to achieve more and do my best.

In addition, I would like to express my gratitude to Professor Akihiro NAKAO, Professor Yasuo OKABE, Professor Kenji ONO and Professor Hirohumi AMANO for their kind support, guidance, and enlightening discussions throughout the meetings along my PhD study. Meetings they attended were really pleasant and fruitful, thanks to their kind comments, and advice that helped a lot to shape my work.

I would like to express my gratitude towards all members of Professor Okamura's laboratory; Mr Othman, Mr Chengming Li, Mr Alaa Allakany, Mr Yowaraj Chetri, Mr Zhaolong Ning, Mr Zafran, Mr Ariel and all others that have their place in my memory. Thanks a lot for your kind accompany, comments, encouragement, and support.

Last but not least, I would like to thank the Ministry of Higher Education of Malaysia and University of Technology MARA Malaysia for supporting my study in Japan. Also, I would

like to thank Kyushu University, and the Graduate School of Information Science and Electrical Engineering, and all their Professors and Officials for their kind support.

Table of Contents

List of Figures

List of Tables

Chapter 1

1. Introduction

In Internet, routing protocols played important roles in the whole packet forwarding operation. Finding the best path for a packet to traverse to their destination is the most discussed in research world. Many routing protocol had been proposed and used in routers to perform autonomous routes configuration. Example of famous routing protocols such as Routing Information Protocol (RIP) [1] [2], Open Shortest Path First (OSPF) [3], Intermediate System to Intermediate System (IS-IS) [4], Border Gateway Protocol (BGP) [5], and many others.

1.1. Background

As stated before, the main function of a network is to transport packets. For this matter, two important elements are decomposed: *routing* and *forwarding*. The *routing* task is responsible to determine how many currently available network nodes, the state of their interconnections, how many available resources and many other factors that the packets must follow while traverse the network. The router also need to compute independently from others on how each node in the network should forward the packets to create a forwarding behaviour for the packets to follow. It is worth to note that the *routing* task must be running continuously because of the online changing behaviour of network that can change at any moment. For example, the interconnected link between network nodes might fail without warning or traffic congestion may be build up due to sudden changing pattern on network traffics.

Routing task may involve a lot of more complex algorithm while on the other hand, the *forwarding* task is much a simpler operation. In this task, each network nodes are responsible to forward the packets base on the information provided by the *routing* task for that particular node and a label is attached to every packet itself. Figure 1.1 illustrate the forwarding operation in a single network element. One important element that reside on every node is the interface between *routing* and *forwarding*. This element is a collection of data structures which is called *Forwarding Table* (FT), one at each network element. The *routing* task responsible to maintain

the changing state of the FT while the *forwarding* task use the available information in the FT to determine how to forward any incoming packets. The split operation into *routing* and *forwarding* allow network nodes to optimized all available components for each task. Nowadays, modern network requirements for the performance demand very good techniques for forwarding packets to their destination.

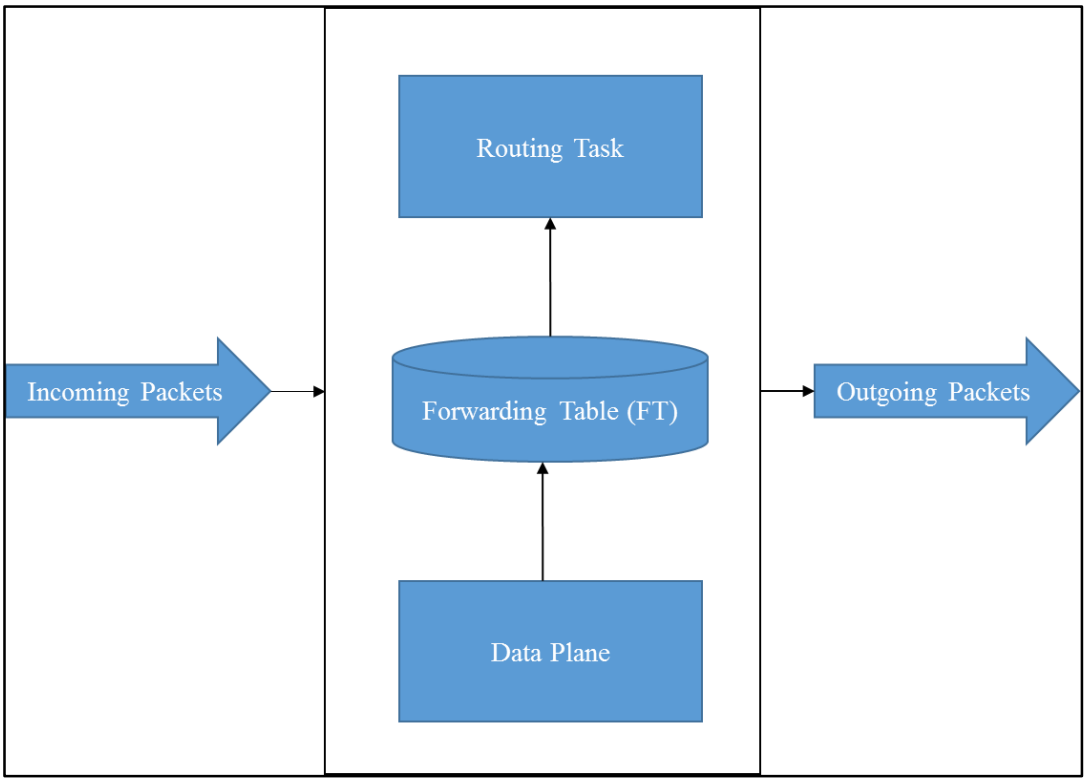

Figure 1.1 *Routing* and *forwarding* task in a single network node

As briefly explained in previous paragraph, current mechanism to control the network via routing protocols imposes a tight coupling between the *control* plane and *forwarding* plane which reside in every single network element. Every decision making is performed on the same single network device that also perform the packet forwarding operation. With more complex Internet architecture introduced, it has become difficult for engineers to deploy any new network applications, protocols and functionalities. In traditional network operations, any new functionalities need to be implemented directly into the network devices which can be a tedious task, especially that all of those available network equipment vendors does not allow direct

privilege on their equipment. Due to business competition, vendors are also tending to keep their functionalities close to others, thus making their equipment with limited capabilities.

One of the current research trends that tend to brake the tight coupling between *data plane* and *forwarding plane* which aim to overcome the above stated problems. In this method, the *control plane* and *forwarding plane* element of the network reside on different machine. Software-Defined Networking (SDN), allows a network to customize the behaviour through some centralized policies at a logically centralized network controller. In this technology, the control plane act as the network operating system itself, which is responsible to control and maintain the state of the whole network under the controller boundary. SDN replaces traditional Internet that closed, tight coupled and riveted network function with much more generic packet processing mechanism, open source program that control by software program through centralized platform. This openness method opens the broad capabilities of network devices and provide the customer with many types of manipulating the network and flexibilities in managing network devices in their network.

In conjunction with SDN, the Openflow protocol [31] has made a drastically important paradigm shift by establishing (1) a protocol that act as a two-way communication channel that enable the centralized controller to install related forwarding rules, querying the current network states and for the switch to notify the central controller for a various event such as when no packet match rules in the switch forwarding table (2) a data path that presents a clean flow table abstraction which contain a/many sets of packet field to match with and with associated action.

The Openflow forwarding abstraction is meaningful enough to allow variety types of packet processing functions that cannot be achieve with the traditional *routing* and *forwarding* before. The Openflow provides a simplified interface for the centralize controller applications and the central controller are also able to deploy the application to whole network under it. This protocol realizing a dream for the network operators by writing a simple and centralized network program with the global view of network states.

1.2. Motivation and Goals

Software Defined Networking (SDN) and Openflow seems to be the future Internet technology that enable innovative and creative applications development that were unachieved in current traditional Internet. By controlling the functions of every network node centrally, the management and network programmability is much effortless and practical. To enable smarter future Internet, a standard body [6] has dedicated to promote and adopt SDN through open standards development. Despite SDN decouple the control plane and forwarding plane separately and have more advantage than the traditional IP networking, it is still incomplete and has some challenges unsolved in certain area of implementation.

The first goal of this thesis is to provide fast traffic recovery upon link failures for important network applications. In [7], upon any single link failure, the service restoration time for the impacted traffic flows should be below 50 milliseconds. Since SDN architecture is managed by a network controller centrally and the forwarding nodes capabilities is restricted to only listen the instruction from the controller, natively the SDN cannot guarantee the failure recovery to be below 50 milliseconds to support the network enterprise architecture. By constructing multiple routing table as in framework [8], different paths for unicast traffic, multicast traffic and different classes of service based on some flexible criteria is achieved. On the other hand, in real Internet environment, no IP fast reroute using backup topologies had been implemented. In reality, IP fast rerouting require the complexity and the ability to coordinate forwarding functions to a forwarding hardware.

The second goal of this thesis is provide more accurate network attack events to reduce the false alarm on network attack detection. To detect the unusual network traffics, [9] and [10] have recommended the use of packet sampling rather than capturing every packet to do the analysis. Capturing every packet requires too much network overhead such as high processing capability and high network resources used.

Network anomaly detection techniques [11,12,13] is based on analysis on network traffic and the characteristic of the dynamic statistic features in order to detect network abnormalities quickly and accurately. It is paramount to balance the trade-off between accuracy of anomaly detection and overhead introduced from the flow traffic measurements (e.g.-scalability). Sampling decision should have some intelligent ability to address some of the requirement such as to reach low false alarm and low computation convolution objective. Therefore, we utilize Openflow and SDN capabilities to control the network topology centrally with the main objective is to provide more accurate network attack detection.

The final goal of this thesis is to detect and mitigate the spoofing IP addresses that exploit UDP protocol to launch the botnet attack to the most critical application in Internet, the DNS services. Connectionless and lightweight protocol make the UDP as the perfect tool for attackers to exploit to launch botnet attacks [14]. To make UDP flood attack effective, attackers overwhelm a large number of random ports on the targeted host with the spoofed IP address UDP datagrams. Thus, as more and more UDP packets are received and answered, the targeted system such as DNS server, loaded with request and at last unresponsive to other legitimate clients [15,16]. Moreover, during large scale of DDoS attacks, the victims drop all of the packets destined to them.

To protect from spoofing attacks, many protection method has been introduced such as ingress filtering [17], rate limiting [18], packet marking [19,20,21], encryption, PKI, IPSec authentication [22], access control. However, it is known that those methods increase the network overhead. In this thesis, we aim to design a low cost authentication method for DNS services so that the DNS provider can clearly distinguish attack packets and legitimate DNS queries utilizing SDN paradigm.

The reminder of this thesis is organized as follows. Chapter 2 illustrates technical background and network architecture on SDN. Fast failover with more optimal path – SDN based is presented in Chapter 3. Chapter 4 describes an adaptive query rate method for anomaly

detection with SDN. Collaborative spoofing detection – SDN based *looping authentication* for DNS is proposed in Chapter 5. At last, Chapter 6 concludes all the works.

Chapter 2

2. Literature Review

This chapter provide more details briefly on the major topics touched in this study which are; the Software Defined Networks architecture as well as the Openflow protocol, the limitation in IP fast rerouting in the current Internet architecture and network security topics which are the effect of flow sampling in anomaly detection techniques and the effectiveness of current DNS services protection against DDoS attacks that manipulate spoofing the victim IP addresses. It is paramount important to understand the following parts of this study.

2.1. Architecture of SDN

At first, it is important to shed the light of SDN itself. SDN is proposed to provide an "industry-wide" standardized protocol that allows a higher-level of abstraction and manipulation of network control to configure the *forwarding* plane and also to interact with the *forwarding* element at the same time. SDN promises innovation in networks which the *forwarding* protocol would allow innovative network management methods to be implemented on network devices without requiring expensive standardization process.

The separate *forwarding* element in SDN should be flexible enough so that various ways and method of control algorithms can be implemented with it. For example, it is not enough to forward packets based on Ethernet packets destination address only, since some of the applications might need to be forwarded using their source address as well. Also, it is not adequate for the network devices only allows the routing task to define a single outgoing port for any given packets, since some of the applications might need the ability to anycast or multicast.

Many specialized network devices function can be eliminated if the generality of *forwarding* abstraction are enough to support variety of network functions beyond the simple Layer 2 or layer 3 forwarding such as the ability to provide more access control rules and also more accurate network traffic monitoring. The use of much simpler packet processing

device may simplify the task of network management, improved the resources utilization of the equipment and also could decrease the operation cost for the network operators.

As proposed by Open Networking Foundation (ONF) in [6], SDN architecture decoupled the network control from the forwarding and it is directly and highly programmable. The network intelligence logically centralized in software-based network controller, which responsible to maintain a global view of the underlying network. The SDN architecture is depicted in Figure 2.1. With the central controller introduced, the network viewed by the applications and policy engines as one single and logical switches entity. It enables the network operator to control the entire network from a single point of view which greatly simplifies the network design and operation. SDN also make the network devices less complicated since they are no longer have to process and understand different protocols standard but just only accept instructions from the centralized controller.

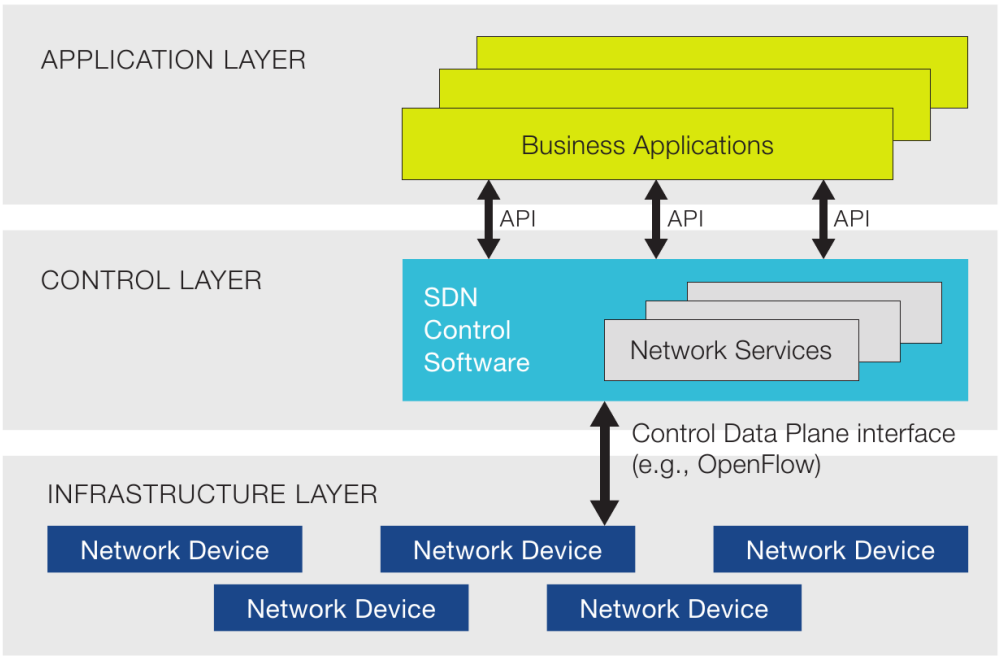

Figure 2.1 SDN Architecture (as shown in [6])

The lowest layer in SDN which is Infrastructure Layer, is a form of interconnected network devices. This layer communicates directly with the lower layer which is called Control Layer through a Control Data Plane interface. This interface is open standard such

as Openflow protocol. The Control Layer is the heart of SDN operation where it consists all kind of network services and the SDN controller that control the network devices itself. The highest layer which is Application Layer where one or more network applications are stored. Those applications communicate to the Control Layer through set of APIs.

2.2. Brief Introduction to Openflow

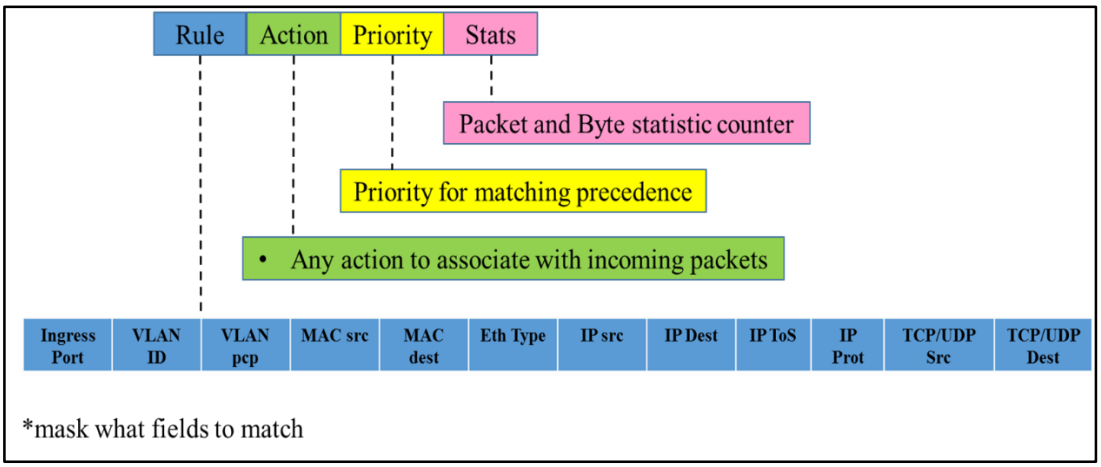

Figure 2.2: Openflow flow table.

Openflow protocol has become *de facto* standard for the SDN switch-controller protocol and forwarding plane abstraction. Variety of enthralling network control functions has been introduced utilizing Openflow protocol such as server load balancing [23], campus network access control policies [24], load balancing routing algorithms in data centres [25] and many others. Initially, an Openflow switch establish communication with their network controller through a secured communication channel. Then, this channel is used to exchange information between the switch and the controller and also to allow the controller to install rules into switches flow table to determine the switch forwarding behaviour.

As depicted in Figure 2.2, a flow table consists of a collection of pre-defined rules, where each of this *rule* consists of a match condition that defines which type of packets the rules apply to. This field is to mask what packet fields to match with, for example, to match with the VLAN ID field. Next important parameter is *action* field. This field define how the associate packets that match the *rule* should be processed. When there are any incoming packets into the Openflow switches, it will find the highest *priority* rule to match first. The flow entries with highest rule will be processed by the switch first. It then executes the *action*

associated with the *rule*. The *stats* field is a counter that hold the flow entries current packet and bytes statistics. If the incoming packets dose not match any pre-defined rules, the packet is encapsulated in an Openflow message called *Packet-In* and sent to the controller for further decision making. This message indicates to the controller that particular packet requires further rules to be decided.

As depicted in previous Figure 2.2, many packet header fields can be as matching criteria for incoming network packets. Openflow 1.0 allows 12 type of packet header fields from layer 2,3 and 4. One type of rule that put no condition on one or more fields is introduced which called *wildcard* rule. This type of rule can be used to reduce the control traffic in network. Openflow rules allows variety of actions, such as modifying the destination address of the incoming packets to reroute to other destination, forwarding packet to more than one destination or replicate the incoming packets. Thus, the network controller can program L2 and L3 routing, do customization on network policies and QoS related forwarding decision. Moreover, each flow entry in switch flow table can be configured with two type of timeouts which is *hard* and *soft* timeout. Any rules defined with *hard* timeout will be removed from switch after *t* seconds have elapsed since the rule installation. Any rules defined with *soft* timeout, the rule will be removed after *t* seconds no packet match the rule.

Openflow protocol support three major type of messages, *controller-to-switch*, *asynchronous* and *symmetric*. *Controller-to-switch* messages initiated by the network controller and used to directly manage or inspect the switch state. *Asynchronous* messages are initiated from the switch and it is responsible to update or give information to controller about any network event and changes to the switch state. Lastly, *symmetric* messages are initiated by either the switch or the controller and sent without solicitation. The message types used by Openflow are summarized in a Table 2.1 below.

Table 2.1: Summary of key OpenFlow messages as Openflow 1.3 version.

Message Type	Sender	Description
Feature Request	controller	Controller requests switch identity and capabilities (i.e. port information)
Configuration	controller	Controller set or query configuration parameter in switch.
Modify-State	controller	Insert, modify, or delete flow table entries
Read-State	controller	To collect information such as current configuration, statistics and capabilities from switch
Packet-Out	controller	Send packets out of a specified ports on the switch
Flow removed	switch	A flow entry was removed due to timer expiration or flow entry delete command.
Packet In	switch	Notification that the given packet arrived at the switch either because it failed to match in flow table or it matched a rule whose action directed the packet to the controller.
Port-status	switch	Inform controller for changes on a port
Stats reply	switch	Report of port or flow statistics of the switch.
Flow-monitor	switch	Inform controller of any changes in flow table

2.3. IP Fast Rerouting

2.3.1. Reliability in Network Design

Reliable network design is classified into two main categories: proactive protection and dynamic restoration [75]. In protection, alternative paths are pre-defined and pre-computed. By doing this, fast restoration against network failure is achieved by avoiding reactive option. Vice versa, reactive restoration performs the computation of backup routes after the failure happen. Even though backup route is optimized in accordance to the network changing behaviour, reactive restoration require more time compared to protection technology. Proactive protection is realized in MPLS [76] while dynamic restoration is realized by IP protocol such as OSPF [77].

In OSPF, each individual router has the whole network topology information and the synchronization of information is performed among routers. In order for a packet to reach their destination, route is computed based on a link metric and the shortest patch which has the lowest metric is assigned as the route. For the IP services reactive restoration, main issue in terms of improving IP resilience is its long restoration time.

Many previous researches focusing on optimization of IP services restoration time is proposed. Minimization of convergence time [78] and the computation time [79] for OSPF is proposed and evaluated. For example, by limiting the notification area in OSPF flooding, network convergence is shortened [80]. IP fast reroute which embed the concept of MPLS proactive restoration mechanism to IP is proposed. The basic strategy of IP fast reroute is to pre-defined backup routes as in MPLS path protection, and able to switch to backup path if the primary path service is disrupted. IP fast reroute approaches require extension to the current IP framework since there are no schemes to maintain the pre-defined backup routes in pure IP network.

For example, backup topology-based approach [81,82] has multiple routing tables stored in a router, and uses them in accordance to changes in network state. For other IP fast

rerouting [83–86], even-though the mechanism of failure detection and packet forwarding are different, the general strategy in this previous work is to prepare backup tables, which store precomputed backup paths.

2.3.2. IP Fast Reroute in Internet

For IP services to respond fast to any link disruption in the network, it requires to use backup topologies. Each of this backup topologies are pre-defined and installed in the forwarding nodes. Each router computes the shortest path and then create the main path to reach their destination and also for their backup topology. When the router detects a link failure, it will independently search for the backup topology that protect the failed link.

Motivation of IP fast rerouting is to achieve millisecond-order restoration time without alternate packet forwarding techniques such as MPLS. However, the current IP services restoration take longer time to recover. Self-organizing and distribute network control is introduced in OSPF and since then, it is widely used as the mainstream IP routing protocol in the Internet. However, when there is a link failure, the convergence process is slow because of their nature; reactive and global [69]. Thus, OSPF cannot support the requirement to be a carrier grade network since the carrier grade network require below 50ms network recovery [70]. The fast rerouting concept then is introduced in MPLS to achieve the carrier grade network recovery requirement.

In MPLS, the alternative approach to realize the IP fast rerouting concept is introduced where the router has the ability to pre-compute the backup path routes that will allow the link failure to be repaired locally by the router that detect the failure without the immediate need to inform other router in the line about the link failure event. In this case, the network disruption time is being limited to only small amount of time to trigger the backup routes. But, the implementation of fast rerouting in pure IP network is not widely implemented since it requires the adaptation of any commodity hardware/routers to the forwarding function since the mechanism to employ backup routes is very different from MPLS.

The problem of fast rerouting using backup topology in IP network is to minimize the number of backup topology information in the router itself to ensure the scalability and relevance to actual condition. This is because the size of routing table in the router is proportional to the number of pre-computed backup topologies. In SDN, the central controller has the ability to modify/delete/remove any flow entries in the router on the fly. The control function is place in a software controller and the routing table which called flow table can be program by the controller, which means that the control of the network does not limited on hardware implementation.

2.3.3. Proposed Research Objective

As discussed in previous sub-chapter (2.3.1 and 2.3.2), IP is used to realize reactive restoration in terms of reliability of the network. As a result, the main disadvantage of IP is restoration time is longer even-though the IP services can handle unexpected network changes by its autonomous and reactive property. Although there were many previous studies to shorten the IP restoration time based on IP reactive property, the network restoration time is still not achieved the level of MPLS proactive protection.

Table 2.2 Objective of proposed research (Chapter 3).

	IP Network
Reactive Restoration	Current Technology
Proactive Protection	**Objective (Chapter 3)** 1. Carrier-grade fast restoration 2. More optimal path selection for backup routes

Motivate by the above mentioned current state of IP fast rerouting, we set our objective to show the feasibility of IP fast rerouting in the actual use. To modify hardware nodes seems to be impossible and may become a proprietary technology. By focusing on SDN based protocol, Openflow can provide the ease to program a network to realize the specific forwarding for IP fast restoration.

Furthermore, by setting pre-established alternative paths, results in a faster switchover compared to establishing new alternative paths on-demand. Fast rerouting may lead to the use of sub-optimal alternative backup path. The combination of both fast rerouting and optimal path computation would be the best solution for service restoration.

2.4. Network Anomaly Behaviour Detection

Anomaly detection is the problem to find patterns in the network data that do not conform the expected normal behaviour. Two commonly used term in anomaly detection context is anomalies and outliers; sometimes interchangeably. Variety of applications use anomaly detection techniques to find anomalous behaviour in their activity such as fraud detection for credit card and intrusion detection for cyber-security events. The importance of this methodology is due to the fact that any anomalies in data often translate to significant action in wide variety of applications area. For example, an anomalous traffic pattern in a computer network could mean that a hacked computer is sending out sensitive data to an unauthorized destination [87].

Over time, a variety of anomaly detection techniques have been developed in several research communities especially focusing in detecting anomalies in network traffic such as [43-45].

2.4.1 Accuracy of Sampled Flow

Flow-based monitoring is the most common tool used to analyse the network traffic to collect information on bandwidth utilization and network resources as well as network dysfunctionalities and attacks. The infrastructure consists two parts:

1. the data collector which responsible to receive any exported flow from the data sources and stores them for future analysis.
2. The data source that responsible to analyse the traverse packets in the network and also responsible to create flows.

The data source analyser is usually implemented individually inside the network device itself, such as routers and switches, and it takes the advantage of the vast amount of its resources to analyse the packets and also store the information while those packet is active in the network. The extracted flows from the data source are sent to the data collector periodically through the same network that is being monitored. The communication between the flow collector and the data source is typically done by the use of some proprietary protocol such as sFlow, Jflow, Netflow or IPFIX, an IETF standard.

Using these devices to monitor the network has several advantages since it is not necessary to add extra component. The entire packet processing is done inside the network devices such as routers and switches and the communication to the data collector is performed using the same network. However, in the context of high-speed networks, they exhibit the following drawbacks that are not possible to mitigate:

- Routers and switches require extra processing overhead especially in analysing high load networks and can lead to overloading their resources. When this devices overload, they also tend to skipped the flow monitoring in order to utilize their remaining resources to route packets, thus might losing important flow-based information.
- As discussed before, communication between the data collector and the data source use the same network that is being monitored. This could lead to network overloading and also to incorrect information since production traffic is mixed with network measurement traffic.
- Current routers and switches are closed and proprietary platform, the administrator are not free to modify how flows are defined or what type of flow information to be collected.

2.4.2 Proposed Research Objective

In order to overcome the limitations listed above, a flexible open-source platform to collect the flow-based information suitable for high-speed networks is needed. It is also desirable that the measurement traffic technique does not become intrusive on the network that is being monitored and does not impact on the network devices either. A separate communication channel between the data source and the data collector is also needed to not overloading the current network devices on their packet processing jobs.

To detect anomalous behaviour in the network, analysing the data via sampling is the key to determine things whether it is unusual or normal network. For example, Netflow is a sampling mechanism introduced by Cisco for a network administrator to determine thing such as the source and destination of traffic, causes of a network congestion and etc. Therefore, impact of sampling method on anomaly detection are widely discussed. In [73] and [74], the author shows that sampling degrade the performance of some type of anomaly detection algorithms and also introduce different type of network distortion.

Flow sampling influence the accuracy of any anomaly detection techniques. With the increasing amount of network traffic, sampling techniques have become widely employed allowing monitoring and analysis of high-speed network links. Despite of all benefits, sampling methods negatively influence the accuracy of anomaly detection techniques and other subsequent processing. If too frequent sampling process is being done to the network traffic, it might lead to high network resource usage. By contrast, if too much gap in sampling time interval is use to find anomaly, it might lead to loss of important traffic feature for the administrator to analyze.

In SDN, the ability of the controller to have the whole view of a network bring a lot of benefit whether to sample every flow or selective flow easily. By separating control plane and data plane feature, the network production traffic can be separated with the network measurement traffic as well, thus the control to not overloading the flow forwarding is also

possible. Furthermore, with the capability to extract many important packet header information form the incoming packet to the network such as source port information deepen the advantage of SDN to overcome the discussed drawback in anomaly detection in IP network. Since SDN use polling method in order to analyze the statistic of any flow in their network, the frequency of polling of every type of flows should be addressed properly to balance the tradeoff between the accuracy of anomaly detection and the scalability of the network resources that was used for the analysis purposes.

2.5. DNS Flooding and Amplification Attacks

The most crucial service in the Internet is DNS application since most of the network applications and services require a translation from domain name to IP addresses. As consequences, even if the service is unavailable for a short period of time, it has a significant impact to the communication in the Internet itself. Furthermore, since DNS use UDP protocol for the DNS queries and responses makes DNS vulnerable to spoofing-based Denial of Service (DoS) attack. Moreover, with the introduction of public DNS such as OpenDNS and Google DNS, every IP in the Internet are eligible to use the service and the spoofing-based DOS attack can easily exploit the platform to launch a famous DNS amplification attack with the objective to exhaust the victim network resources and interrupt the services for any legitimate users. Therefore, it is impossible for the DNS server to differentiate between legitimate and attack DNS query packet from the requesting host.

The hackers usually employ a large number of botnets to attack the DNS application. The botnet then is configured to generate small DNS queries with forged source IP addresses which can generate a large volume of network traffic since DNS response messages may be substantially larger than DNS query messages. Then this large volume of network traffic is directed towards the targeted system to paralyze it.

The employed botnet IP addresses are usually spoofed under the control of the attacker, which makes it very difficult to traceback the attack traffic even to the botnet itself. Because of

this problem, the DNS server has no choice but to handle all incoming DNS request and it will start to drop incoming request when the server itself is overloaded with quite number of requests. As a consequence, the legitimate requester will see the drops sign as a congestion problem and it will rearrange for another transmission, thus it will drastically decrease the number of legitimate DNS queries served by the overloaded DNS server.

2.5.1. Defence Against Spoofing IP Addresses

By the time a DDoS flooding attack is detected, there is nothing that can be done except to disconnect the victim from the network and manually troubleshoot and fix the problem. Hence, the ultimate goal of any DDoS defence is to detect and stop the attacks as close as possible to their sources.

The current IP protocol allows any source hosts to alter their source addresses in the IP packets. Thus, it creates a huge problem in detecting DDoS flooding attacks since the victim does not have the ability to distinguish attack packets from legitimate one if only to be based on their source addresses. IPSec protocol [88], [89] is introduced to help on this problem by authenticating the source of any IP, but this method is not widely deployed among the service providers because of its large network overhead.

Hop count filtering mechanism [90] record the information about a source IP address and the number of hops from its destination when the destination is not under attack. Once the attack alarm is notified, the method will inspect the incoming packets source IP addresses and their corresponding hops to identify and make decision to decide whether the incoming packet is legitimate or spoofed packets. However, attackers can spoof the IP addresses with the same hop count as their machines do. Furthermore, the legitimate packets are also can be classified if their hop count to IP mapping is incorrect or there is delay in the mapping database update [91].

The other possible solution for this matter is to be able to identify the legitimate source for the incoming DNS queries; spoof detection and prevention mechanism likes the ones

proposed in [61], where the author proposed a method to detect the spoofing DNS query packets. The method requires the DNS server to generate some type of cookies embedded in the DNS response packets. However, it only can authenticate the requests between the resolver and the DNS servers. The main attack tools that is from the general DNS clients, cannot been verified.

DNSSEC is then introduced as an extension to DNS which provide to DNS clients (resolvers) the origin authentication of DNS data, authenticated denial of existence, and data integrity instead of authenticating the DNS requester. It has no protection against DoS attacks thus it does not offer efficient countermeasure against flooding attacks as already argued in [71]. Furthermore, any DNS packet signed with DNSSEC will result in much larger DNS response packets than the normal DNS response packet (unsigned) and often exceed the maximal transmission unit (MTU) [72].

2.5.2. Proposed Research Objective

As highlighted in [87], the author discussed the need of collaborative defence approach against DDoS flooding attacks. Since the attackers collaborate with the botnets to launch the flooding attacks, the defenders are also encouraged to from alliance and cooperate each other to defeat the DDoS attacks. The author also believes that by combining the source address authentication and filtering mechanism is the most effective countermeasure against attacks.

Motivate by the proposed idea in [87], this thesis proposed a collaborative authentication of source IP addresses between the server and the client network. At the same time, to be effective in filtering and able to differentiate between attacks and legitimate DNS queries packets is also desired.

In SDN, with the ability to modify/replicate/copy any packet in the network, the controller has the ability to manipulate any incoming DNS queries before it reaches any DNS server for name resolution purposes. Moreover, in any existing firewall or routers, TCAM is a main issue because of its limitation and expensiveness. In general, rules or access list has to be manually pre-configured in the devices earlier, thus the memory consumption of the routers are

the main problem. With the capability to modify/delete flow entries on the fly, the rules to block or allow any incoming packet can be install only when needed, thus reduce the memory consumption problem in the router resources.

Chapter 3

3. Fast Failover with More Optimal Path – SDN Based

3.1. Introduction

In order to elaborate more on our fast failover mechanism, it is important to put the highlight on OpenFlow which is a clean slate project introduced by Stanford University. OpenFlow was implemented as the first open standard interface for Software-Defined Network (SDN) architecture. SDN enable network administrators with a central programmable management interface, which is decoupled from the underlying network infrastructure for current layer 2 and layer 3 switches. This function removes the hardness to do any testing with new protocols in real network infrastructure. In OpenFlow, the data path and the high level routing decision are made from two different devices, which is the OpenFlow-enabled switch and controller, respectively. The central controller provides the switches with the operational rules instructions, which is pushed by the controller to the switch as individual flow entries via a secured channel between them using Openflow protocol. The switches search the flow table corresponding entries and if there is any rules match, it will process the packets according to the pre-specified actions in the entries.

The incoming packet in Openflow-enabled switches is matched against the flow table and the associated actions are taken into action: similar to the existing conventional switches. The main difference is that if the packet has not match any rule in the flow table, the packet would be drop or Openflow-enabled switch encapsulate the packet and push the packet to the controller. Then the controller uses the flexibility of software to do further analysis and then decide what to do with the packets. The controller then passes the packets back to the switch together with associated action.

To ensure application services are not interrupted during flight in the network, SDN must impose a mechanism to continue to forward traffic whenever there is a link or node failure. Therefore, as in traditional networks, SDN adapt to failures by forwarding the traffic flows over alternative path similarly to any link-state routing protocol – reconverge after link failure is

detected. In SDN, central controller does the computation of path and to achieve high throughput, it can be done in advance to predetermined the backup path in the case of any link failure. In a carrier-grade network, single failure event can cause huge amount of network traffic disrupted. Therefore, network reliability is an important feature to taken care of and fast recovery of the network is strictly required.

3.2. Related Works

Recently, there are some proposals for network fail over mechanism in the SDN is proposed. The author of [26] designed a network restoration method if there is link failure in SDN. In the proposed method, the OpenFlow- enabled switches that detect the link failure for the particular flow will inform the controller about the situation. Upon receiving this notification, the controller will first examine if the precompute old path is affected by this changes. If it is, it will then calculate a new backup path for the affected flow and update the OpenFlow switch about the new information. The author also shows that the network restoration time is around 200 ms. However, the centralize controller could be overloaded with recovery messages and thus might introduce scalability issues.

In [27], the author introduces an OpenFlow-based fast recovery in optical transport networks. The centralized controller configures main and backup path and that information are provided to OpenFlow switches in one single flow table but with different priority levels. The main path is configured with the highest priority and the backup path sits in the next place. When failures happen, the affected switches will use auto-reject message to delete the main path entries and switch to the next backup path. However, the pre-configured backup path is not always the optimal path and the process of switching from main path to backup path would also introduce a significant delay.

From Figure 3.1, consider a scenario where the main and backup path are precomputed by the OpenFlow controller and added to the flow table of respective OpenFlow switches (main and backup path are calculated based on shortest path algorithm). When link between source node *s* and node *C* is fail, the incoming traffic will redirect to the next configured backup path, which is link *s-E-F-d* (the next best shortest path). The link utilization of backup path is already achieving 90% of the total bandwidth provided. When the traffic is redirect to preconfigure backup path and when the incoming packet is using this backup path, the path is already congested and might affect the overall performance. As enormous amount of traffic of data

communication within the data center projected by Cisco [28], clearly traffic delay is not a tolerable issue. Inspired by the problem stated in Figure 3.1, we design a fast failover mechanism for SDN using Openflow to provide a link protection, fast failure restoration in the case of link failure and to ensure the traffic are route through more optimal available path.

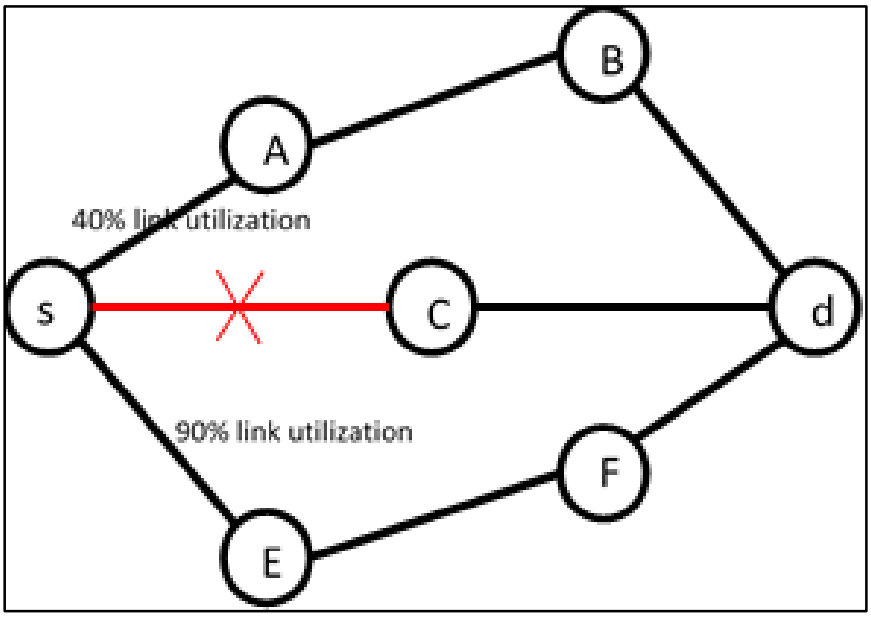

Figure 3.1. Non Optimal Backup Path Problem

3.3. Fast Failover with More Optimal Path Mechanism

3.3.1. Backup Topology Decision

A designed flow table scheme must be proactive; able to preconfigure the backup routing information in advance before the network fail and local; the process of rerouting can be done without any failure notification in order to achieve fast reroute. By constructing multiple routing table, as standardized by IETF [29], it can ensure a fast restoration when a single link fail. There are many works done in literature focusing on determining backup path to ensure network reliability, most notably [26]. The author proposed a fast convergence by preconfiguring a node next-hop backup and allow packet to reroute immediately after a failure detection. The proposed method use a multiple backup routing table installed in every node and the routing table selection decision is based on the network state.

In Figure 3.2 shows the example of backup topologies. Links in red are called protected links and the backup topology provides detour routes on the failures of those links. Every link is required to be a protected link in at least one backup topology and backup topology is defined

as a connected graph that does not contain the protected links. In this way, we can achieve fast recovery against any single link failure by using backup topologies.

In this paper, we adopt Openflow protocol [31] which can provide the programming flexibility of network to realize our IP fast rerouting forwarding in SDN. Combination of both optimal path discovery and fast reroute mechanism is known to provide the best solution for network service restoration [32].

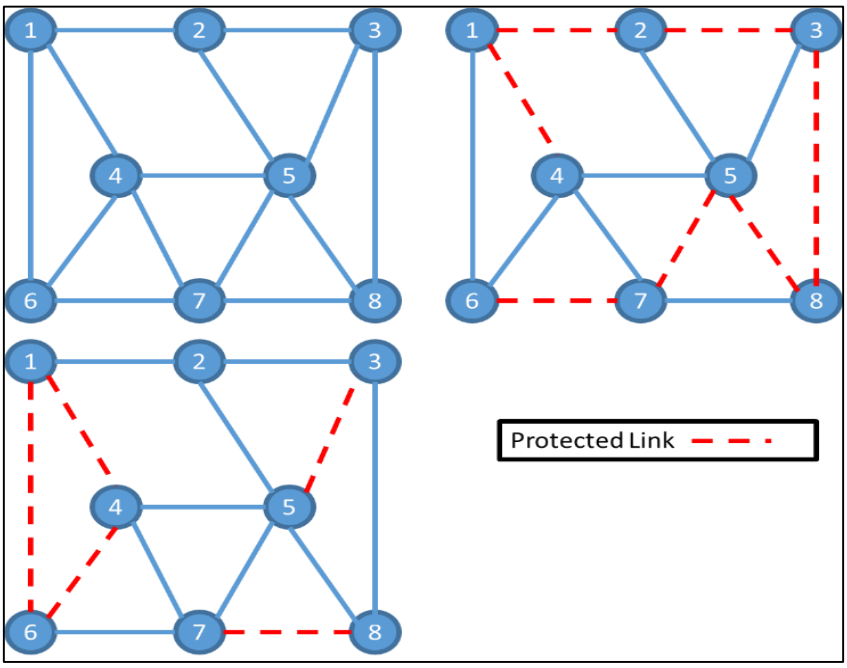

Figure 3.2. Example of backup topology

3.3.2. Optimal Path Calculation

We define optimal path as network load over a path as the *average rate of bytes* that goes through the link and correctly received by the other node that attached to it. Therefore, the link load utilization is expressed as the current link usage over the maximum link capacity and is a number from zero to one. When the utilization is zero it means the link is not used and when it reaches one it means the link is fully saturated [33]. We use Load Sensitive for Software Defined Networking (LSSDN) terminology to denote the proposed load sensitive metric at each link.

To get the link load utilization, the formula are as follows

$$\left(\frac{throughput}{datarate}\right) * 100 \quad (1)$$

The *throughput* is measured by the sum of incoming and outgoing bytes. In Openflow, the needed information (*byte count*) is available from Openflow switch port counter [34]. To measure the link load utilization, the Openflow controller needs to monitor all switches port traffics, cache all of the number of incoming and outgoing bytes through their interface for further calculation.

Assuming that L links are available between two nodes. The bandwidth of i^{th} link between $node_m$ and $node_n$ is $B^i_{m\to n}$ (i=1,…L). If the total number of bytes going through an interface denote as T, then we can calculate the traffic utilization as follows. We define the link load utilization of a node interface, U over $link_i$ from $node_m$ to $node_n$ as the *current total of throughput* through the interface at certain time over the *link transmission rate*. The formula can be simplified as below.

$$U_i = \left(\frac{\sum(T^i)_{m\to n}}{B_{m\to n}}\right) * 100 \quad (2)$$

The path weight of the LSSDN metric is defined as (consider end to end path including H hops),

$$\omega U_i = \sum_{i=1}^{H} U_i \quad (3)$$

Note that the LSSDN metric given in (3) is under the assumption that all the packets can continuously go through all the path hop-by-hop without any node or link failure.

Let vector [$B^i_{m\to n}$, $T^i_{m\to n}$,L] is characteristic of link between $node_m$ and $node_n$. $B^i_{m\to n}$ denote the bandwidth of i^{th} channel between $node_m$ and $node_n$, L is the number of available links and $T^i_{m\to n}$ is the total throughput of i^{th} link between $node_m$ and $node_n$. For a given network of $G\ (V, E)$, and the source node N_s and destination N_d, the LSSDN algorithm include the following steps:

Step 1: Calculate the bandwidth capacity ($B^i_{m\to n}$) for every link in network $G\ (V, E)$.

Step 2: Calculate the total throughput (in *bytes*) over a link $T^i_{m\to n}$ for every link in network $G\ (V, E)$.

Step 3: Calculate the weight ωU_i for every link in *G (V, E)* according to Eq. (3)

Step 4: Use Dijsktra Algorithm to find the smallest sum of weight in the paths of *G (V,E)* from node N_s to node N_d. The details of LSSDN are given in Figure 3.3.

Input: $B^i_{m\to n}$, $T^i_{m\to n}$, *(i=1,...L);*
V={v_1,v_2,*...,*v_n*} :The set of nodes;*
$N_s \epsilon S$*: source node;*
$N_d \epsilon V$*: destination node;*
for j=1 to N do
  for k=1 to N do
    for =1 to M do
find $B^i_{m\to n}$ *;*
      find $T^i_{m\to n}$*;*
calculate ωU_i*;*
    end for
  end for
end for
S: The current set of nodes (from N_s *to* N_d*.) which has smallest load path*
$T(V_i)$*: The current sum of link load of the links on the smallest weight path from* N_s *to* N_d*.*
for =1 to N do
$T(V_i) = \infty$;
$T(N_s) = 0$;
$S = \emptyset$;
end for
while $N_d \in S$
{
$u=v$; *//*$V_i \in S$ *and ( )* $T(V_i)$ *is smallest load in all nodes in V→S*
$S = S \cup \{u\}$;
for all $v_k \in V \to S$
$if\ \left(T(v_i) + \omega U_{m\to n} \leq T(V_k)\right)$
$T(V_k) = T(V_i) + T(V_k) + \omega U_{m\to n}$;
}

Figure 3.3: Smallest Link Utilization Path Selection Algorithm

Besides the update frequency, the number of transmitted and received bytes' information affects the estimation of link utilization for the LSSDN metric. The number of throughput changes instantaneously. If we use the value directly for link utilization calculation, frequent rerouting might have occurred. To avoid the problem, we maintain a weighted average link utilization in the controller, denoted as $\overline{U}$ and controller use this weighted average value as

the backlog information instead of instantaneous sample value for the LSSDN computation. Specifically, the controller samples the instantaneous throughput according to a schedule, and let U_n denote the n^{th} sample. The average link load utilization, $\overline{U}$ by incorporating the instantaneous link utilization U_n, according to the exponential weighted moving average scheme [10], is

$$\overline{U} = (1 - \alpha)U_{old} + \alpha * U_n \quad (4)$$

$where\ 0 \leq \alpha \leq 1$ //U_n = n^{th} sample

To show the need to include link utilization as a metric in SDN forwarding decision, we demonstrate the relation between link utilization and the input load using simple simulation. In the simulation, we tried to vary the aggregate input load traffic and measured the usage of the bandwidth. To simplify the network, the packet size was set to 1000 bytes and the link data rate was set to 1Mbps. The chart as illustrated in Figure 3.4 is to emphasize the impact of the link load utilization on forwarding packet to their destination. From the figure, it clearly shows that the link load utilization is proportional to the traffic that sent through the link. It is observed that the bandwidth utilization is increase linearly with the input load and then it gets saturated as the input load reach approximately 800 Kbps. When the link is saturated, the link load utilization is almost constant even though the input load increases and it is happened because the available link bandwidth is almost fully utilized. The higher the value of link load utilization, the lesser traffic can be send over the link.

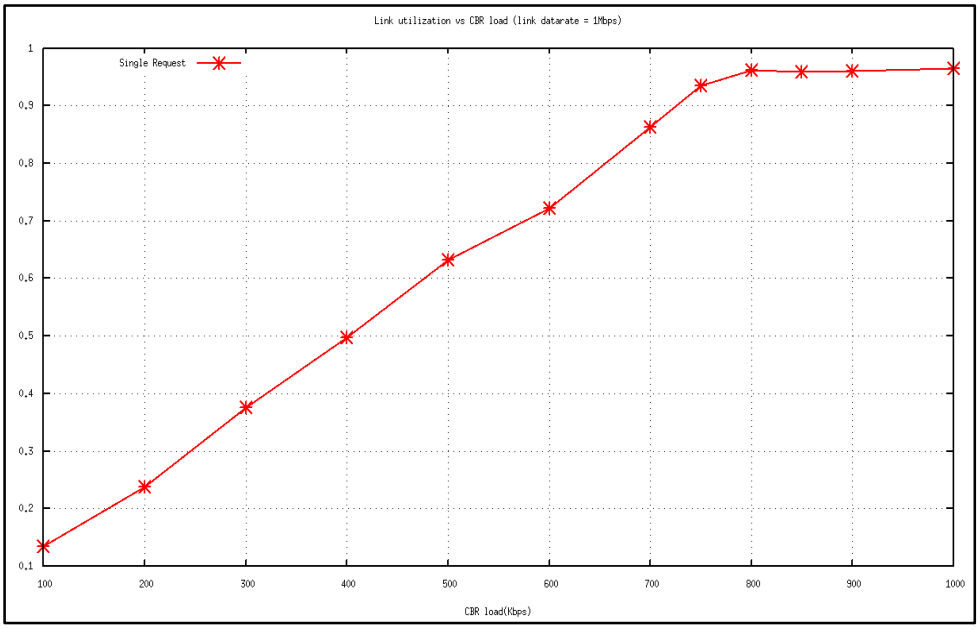

Figure 3.4. Link utilization as a function of input traffic

We also carried another simulation to study the capability of a link for to tolerate more traffic at different link load utilization. We simulate 5 pair of nodes exchanging data with another and we can derive the relation between link load utilization and delay of packet. As illustrated in Figure 3.5, we observed that the packet delay time increase dramatically when the link utilization starts to climb at 90 percent of link utilized. This shows that the need to consider the link load is vitally important at high link utilization but its effect may also be ignored when the link load in under-utilized.

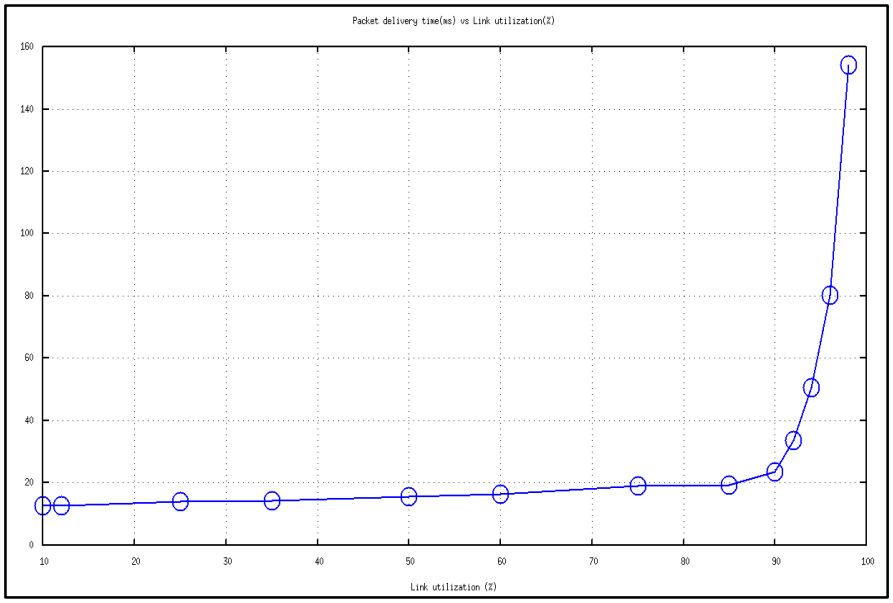

Figure 3.5. Packet delay time as function of link utilization

3.3.3. Update Frequency

Our proposed LSSDN forwarding metric can be viewed as a load sensitive metric as it is heavily depending on the switch port information. The Openflow controller is require to perform recalculation by updating the traffic status to avoid usage of the congested link in the network. To balance the trade-off between performance and the overhead, the route update frequency is a critical factor. More frequent updates of network state will introduce unnecessary overhead. On the other hand, large gap of update frequency will prevent the route from timely tracing the network status, and the network performance may have dropped. We adopt the multipart message provided in Openflow feature which use to encode request or replies to or from the controller to switches. We simulate the feature to get port statistic for all of the switches to calculate our proposed link load metric in the controller. In our proposed algorithm, the message collect *byte count* information to measure the packets going in and going out through a particular port. In our simulations, we set the time for the Openflow switches to update the controller with the needed information automatically every *5 seconds*. After the controller receive the new information it will perform recalculation based on our proposed metric to define the most optimal next forwarding path. In Openflow, the controller able to modify the existing flow entries *action* field that installed in the Openflow switches via flow table modification message that modify all flow that match.

3.3.4. The Design of the Mechanism

In our proposed design, one single backup path for every main path in switch flow table is pre-configured. Once the controller path calculation algorithm decides the main and backup path, it will install both of them to correspondence switches. As in Table 3.1, the main and backup path is differentiated by *priority* field where main path has higher precedence than backup path, thus every incoming packet traffic that match the criteria will be forwarded to their next destination using the flow entry with the highest priority field.

Table 3.1. Switch Flow Entry

Match	Action	Priority
S-D	To port 1	*High*
S-D	To port 2	*Low*

Whenever there is any port in the Openflow switches is added, deleted or modified, OFPT_PORT_STATUS message is sent to central controller indicating any changes of the port status. To support fast failover, a local controller for every Openflow switches is proposed. Instead of sending the port status to central controller only, we configure the port status message to also send the port changes information to the local controller. The central controller responsible to perform the path calculation and decide the main and backup path of a particular flow while the local controller perform the local rerouting to backup flow entry. The local controller network restoration does not depend on the centralize network state.

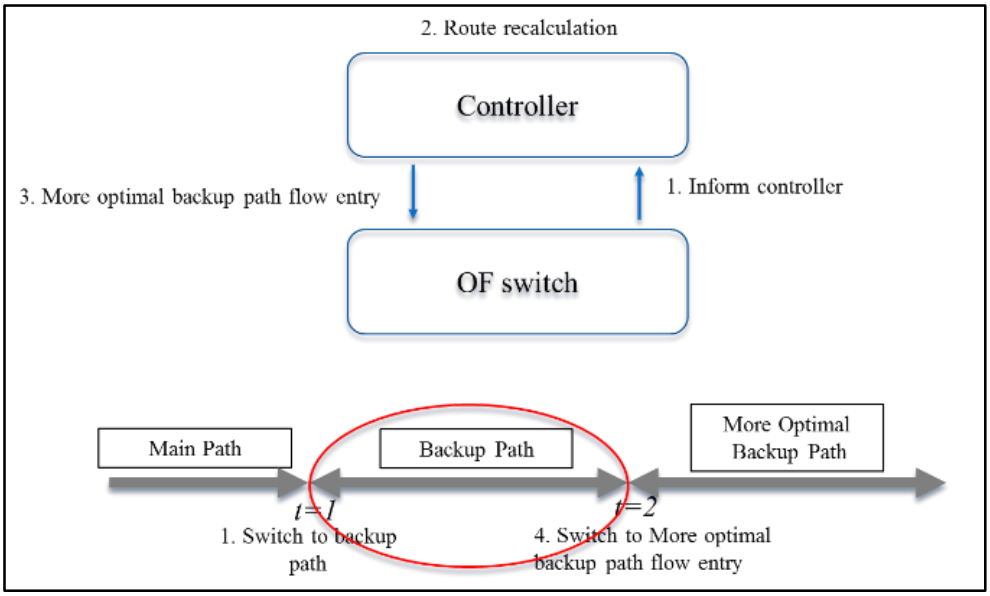

Figure 3.6. More Optimal Path Fast Reroute Workflow

As in Figure 3.6, when a link fail, at *t=1*, the local controller will perform function to switch the incoming traffic to use the pre-defined backup path. While at the same time, the port-status message is sent to the controller. When the central controller receives the message, it will perform another path calculation to decide the best available path for that particular flow, excluding the unusable port that reported by the affected switches for their next calculation. After the central controller decide the new main and backup path for the affected flow, it will install both flow entry into affected switches, at *t=2*. In our work, whenever the central controller receives the port status message, it will recalculate path and install two entries back; the main and backup path with different *priority* level as explained before. We describe the proposed working mechanism in the following section with example given.

3.3.5. Link failure scenario

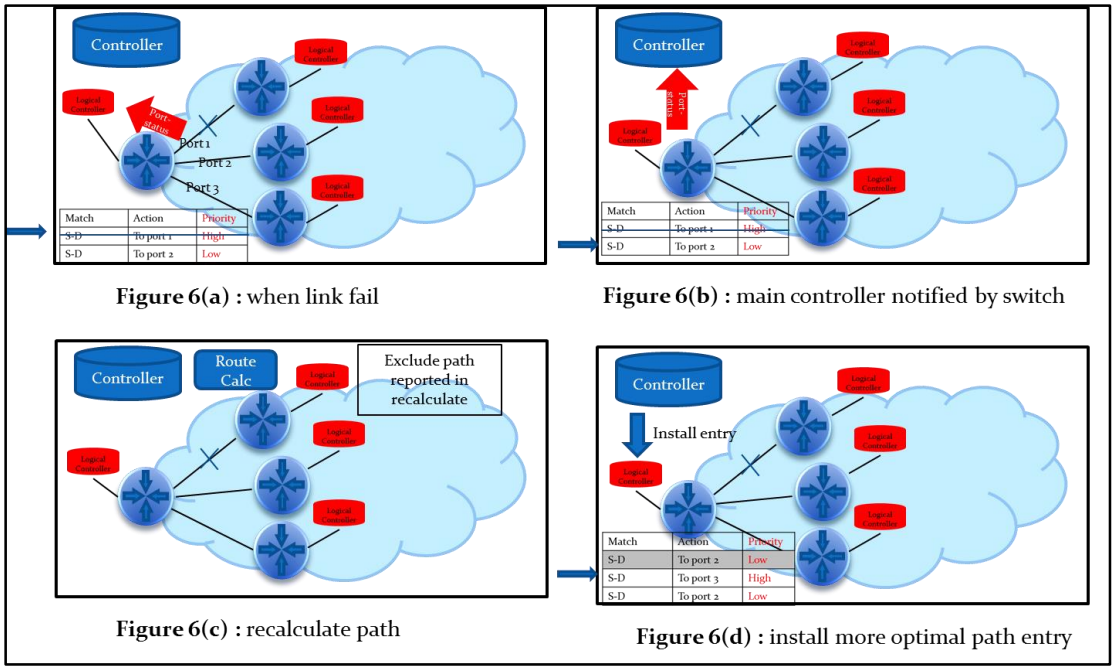

Figure 3.7. Example of single link failure scenario

This section describes our fast rerouting mechanism using Openflow protocol. We illustrate the steps to achieve our objective using example in Figure 3.7. In normal condition, entries in the flow table specify the outgoing interface for every flows in *action* field. At the same time, each switches also stored backup path for all flows that use main path in the flow table entries. In Figure 3.7(a), when the local controller detects a link failure, we configure the local controller to receive the OFPT_PORT_STATUS message from the affected port. The local controller finds the list of flows in their database that use the failed port and begin the operation to delete the affected entry from their flow entries lists. Local controller performs delete operation for all flows that use the affected port as their forwarding interface. Note that the deleted entry is main path with the highest *priority* precedence field. By doing this, fast rerouting is performed without need to wait for global route calculation from the centralize controller. Thus, the incoming packet will autonomously reroute to backup path entry that previously installed by the central controller. Rerouting processes using the local controller are only performed at the switch that detects the link failure.

In Figure 3.7(b), when a link fail, the OFPT_PORT_STATUS message was also sent by the switches to the controller. It is worth to note that during this process, the subsequent packet that previously use main path entry for forwarding has already routed using the backup path since there is a match in earlier predefined flow entry. Once the central controller receives this message, it performs same initial step like the local controller did. Central controller holds the global routing information of the network. The central controller will calculate the new path for the affected flows. This time, once the central controller identified the flows that use the affected port as their forwarding port, it removes the affected port from the new path calculation. By doing this, the central controller ensure that the new path will not include the fail port to route the flows to their next destination. The central controller again performs route calculation process two times. The second time calculation will not include the first port that had been identified as main forwarder interface. In Figure 3.7(d), once the central controller had identified the new main and backup path for the affected flows, it installs both entries into respected switches flow table. In this example, the central controller decides the best path for the affected flows is through port 3. Then it installs the entry with high *priority* into the respective switches. In Openflow, whenever there are two identical entries but with different *priority* precedence, the flows will automatically always use the entry with higher *priority* to reach their next destination. In this case, the flows will autonomously reroute to port 3.

3.4. Performance Evaluation

To show the ability of our fast rerouting mechanism, we evaluate the proposed mechanism using two parameters which utilizes the Openflow framework. Our objective is to achieve the main fundamental for fast rerouting; able to reroute within 50 *milliseconds* [35]. We also compare our proposal to the related works [26] which require the central controller to acknowledge the link failure and make decision to install the backup path into the switches and [27] which also implemented pre-defined backup path without the centralize controller involvement.

The simulations are performed using Omnet++ with Openflow protocol enabled [36]. Openflow component are integrated into the Omnet ++ network simulation environment with the support from INET framework [37]. For the network topology, we use power-law model [38]. In this model, most of the network nodes have small number of connected links, while a small number of network nodes have a large number of connected links. We simulate

incremental number of network nodes ranges from 20 to 200 nodes and each node is configured to have four connected links on average.

3.4.1. Restoration Time

The first parameter is restoration time. To find the restoration time for the affected flows to recover from a link failure, the link failure detection, T_{detect} and updating flows to use backup path, T_{update} are formulated as follows:

$$T_{local} = T_{detect} + T_{update} \quad (5)$$

For the failure detection T_{detect}, we define as the time for the logical controller to receive the OFPT_PORT_STATUS message before make decision to do the restoration process. T_{update} is define as forwarding table update which force the affected flows to reroute using the pre-configured backup path. From the equation (5), we are able to calculate the time taken for the local controller to take action to reroute flows locally.

To find restoration time in [10], they formulated the restoration time as follows:

$$T_{central} = T_{detect} + T_{calc} + T_{update} \quad (6)$$

T_{detect} is the time when the failure notification message is sent to the central controller. When the central controller is notified by the switches about the failure, the central controller will find the affected flows and the new path are calculated, T_{calc}. After that, the switches receive the flow-mod message from controller to update the flow entry. When the flows are redirected to new path, T_{update} is calculated.

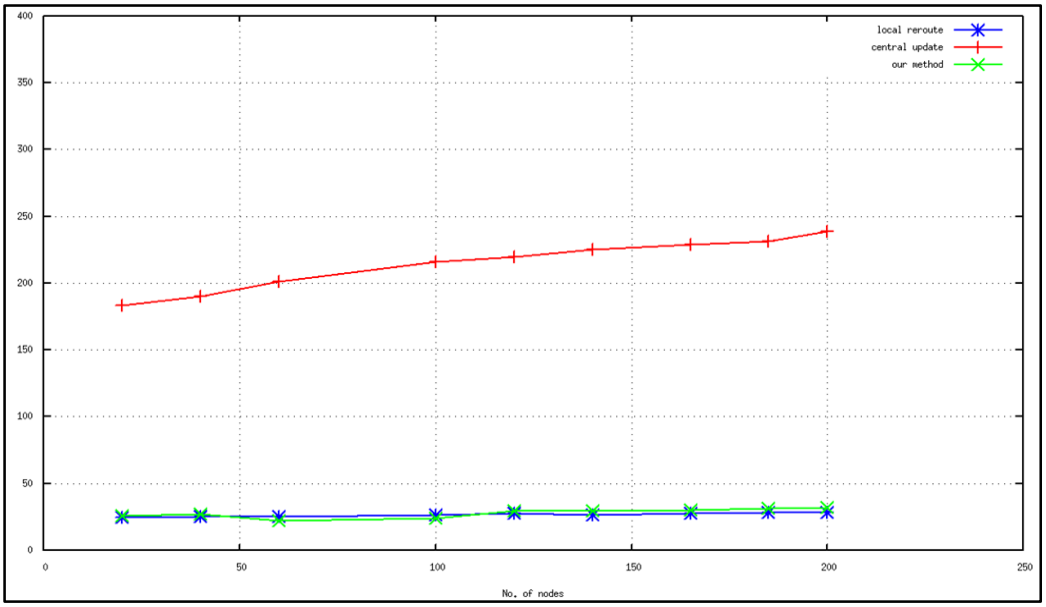

Figure 3.8. Network Restoration Time

Figure 3.8 shows the restoration time with three conditions. The first require switches to notify the central controller for the link failure event and no pre-defined backup path is configured in switches. The second case use pre-installed backup path and perform local rerouting when there is a link failure without the involvement of central controller. The third which is our method, is a hybrid mechanism where both local and central controller contributions are required when a link is fail between devices. Whenever a link fail and the switches need to involve the central controller for rerouting decision, clearly the method does not achieve the 50 milliseconds carrier grade restoration time. From the figure, the restoration time takes approximately 200 milliseconds on average for the network to recover from link failure. In our method, the restoration time achieve below 50 milliseconds carrier-grade network where it takes 35 milliseconds on average to restore from a link failure. Our fast rerouting, which performs rerouting locally, does not require the global acknowledgement. Thus it always achieves below 50 milliseconds restoration. While for the case of local rerouting without the intervention from central controller, the method also achieves below 50 milliseconds restoration but the rerouting to predefined backup path is not always an optimum solution. We show the drawback of this kind of method in the following section.

3.4.2. Average Number of Hops

In this section, we compare the average number of hops taken over paths from each source and destination pair in the network. We find the average value for the following three cases, (i) flows are rerouted using most optimal link decided by the central controller without any link failure (ii) flows are rerouted using pre-defined backup path decided by the central controller when link fail happen and then (iii) flows are rerouted through the best recomputed path when there is link failure. In this work, we assume that a single link failure does not leave the network disconnected.

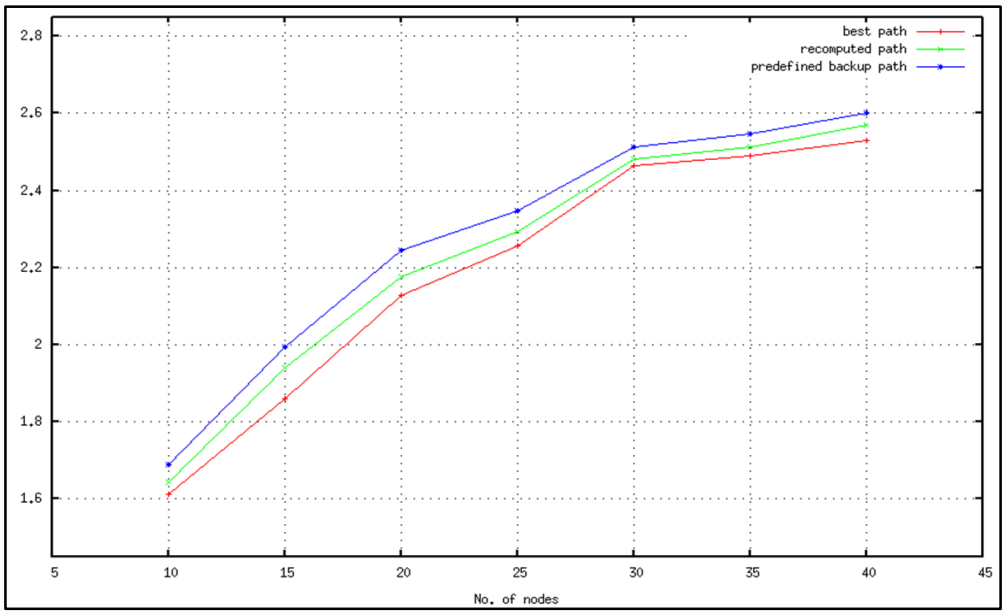

Figure 3.9. Average Hop Count

Average number of hops for all of three cases as mentioned above are plotted in Figure 3.9, for a network topology with increasing number of network nodes ranging from 10 to 40 nodes. As can be seen from the figure above, clearly the average number of hops for our method which is the case (iii) are slightly lower than flows that rerouted to pre-defined backup path when the link fail. It shows that when all flows are rerouted to the recomputed path when the link is fail is able to reach their destination faster. However, in turn the average number of hops for our method is higher than using the main optimal path when there is no link failure.

In the case of a network link failure, it is expected that the network size decreases due to the lost link availability. The path availability is always dependent on network load and it is

varying inversely with average number of hop count. This is because the number of available path will decrease because of the link, thus give impact on the routing performance overall.

3.5. Discussion of Implementation

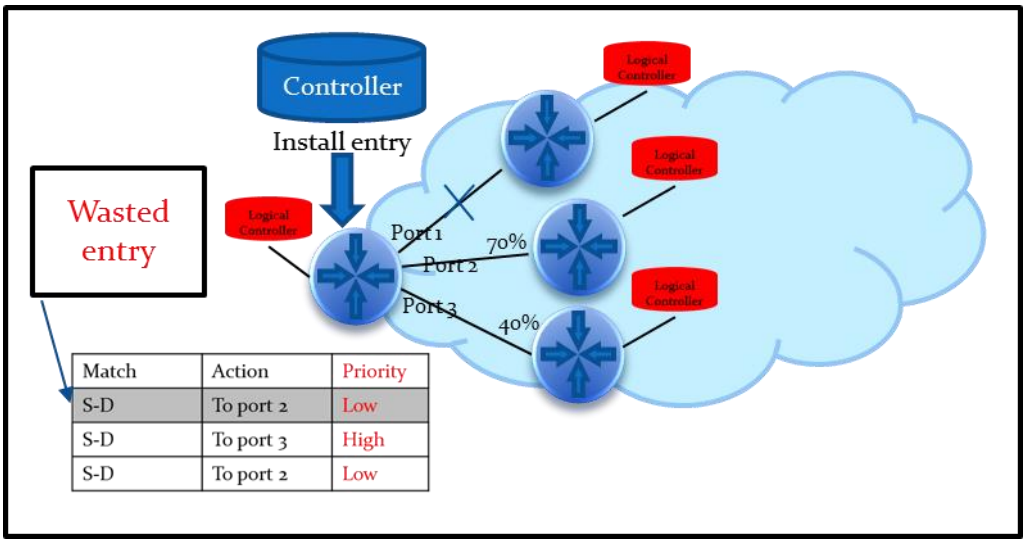

Figure 3.10. Wasted Flow Entry

Our method always ensure the central controller make decision to elect both main and backup path during the path calculation process. This is to support the fast rerouting to all flows that go through the Openflow switches. While it is the best to have pre-defined backup path for all flows in switches, our method come with an effect to the switches performance. Referring to example in Figure 3.10, the local controller had previously updated the affected flows that use the port 1 as their forwarding port rerouted to pre-defined backup path which is port 2. Note that during this time, the central controller will update the topology with the new information. In this example, central controller had identified that port 3 is more optimal than port 2, thus the central controller elect port 3 as the main path for the flows to reroute with. Then the central controller installed two flow entries into the affected switches with different priority *precedence*.

Once both entries are installed, the affected flows autonomously rerouted to their next destination using port 3 as their main path instead of port 2. Previously, local controller had performed local reroute to use port 2 as the next destination. Now, it is replaced by the central controller that force flows to use port 3. In this case, they will be some redundant flow entries in the particular switches which has the same *match* and *action* criteria which is forwarding

those flows to use port 2. It is clearly a waste of unused number of flow entries in the switches which might lead to flow table explosion. In our case, we simply let the unused flow entries to expired and deleted automatically from the switch flow table entry.

3.6. Summary

This work proposed autonomous fast rerouting to more optimal path using Openflow. We combine both ability of central and local controller for the recovery processes. The scheme relies on pre-configured backup path and also a global update of topology from the central controller to reroute to most optimal path. When there is a link failure, the local controller autonomously reroutes affected flows to their next destination without the need of help from the central controller.

By utilizing both, in the experimental demonstration results shows our method achieves below 50 milliseconds restoration time which is crucial for the IP services infrastructure to be a carrier-grade network. It also shows that this method also requires less number of hops for the affected flows to traverse through the available links after a link failure.

Chapter 4

4. Adaptive Query Rate for Anomaly Detection with SDN

4.1. Introduction

Flow-based method lures the interest from researchers for the network analysis of high-speed networks. With ever increasing load and network usage, clearly scalability is paramount issue to be tackle. In order to perform analysis, statistic features need to be recorded and it expose to unrestraint processing capacity and network bandwidth. For this reason, the accuracy of anomaly detection as well as the need to balance the trade-off between the overhead and the accuracy is crucial.

The emergence of SDN that promise the simplicity in managing networks seems to be the future of the current Internet architecture. In fact, several organizations have adopted SDN in place, most notably Google [39]. By separating the control plane that orchestrated by logical network controller platform and data plane as a forwarding drive, SDN, however, expose to network security threat that can be commence from outside as well as from inside attack of the network.

Network traffic statistics data has been used as inputs for anomaly detection as security analysis is critical for organization or network providers. In this paper, we emphasize on the issues related with the effect of sampling on anomaly detection problem. Network anomaly detection techniques [40] is based on analysis on network traffic and the characteristic of the dynamic statistic features in order to detect network abnormalities quickly and accurately. It is paramount to balance the trade-off between accuracy of anomaly detection and overhead introduced from the flow traffic measurements (e.g.-scalability). Sampling decision should have some intelligent ability to address some of the requirement such as to reach low false alarm and low computation convolution objective. In this thesis, we proposed adaptive sampling decision for the controller to capture the most dominant service type of DDoS attack in the network where we aim to provide accurate anomaly detection while at the same time lowering

the number of false reported alarm. Our sampling decision method ensure malicious flows that come from commonly used attack service port is given higher priority than others, thereby addressing the accuracy parameter.

Previous related work is discussed in Section 4.2 then the following Section 4.3 gives a brief detail of our adaptive query for anomaly detection purpose framework. Our flow collector method is presented in Section 4.3.1 followed by Section 4.3.2 where we explained important network features that are crucial for our anomaly detection. The methodology and algorithm of our proposed sampling decision technique is explained in Section 4.3.3. The performance evaluation results are shown in Section 4.4. Finally, we conclude our discussion in Section 4.5.

4.2. Related Works

Previously, many researchers started to give attention to the impact of sampling method on anomaly detection. [41], the author has signified that the packet sampling degrades the effectiveness of anomaly detection and dramatically increase the false positives. In [42], the author compares two type of sampling which is random flow sampling [43] and sample-and-hold technique [44]. The result shows that random flow sampling performs best for anomaly detection. In [45] the authors used flow-based metrics such as the number of source IP addresses for the detection and proved that the accuracy and performance of the anomaly detection depends mainly on the sampling rate applied and the author also proved it is less dependent on the sampling technique used.

Intrusion detection problem has not drawn so much intention from researcher despite substantial amount of work in Openflow. The eminent work regarding anomaly detection in SDN is reported in [46], [47] and [48]. In [46], the author utilizes the Openflow architecture for detecting the Distributed Denial of Service (DDoS) on the data plane. Periodic sampling is used for flow statistics collection retrieval and Self-Organized Map has been exploit for the intrusion detection classification. In [47], the author uses multiple type of anomaly detection algorithm in their research test where the author validates their algorithm in Small Office/Home Office (SOHO) environment.

The author utilized Openflow for detecting network security problem close to the source of abnormality using the idea of decentralization control of the network devices. The author also uses periodic sampling for the flow statistics collection. Contrast to previous work, author

[48] decoupled the controller communication channel with Openflow switches where the *sFlow* flow statistics collection method is used and the native Openflow communication channel is used only for the forwarding purposed separately. The experimental results show significant reduction in flow table size and the control plane load. However, the method used by the author increase the false positives percentage in the intrusion detection.

4.3. Adaptive Query Methodology

To provide scalable and accurate sampling decision, we considered two important parameters which is the anomaly detection and the traffic measurement. In our work, two important method applied to the incoming packet into the controller. At first, the step is to be able to classify the incoming packet is either anomalous or normal traffic while the second step is responsible for our flow sampling decision. Our proposed architecture has been generalized in Figure 4.1 where it contains 4 main components namely as *Flow_Collector, Flow_Processing, Sampling_Decision* and *Forwarding_Logic* modules.

The components are developed to be part in the POX controller [49] that we use for our simulation. In general, *Flow_Collector* module collect all active flows in the network while the *Flow_ Processing* carried out flow-level analysis to find the anomaly behavior from the network flow inspected. The module also responsible to provide attack event and also the type of service of the incoming packet that has been identified. This two information are sent to the *Sampling_Decision* module for the sampling decision making. The consideration of both the type of service port number and the condition of traffic, our sampling decision is defined in such a way that any malicious traffic service is given higher priority to be queried and sampled. This is to ensure to improve the accuracy of the anomaly detection in SDN. We describe each of the components function precisely in the following section.

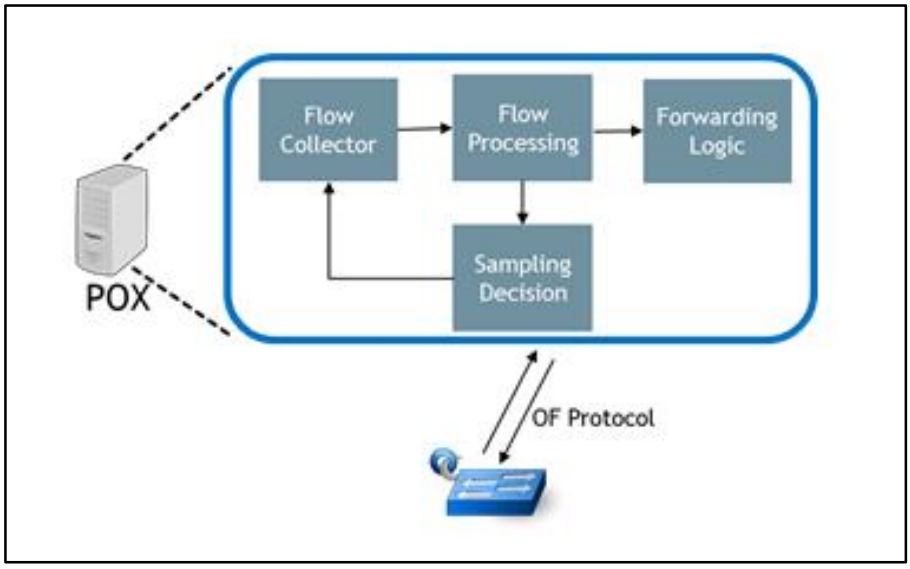

Figure 4.1. Main Skeleton of proposed architecture

4.3.1. Flow Collector

It is important to describe how Openflow mechanism works when a new packet of a particular flow arrive in the switch. First, when any new packet reaches the switch, matching process operation are performed. The switch will check whether the flow entry that installed previously match with the incoming packet. If there is a match, the packet will follow the action set for the related flow entry. Otherwise, the switch will send the packet to the controller for further decision making. The controller receives the header information from a control message namely as *Packet_In* from Openflow switch which is result from unmatched flow in the switch flow table.

Furthermore, the controller will perform necessary decision and install the rules for the flow into the switch flow table as a flow entry by using *FlowMod* control message. The switch then forwards the packet to their destination and subsequent packet of that flow are forwarded without interrupting the controller. In our proposed work, we record the flow information in an active flow table in the controller that consist of all current active flows in the network. Our structure of *Flow_Collector* can be described as in Table 4.1.

Table 4.1. Active Flow Collector Table in Controller

Flow	**Counter**	
$flow_l$	Packet Count	Total Bytes

The table contains 2 main field namely as *Flow* and *Counter*. The *Counter* field is further divided into *Packet_Count*, *Total_Bytes* and *Duration* field. After the controller receive the incoming *Packet_In* and perform the rules and forwarding decision, our method adds the new information to a list of active flows, $flow_l$ (in *Flow* field) which is associated with the incoming port of flow. Then, controller installs the associated rules as a flow entry in switch flow table via *FlowMod* message. When the flow information has been stored, the controller will send *ofp_flow_stats_request* control message to every switch that has information about the active flow in order to update the controller with their necessary statistic. The statistics are as follows: we record the number of packet per flow, total bytes per flow and duration of the flow. Controller kept previous time-window value record for the *Counter* and perform calculation when get new statistic to update current time-window value. Our active flows *Counter* are updated in controller when the Openflow switch reply to request message.

4.3.2. Flow Processing

As the controller manages the network in centralized fashion, it was able to monitor all switches and further do analysis on their traffic for the anomaly detection problem. Our *Flow_Processing* module contains 2 important sub-modules name as Feature Extractor and Anomaly Classifier. The details of the sub-modules are explained in the following section.

4.3.2.1. Feature Extractor

Selection decision of different set from network traffic feature sets to be used is common problem in anomaly detection. As an example, various types of features are widely known such as detection based on packet headers, application layer protocol or content byte streaming. In Openflow, the controller has the ability to check the packet header information. When the switch cannot match the incoming packet with their predefined flow table, it will forward the packet header information to the controller for further action.

Our *Feature Extractor* module extract flow information from the *Active_Flows* module which are vital for our anomaly detection classification. All of the important features are derive from the packet header (source port, destination port, source IP, destination IP and protocol). In Openflow, we can get that information from control message namely as *Packet In*. The important features are grouped into *5-tuples* flow information in a hash table. In this stage, all

features selected is most likely that influence the judgment to classify network traffic as normal or as an attack.The *5-tuples* features are as follows:

1. *Flow Byte*, B_l - the number of bytes of a particular flow capable to provide us a useful information for anomaly event in network, such as port scan, and it is normally small in size in order to increase the coherence of attacks.
2. *Flow Size*, F_l - IP spoofing is one of main example of DoS attack that make the task to detect the true source of spoofing is nearly impossible. The normal operation of spoofing usually generates flows with a small number of packets. This contradicts from normal network traffic where it usually generates a large number of packets for a particular flow.
3. *Number of different flows to same Destination IP*, $n(D_{IP})$- Flood attack are created to consume the resources of victim host and usually will generate a high number of flows. This feature will calculate the number of flows to same victim's destination IP address.
4. *Number of flows to different Destination Ports*, $n(D_{port})$- port scan attack is a process that send requests to a number of server port addresses on a particular host. The aim of this attack is to penetrate an active port on that host and any large number of different destination port indicate the abnormality and shows higher possibility of the network are under attack.
5. *Number of different Source and Destination pair*, $n(SD_{pair})$ [1]– this feature able to spot the port and network scans as well as distributed type of attacks, which spike the number of source and destination pairs. We define this 5 features as $X_t = f\left(F_l, B_l, n(D_{IP}), n(D_{port}), n(SD_{pair})\right)$ where X_t is the observed value at time t. Furthermore, this 5 features selected vector fed to our *Anomaly Classifier* module. We purposely choose this 5 features since the number of packet and bytes of a flow allow us to detect anomalies in traffic volume while the others will show increment values in the number.

[1] We consider port numbers and IP addresses in utilizing this feature

4.3.2.2. Anomaly Classifier

There are many notable algorithms that has been successfully proven to classify network traffic for anomaly detection [50,51]. In our simulation, we adopt K-mean algorithm as our anomaly classifier for simplicity and brevity purpose. This algorithm has the ability to learn and detect anomalies from the audit data without the intrusion signature which is usually provide by the security expert. The advantage of this machine learning algorithm is it can automatically identify groups of similar objects in the training dataset. This clustering algorithm groups multiple objects into predefined K disjoint clusters.

We summarize the steps of performing this algorithm in the followings:

1. Define the number of *K clusters*. In our anomaly detection problem, we set the ***K=2*** where we assume that legitimate and anomaly network traffic features are from different cluster in space.
2. Initialize the randomly chosen *K* clusters and set to be as *centroid* (center of cluster).
3. The calculation process begins to find the distance from each objects to all centroids using distance function method where the algorithm continues to read each objects from the data set and assigns it to their nearest cluster.
4. Recalculate/iteration process is done after every new objects insertion to the algorithm to get the new cluster centroids.
5. Step 3-4 are repeated until the centroids do not change.

In this algorithm, distance function is required to calculate the similarity between two different objects. The following equation is Euclidean function which is commonly utilized to compute the distance where $a = (a_i, \dots a_m)$ and $b = (b_i, \dots b_m)$ are the two input vector with m features.

$$d(a,b) = \sqrt{\sum_{i=1}^{m}(a_i - b_i)^2} \quad (1)$$

Using this function however, the features must be normalized first since the features are usually measured with different metrics. For the evaluation of our proposed adaptive anomaly detection method, we use weighted Euclidean function as in the following equation:

$$d(a,b) = \sqrt{\sum_{i=1}^{m} \frac{(a_i - b_i)^2}{s_i}} \quad (2)$$

The s_i is weight factor and empirical normalization and of the i^{th} feature. The classification of the network traffic is done by the controller where it utilizes this algorithm to detect the anomaly. Whenever the *5-tuples* features are classified as attack, alerts are notices to an administrator.

4.3.3. Sampling Decision

Accuracy and efficiency are two important factors that our formulation for the sampling decision is based on. The effect of polling rate to anomaly detection and traffic measurement derived from accuracy parameter. Higher polling rate is favourable to accurately detect the network traffic abnormality within short period of time. On the contrary, efficiency factor denotes the effect of the polling method to the controller memory and CPU resources. Since high polling rate in the network lead to large number of sampled flows, it is crucial for the controller to have the ability to vary the polling rate so that it will not drain the resources. Therefore, our sampling decision must have the ability to dynamically adjust the *polling rate*, $S[t_i]_l$ based on previous stated two parameters.

4.3.3.1. Anomaly Detection and Traffic Management

According to a report from Akamai [52], the concentration of attack traffic is increased during the second quarter of 2013 where the increased concentration was driven by indicatively increases in attack volume targeting Ports 80 (WWW/ HTTP) and 443 (SSL/HTTPS). For our objective of traffic measurement accuracy, we classify common attack port as the commonly used source of attack port service over the overall flows population.

We favor sampling to flows that had been attack with commonly used source port. Consider a set of *m* flows of various source port service, src_{port}=$\{src_{(port)l}: l = 1, 2, \dots m\}$ where $l \subset m$. If the source port service of a particular flow, $src_{(port)l}$ is port *80*, we assume that flow is using commonly used attack source port service. Priority is given to the network traffic based on severity level and the service port of the particular flows where we define any attacked flow with source port service is port 80 is given highest priority. Given a flow with attack probability/event $\hat{A}_l$ and the source port service of the flow is $src_{(port)l}$, the prioritization can be expressed as following equation:

$$Pr(\widehat{\boldsymbol{A}}_{\boldsymbol{l}}) = x_{\boldsymbol{l}} * \omega_{\boldsymbol{l}} \tag{3}$$

We define $x_{\boldsymbol{l}}$ as $\{(\widehat{\boldsymbol{A}}_{\boldsymbol{l}}) * src_{(port)l}\}$ and $\omega_{\boldsymbol{l}}$ is a weight given to the x_l. From the equation above, a large x_l value denote the severe network attack on a flow with commonly used attack port service. Since both information are known parameters, the $\omega_{\boldsymbol{l}}$ value is constructed in such a way that higher value is given to abnormal flows. Thus, we ensure that the flow with the priority is given more precedence compare to other flows. With the network dynamically change from time to time, it is very challenging to determine the exact value of the weight $\omega_{\boldsymbol{l}}$, for that reason and also for the simplicity, we manually define the value of the weight. The appropriate value of the weight of a particular flow can be defined by using any other heuristic algorithm. After the above steps are completed, our *polling rate*, $S[t_i]_l$ for a particular flow are decided as following equation:

$$S[t_i]l \begin{cases} \frac{T}{\alpha} & for\ Pr(\widehat{\boldsymbol{A}}_{\boldsymbol{l}}) \\ \frac{T}{\beta} & for\ (\widehat{\boldsymbol{A}}_{\boldsymbol{l}}) \\ T * \partial & for\ normal \end{cases} \tag{4}$$

For flows with priority $Pr(\widehat{\boldsymbol{A}}_{\boldsymbol{l}})$, we set to poll the statistic information with higher frequency, $\frac{T}{\alpha}$ and for the flow with attack that has lower severity, we poll the flow information lower than the higher priority flow. We leverage the accuracy and scalability of the sampling decision by lower down the poll frequency for legitimate traffic, $T * \partial$. The decision for *ofp_flow_stats_request* scheduling timer are set within predefined minimum and maximum timeout value where $T_{min} \leq \propto, \beta, \partial \leq T_{max}$. The pseudo-code is given in Figure 4.2. Note that our sampling decision favour flows with certain bias criteria where higher priority is given to malicious flows with commonly used attack service port number.

```
1:  Input x_t, Feature : Active_Flows, A, C, control message,
    C=C1 (Packet_In), C=C2 (ofp_flow_stat_request),
    C=C3 (ofp_flow_stat_reply)
2:  Function , S[t_i]l,

3:      CHECK_INCOMING_PACKET_TYPE
4:      if (C=C1)
5:          store flow into A
6:          for all flow
7:          send C2 to flow switch, S[t_i]l = T
8:          end for
9:      else if (C=C3)
10:         for all flow in A, extract x_t
11:         execute (intrusion_detection_module)
12:         then
              if P_r(Â_l) then S[t_i]l at T/α
            end if
            else if (Â_l) then S[t_i]l at T/β
            end if
13:     end if
```

Figure 4.2. Adaptive Poll Pseudo-code

After the controller receive the *Packet In* message, it will send *FlowMod* message to install the flow entry into the related switch. Then, we utilize Openflow standard message type [53] **OFPMP_FLOW_STATS** request which is sent from controller to switches. Our poll scheduling algorithm will start to send message to Openflow switch requesting the flow statistics information. Furthermore, the classification of anomaly is done where the anomaly flow will be marked as $\hat{A}_l$, the sampling decision is made. The process of the sampling decision is simplified for viewing in Figure 4.3.

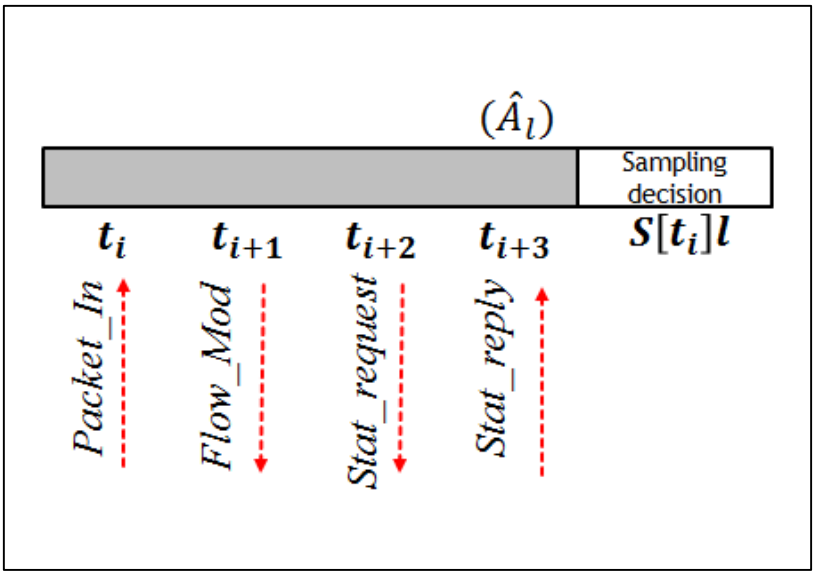

Figure 4.3. Sampling Decision in Timeline

4.4. Performance Evaluation

In this section, we present our experimental setup and performance of our proposed anomaly detection method. We focus on the accurateness level of anomaly detection with our adaptive poll method and perform comparison with static poll mechanism. We also measure the CPU performance for the controller in order to leverage the possible overhead introduced using our proposed technique. We used Mininet [54] to emulate the network attack consisting of Openflow switches, links and hosts on a single machine.

4.4.1. Experimental Setup

In our anomaly detection method, all of the algorithm is implemented on POX controller which is written in Python language. For the simulation purpose, we choose Mininet network emulator version 2.2.0 with software switch availability that support the Openflow standard software switch which is OpenvSwitch [55]. We ran our experiments to emulate the network attack scenario, as well as to train the K-mean classification algorithm, on a system with an Intel core i3 CPU and 8 GB RAM memory capabilities. Figure 4.4 shows the topology setup and the network attack scenario that has been used for our simulation. *Victim* network consist of three Openflow standard switches that connected to POX controller via Openflow protocol channel. We configure the link between *Victim* and *Attacker* network via a gateway with 1 Gbps bandwidth and 20 milliseconds of delay and all other links are assumed to have 100Mbps

bandwidth. The network attack simulated is assumed origin from outside of *Victim* network and all *Victim* host is connected directly to Openflow switch 3 (OFS3).

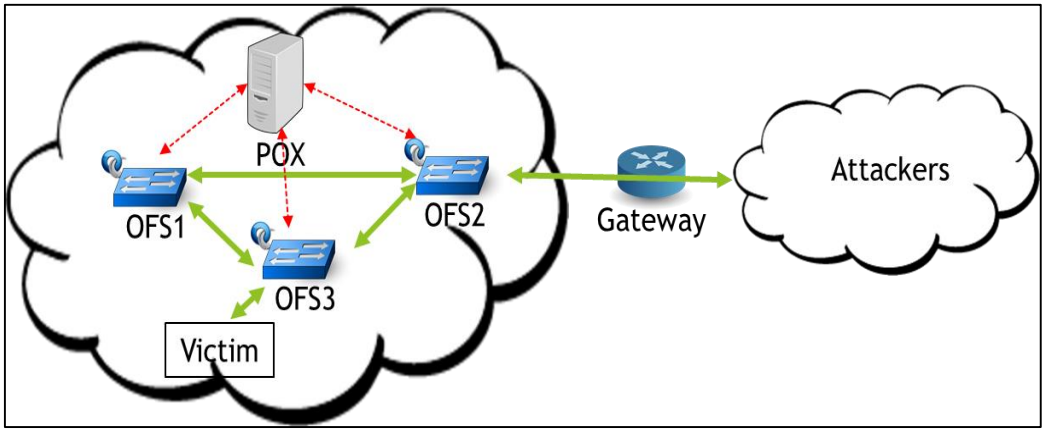

Figure 4.4. Simulation topology.

4.4.2. Dataset and Traffic Generation

In order to simulate high traffic network, we use CAIDA benign Internet trace [56] aiming to evaluate our anomaly detection method with real network environment. Since the dataset size is huge, we extract only 10% from the data which make up approximately 110Mbps of packet and almost 6000 flows, enough to simulate high traffic behavior of high traffic network. This dataset was used to evaluate the accurateness level of anomaly detection with the adaptive method proposed. We use Tcpreplay tool [57] to replay the extracted CAIDA dataset in the Mininet. This tool has the ability to do editing and replaying previously captured network traffic and initially it is design to replay the malicious network traffic patterns to Intrusion Detection/Prevention Systems. For the network attack traces, we utilized Scapy [58], a computer network manipulation tools written in Python. This tool allows us to generate sequence of traffic randomly, thus it can be used to simulate attack traffic behaviour. For port scan attack scenario and to imitate the commonly behaviour of the attack, we generate and injected packet with specific source and destination IP address. Furthermore, the source and destination ports were randomly selected in each packet generated. Next, to emulate the DDoS attack, SYN packets with a set of predefined destination port and IP address, together with a constantly changed and random set of source port and IP address.

4.4.3. Training Time and Traffic Classification

In our anomaly detection, a model that represent the normal behavior of a particular network is constructed. We train the benign CAIDA dataset with the weighted K-Mean algorithm to learn the normal behavior of the data. For the testing phase, we manually inject the attack packet and let the weighted K-Mean algorithm differentiate and classify the attack packet as anomaly. In Table 4.2, we present the training and classification time take by the algorithm to perform task such as training time and classification of the sample. From the 5 data feature set that we used, the training time takes around 7 hours and the classification time takes around 315 miliseconds.

Table 4.2. Dataset training and classification time.

	Weighted K-Mean Training Time	**Weighted K-Mean Classification**
	Hours	Milliseconds
5-tuples	6.37 hours	314 Milliseconds

4.4.4. Accuracy and Anomaly Classification

Three important factor to evaluate our proposed mechanism is considered: (i) average network traffic rate, (ii) the number of attack packet per second and (iii) polling rate as shown in Table 4.3. We used a real 110Mbps Internet dataset derived from CAIDA. We injected attack packets that emulate the DDoS and port scan attack at different packet rate. For our experiment we replayed the benign 110Mbps dataset while injecting DDoS and port scan.

Table 4.3. Parameter values used in experiment.

Average Traffic Rate (Mbps)	**Attack rate (pps)**	**Poll Rate**	
110 Mbps	200,350,500	Every 5 seconds (static)	$S[t_i]l$

Our objective in this experiment is to have a better accuracy in detecting network anomalies by doing comparison using two different kind of network polling rate mechanism. For the first experiment, we manually set the polling rate to collect the network statistic from the Openflow switches at every 5 seconds and the next experiment we tested our proposed $S[t_i]l$ mechanism.

In anomaly detection problem, Receiver Operating Characteristic (ROC) curve is usually used to measure the performance of the method. The ROC curve is a plot of intrusion detection accuracy against the false positive probability. In Figure 5 and Figure 6, we present the ROC curves that we have experimented with two different type of attacks with different polling rate mechanism (Table 4.3). In this first experiment, we inject 200 network attack packet per second.

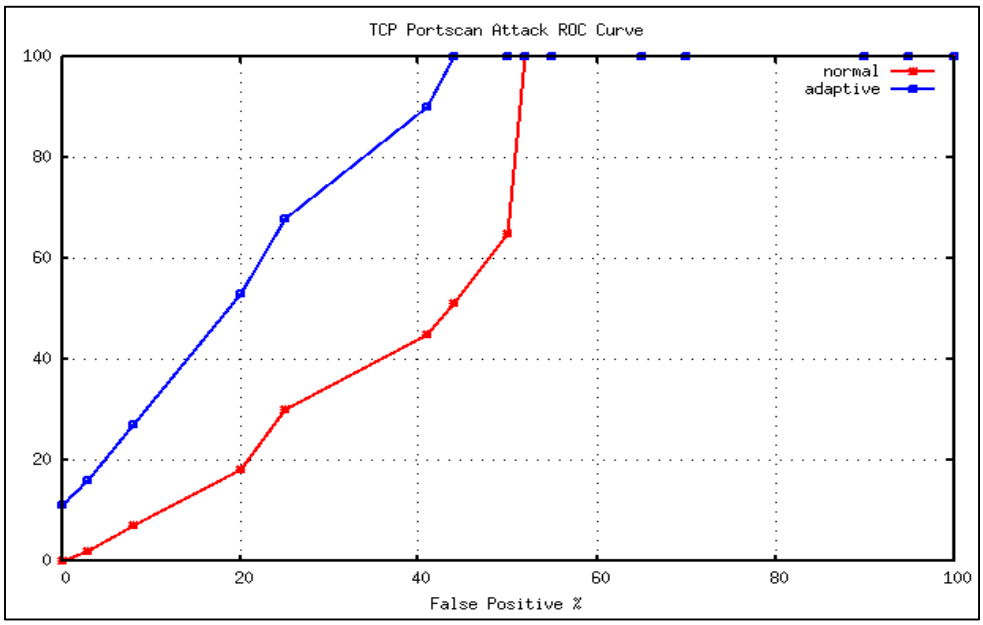

Figure 4.5 ROC Curve for TCP Portscan attack.

Figure 4.5 depicts the ROC curves for TCP Portscan attack with the static and $S[t_i]l$ algorithm. We set the polling rate value of $\alpha = 3, \beta = 2$ and $\partial = 1.1$. From the graph, the K-Mean anomaly classification algorithm achieve *nearly* 100% anomaly detection accuracy for both type of polling rate mechanism. For the static polling rate, the False Positive is approximately almost 52% whereas our proposed adaptive polling rate implementation performed better where the False Positive of is almost 43%. This clearly shows that while the detection rate is almost identical, our proposed method able to reduce the False Positive factor where the legitimate traffic classified as attack which can lead to different action taken from the network administrator.

Furthermore, it also can lead to unnecessary network service disruption for the real customers. In Figure 4.6, the ROC curve illustrate our experiment with DDoS type of network attack. In this experiment, we also injected 200 attack packet per second. When we experiment

the static polling rate, the False Positive is approximately almost 45% and when we tested our algorithm, our proposed adaptive polling rate implementation performed better where the False Positive value significantly drop to almost 34%. The main achievement of our method is that when using adaptive poll rate, the anomaly detection rate is much faster and more accurate than normal poll rate thus enable administrator to alert/mitigate anomalous or suspicious packet efficiently.

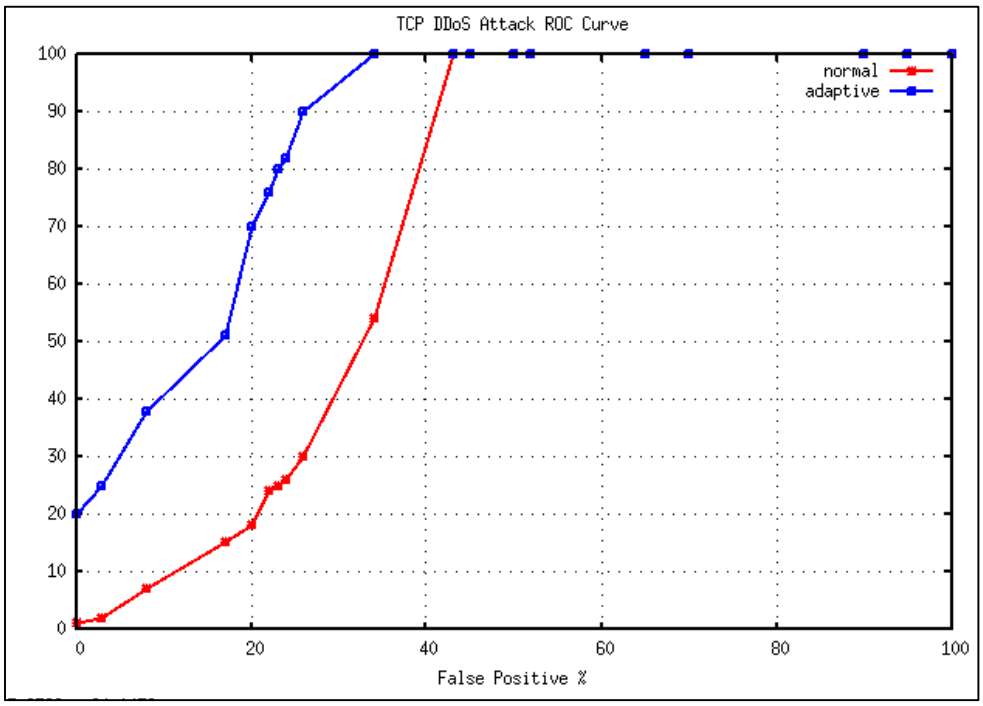

Figure 4.6. ROC Curve for DDoS attack.

With the adaptive poll rate proposed, the algorithm might force the POX controller to perform more computation processing thus could increase the CPU processing time since the controller need to handle anomaly detection and forwarding decision at the same time. Furthermore, it also could increase the communication between the controller and all Openflow switches under control. We analysed the impact of the algorithm proposed on controller CPU processing in the following Section 4.4.5.

4.4.5. CPU Performance

From our first experiment, where we use 110Mbps of traffic rate together with 200 attack packet injected per second, we further test the performance of the controller to find the possible overhead that might introduced. We measure and compare the system resources of the

controller with two types of polling rate. From Table 4.4, we depict the positive factor of the adaptive approach versus the static polling rate approach. We perform two type of test where the first experiment is tested without the attack injected. In this first experiment, we make comparison between the static and adaptive polling rate with the objective is to find the average CPU percentage introduced while performing the polling mechanism. As we can see, the required CPU cycles for our adaptive polling rate with the weighted K-mean classification algorithm is reduced to only 45% when compared to the respective static polling rate approach (57%).

For the next experiment, we inject 200 attack packet that depict the Portscan and DDoS attack while replay the 110Mbps traffic rate. During the attack phase, the static polling rate CPU utilization is increased from 57% to 74% (average of 17% increment of CPU power needed to perform the algorithm. While with our adaptive method, the CPU increase from 45% to 61% (14% different). As shown in Table 4.4, we can achieve a slightly decrease in the CPU cycle usage of the POX controller with our adaptive polling rate methodology. In our method, even though we poll more frequently for flow that classified as attack by the weighted K-Mean algorithm, at the same time we leverage the polling rate for flow that not classified as attack to be more relaxed. By doing this way, we can achieve lower increment in term of CPU usage percentage for the controller.

Table 4.4 CPU performance comparison between the static polling rate versus our proposed adaptive rated methodology.

	110 Mbps traffic (%)	110 Mbps with 200 attack pps (%)
Weighted K-Mean normal poll rate	52	74
Weighted K-Mean adaptive poll rate	48	69

4.5. Summary

We presented our work on developing more accurate intrusion detection mechanism for the network attack in SDN paradigm, ultimately allowing better defence against the network cyber-attack for an organization. We showed that the adaptive query rate anomaly detection is able to detect the abnormal traffic behaviour much more accurate compared with static interval polling rate time. We prove that by using the adaptive method, the False Positive is reduced significantly. Furthermore, by relaxing the polling rate for traffic that not classified by our method as abnormal, we only introduce small increment in CPU percentage compared to static polling rate that does not differentiate any type of flows.

As proven from our simulation, the classification K-Mean algorithm did not achieve 100% detection rate for the injected attack packets. To be exact, the algorithm only able to detect 97.82% from total manipulated attack packets. We strongly believe that this algorithm is not suitable to be used as any DDoS attack defence mechanism for classification. The important achievement in this work is to prove that by giving higher polling frequency to any high probability attack flows from the network, we are able to detect more accurate attack flows as opposed to the previous related work [46,47] that use fix time periodic sampling for all type of flows.

Chapter 5

5. Collaborative Spoofing Detection – SDN based looping authentication for DNS

5.1. Introduction

One of the major cyber-attacks that designated to cripple down the computer or network resources is DDoS attack. Majority of recent large volume of DDoS attack use amplification technique, such as exploiting DNS servers and Network Time Protocol (NTP) [59]. Since almost every Internet services depend on DNS, it is much more damaging than the others are. The introduction of Open Recursive DNS server such as OpenDNS[2] spike security threat higher to this type of attack since it permit any IP addresses to access their service for IP resolution. The hackers can utilize a large number of botnet army with spoofing the victim IP address and make a large number of DNS query attempt to flood the DNS server with request for services.

This will make it hard for the application servers to distinguish between attack and legitimate query, since the application are designed to accept any IP addresses to process. As a result, it would simply process all the DNS query from both of them and send the responses which will limit the resources of the DNS server to process other legitimate request. The other type of attack that manipulate the open recursive DNS server known as DNS Amplification attacks.

This attack aims to amplify the attack traffic to a targeted victim. Since the open recursive server accept any source to send query packet, hacker include victim IP address in the DNS query packet (spoofing) and the query packet size is much smaller than the response packet, so the attack is amplified to the victim with higher impact. Current protection against this type of threat, such as IDS or firewall, is having difficulties of differentiate which response packet is attack or legitimate. Furthermore, by using only the network statistic behavior, it is

[2] Most of upstream resolvers that provide resolution service to any Internet client address are open/public services e.g., Google DNS and OpenDNS.

not enough to separate or blocking such intelligent attacks and most of the preventive action has been left to the victim side [60].

The emergence of Software Defined Network (SDN) that promise the simplicity in managing networks seems to be the future of the current Internet architecture. By separating the control plane that orchestrated by logical network controller platform and data plane as a forwarding drive, SDN, can play an important role as a network protection tool against DDoS attacks. An effective defence against spoofing UDP based DDoS attacks on DNS servers requires source address spoofing detection. Assuming the SDN-managed network is implemented in where the DNS server reside can distinguish between spoofed DNS packets from real queries, it can selectively drop those spoofed packets and authenticate the legitimate DNS query packet with little collateral damage.

In this paper, we present spoof detection mechanism for DNS query packet. In our approach, a server controller that reside in the DNS server domain sends an authentication packet to each host that request for the DNS services. By validating the authentication packet that replied by the requesting host, the server controller is able to determine if a DNS query launch from the source address is indeed a legitimate query or attack packets. We developed a module called *CAuth*, which can be implement without any changes in DNS application servers. *CAuth* can be deployed at any time during DNS server runtime without require dataset training or manual tuning from the administrators.

In this paper, we propose a novel mechanism that autonomously block the “unwanted” DNS query packet that force the DNS servers to amplify the traffic to the victims. Therefore, we did not use any statistical analysis of anomalous flow behaviour for the detection. In Section II, we discuss the existing approaches for defending against amplification attacks and argue why they are not adequate. With the granularity and the flexibility promised by SDN, we state the objective of our proposed method and introduce the idea for the protection against spoofing attack on DNS services in Section III. In Section IV, we empirically evaluate the effectiveness of the approach and present the analysis of the experimental results. Finally, in Section VI, we summarize the paper and list the future work.

5.2. Related Works

DNS, a UDP based network services is crucial and the services need to be offer to the whole public. However, it is much easier for hackers to spoof the source IP address as UDP itself is connectionless and does not require a handshake like TCP does. Therefore, it is worthwhile to design a countermeasure against DDoS flooding that suited the DNS traffic. Through the years, quite a number of DDoS attack detection method against the DNS server has been proposed, but these methods have certain drawbacks such as no strong incentives for the providers to employ since it protects the others network but not protect themselves from the flooding attacks, false positive and false negative, etc.

In [61], the author proposed a method to detect the spoofing DNS query packets. The method requires the DNS server to generate some type of cookies embedded in the DNS response packets. However, it only can authenticate the requests between DNS servers. The main attack tools that is from the general DNS clients, cannot been verified.

RFC 2827, a mechanism that deploy filters at the border of the network to block incoming source IP addresses that not origin from the network itself. Unfortunately, the effectiveness of this method depends on the global deployment across the Internet. This method is "neighbourhood policy" than require all ISP to participate to provide the list of IP addresses that does not belong to their network. No doubt, the spoofing IP problem can be solving by this method but it requires the needed information to be pass between the ISP efficiently. In [60], the author proposed amplification attack detection where they detect the attack using one to one mapping between the DNS query and response.

However, vast amount of database size could increase rapidly when traffic rate is high make the approach is not scalable. In Pushback [62], the mechanism allows network routers to limit the effect of DDoS attack to some destinations. SIFF [63], Stateless Internet Flow Filter permit the packet receiver to inform the router and selectively discard the flows from reaching their network. All the above-discussed related works is a network based solutions and detections method. The main issue in this type of solutions is the complexity of distributed environment, which require quite a number of network resources to be sacrifice.

5.3. Collaborative Authentication Protection

In this section, we elaborate and present our approach that aims to detect the spoofing DNS query packet that gave high impact on the server side resources and performance. The approach exploits the ability of Openflow protocol [31] that provide secure communication channel between the network controller and the router/switches inside their network. We depict the main components of our architecture in Figure 5.1.

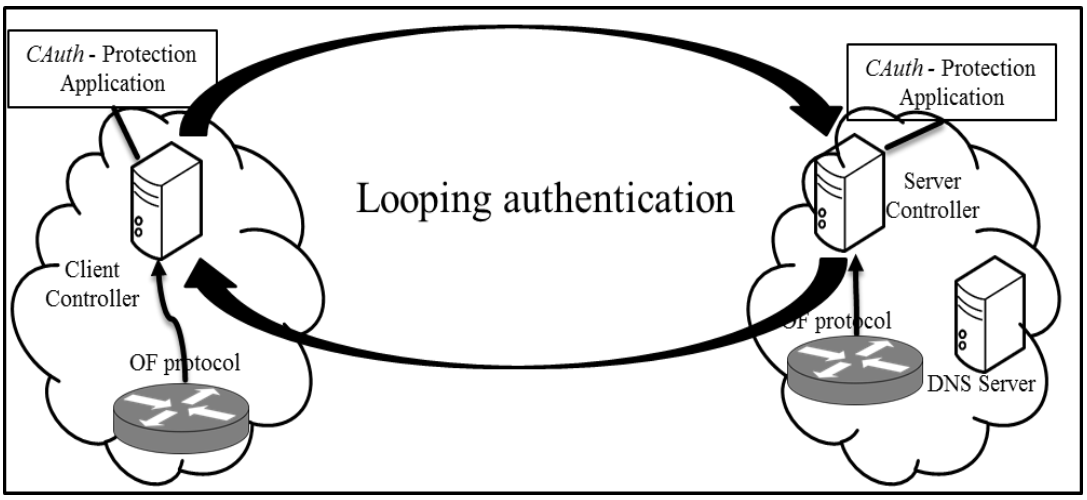

Figure 5.1. Main *CAuth* architecture

5.3.1. Main Component

Based on Figure 5.1, our protection architecture against DNS amplification attack composed of a SDN network controller in both client and server network domain, POX controller [49], the *CAuth* application that include as a component in the network controller and Openflow switches that communicate with the controller via Openflow protocol. Our system require collaboration between client and server side network. We make several assumptions about the architecture components in order our protection application to suit the real environment. The assumptions are as follows:

- The *client controller* holds a list of legitimate internal network IP addresses. The controller uses the addresses to identify the inbound and outbound traffic that flow in their network. The lists play a crucial part in our decision making whether to forward or to accept the incoming DNS response packet. In real practice, IDS/firewall might also have the list of legitimate internal IP addresses to make filtering decision.

- The DNS flooding attack are initiated from botnet by spoofing the IP address of the victim which made the decision to differentiate legitimate or attack traffic become more challenging task. Moreover, the botnet did not show any anomaly behaviour the network operator could easily detect and mitigate.

Based on these assumptions and the system architecture depicted in Figure 5.1, we design a SDN-based DNS spoofing blocking application *CAuth* component.

5.3.2. *CAuth* Table Structure

In our proposed work, *CAuth* record the *inbound* flow information in an active flow table. The main flow table contain all current active *inbound* flows in the network. The structure of the table described as in Table 5.1. In this table, two important field created to record further information about a particular *inbound* flow. In $flow_l$, we record the *5-tuples* (*source port, destination port, source IP, destination IP and the protocol*) of inbound flow. The approach records the *5-tuples* packet header information that sent via *Packet-In* message as the input. The *5-tuples* provided by the switch/router to the controller when the incoming flow has no match to be found in their flow table entries.

In Openflow, when packets received by the datapath and sent to the controller, a control message OFPT_PACKET_IN is used [50]. This control message embeds the packet header together with other important fields to the controller for decision-making.

Table 5.1. Controller Inbound App-Table Structure

***CAuth* App Flow Table**	
Inbound Flow $(src_{ip}, dest_{ip}, src_{port}, dest_{port}, protocol)$	$\boldsymbol{Pi_{hit}}$ **Counter** $(min = 0, max = 2)$
$flow_l, B_{id}$	

Next, Pi_{hit} counter, which is critical spoofing detection indication in our detection method is introduced. This field record how many times the same $flow_l$ information has been receive by both client and server controller. We elaborate more details about this counter in the next section.

5.3.3. Proposed Protection Method Workflow

Initially, when a new query from client (Figure 5.2, step 1) arrives at Openflow switches at the server network, it does not match any flow entry in the switch flow table. As a result, the

switch will buffer the packet and send the copy of the packet to controller via *Packet-In.* When the controller receives a *Packet-In* (Figure 5.2, step 2), assuming there is no information about this particular flow, *CAuth* save the *5-tuples* together with the packet buffer-id, B_{id} in the *CAuth App Table* (Figure 5.2, step 3).

The buffer-id is unique value used to track the location of the buffered packet in the switch. Note that in this work, we did not consider the buffer arrangement problem in the open Vswitches. Since the flow information is not in the server controller list before, *CAuth* update the Pi_{hit} counter to 1. Initially, the value of Pi_{hit} is zero and maximum value is two.

Next, we manipulate the ability of the Openflow protocol to make changes to the packet header information. In (Figure 5.2, step 4), *CAuth* replicate the *Packet-In* flow tuples received with no changes to the original buffered packet. Then, *CAuth* modify the flow information at the replicated packet to send this packet back to the client. As example, the originally initiated DNS query packet source IP address is 1.1.1.1 with source port number 1234 sent to the DNS server with IP address of 2.2.2.2.

CAuth then swapping the IP and UDP information of the *Packet-In*, with the objective to send this packet back to the originator. Then, *CAuth* install the entry in the respected switch with the action to forward this packet back to the source. The modified packet is use by our method as an authentication mechanism to detect the spoofed DNS query packet. It is worth to note that the original query packet still buffered in switch and the flow entry is yet to be installed.

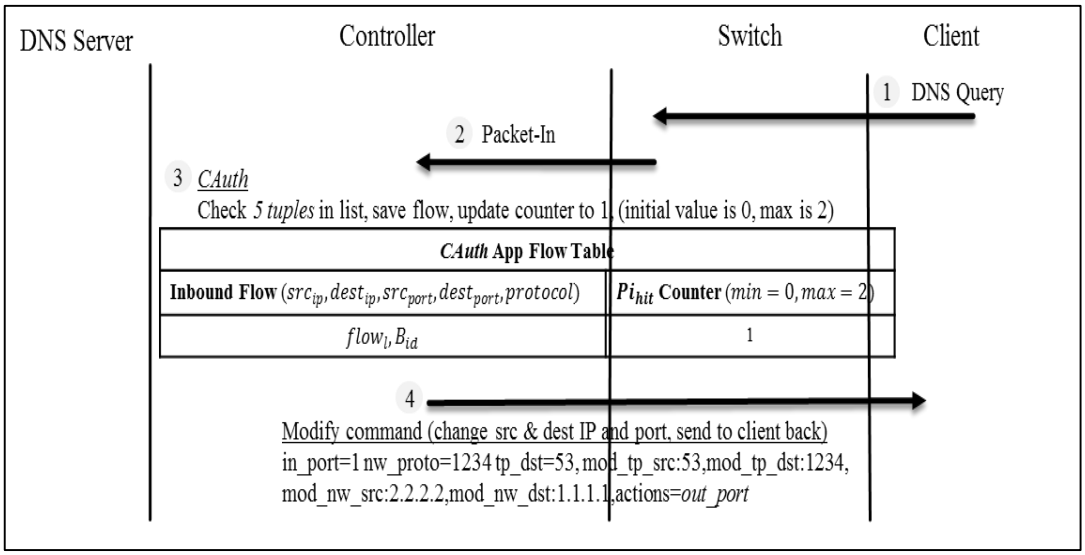

Figure 5.2. Initial *CAuth* defense strategy in server network

The client controller also implements *CAuth App Table* (Figure 5.3) in their database. When the authentication packet arrives at the client network, there will also no match for the packet since it is new flow information to the client network, thus this packet is sent to controller for further decision making via *Packet-In* (Figure 5.3, step 1). When the client controller receives the modified packet from the server network, the authentication process start. First, the client controller will check the destination IP address of the packet header (Figure 5.3, step 2).

The client controller will only perform the authentication process if the destination IP is in their network domain; otherwise, it will forward the authentication packet to the next hop. Here we assume that all client controller has a list of all known IP addresses in their domain. Furthermore, the client controller also checks the source IP address of the authentication packet (Figure 5.3, step 3). If the source IP address is from the server, the client controller further checks in their *CAuth App Table*. Since it is new information for the client controller, the *5-tuples* information copied into the table and the Pi_{hit} counter updated to 1.

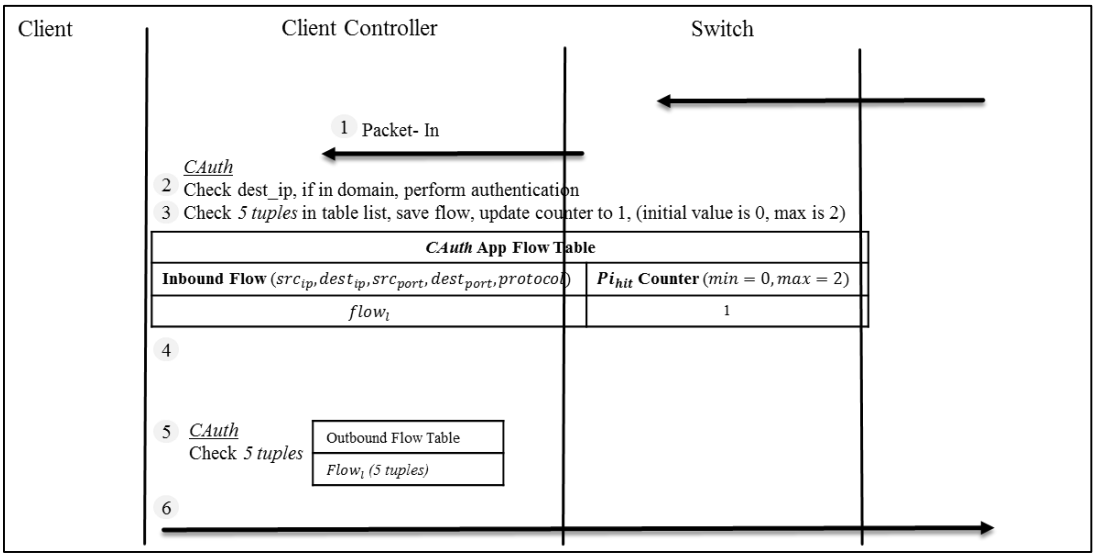

Figure 5.3. Initial *CAuth* defense strategy in client network

Next, the client controller modify the authentication packet received by swapping back the IP and UDP information in the packet header (Figure 5.3, step 4). This time, this authentication packet is aim to be redirect back to the DNS server. The client then checks the modified authentication packet *5-tuples* information with their existing outbound flow table (Figure 5.4, step 5). If the client controller found a match, then the client controller confirms that there are DNS query packet previously sent to server but still waiting for the reply.

Finally, client controller installs the flow entry in the switch with the action to forward the packet to the server (Figure 5.3, step 6). By this means, the authentication packet that originally sent by the server network is redirect back by the client network.

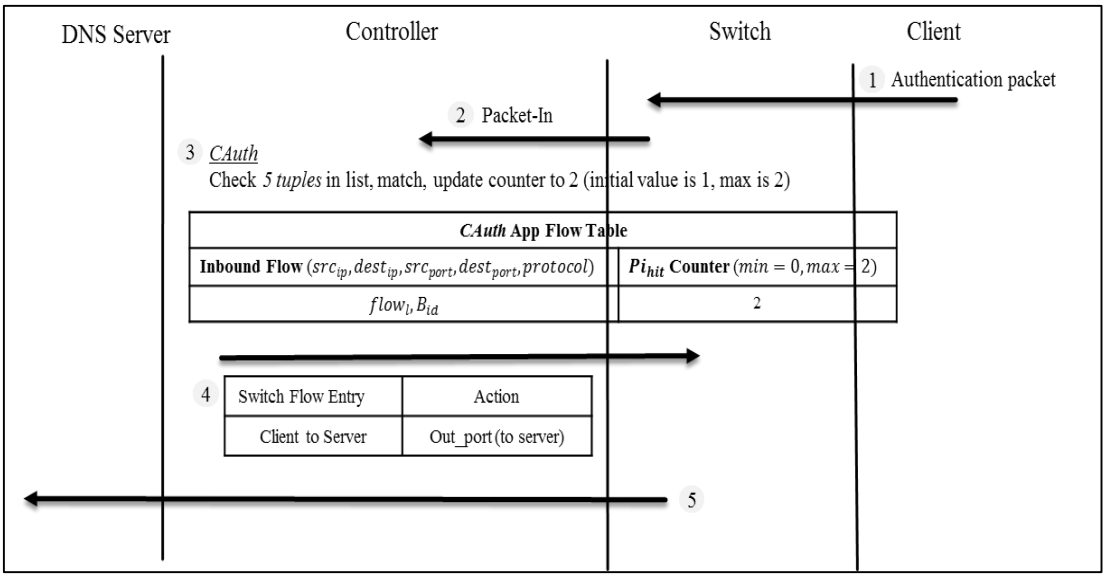

Figure 5.4. Server controller receive *CAuth* authentication packet

When the server controller receive the authentication packet sent from client (Figure 5.4, step 1), this packet information is not match any entry in the switch, since we did not install the flow entry for the original DNS query packet yet. As a result, this packet is being forward to the controller as *Packet-in* again (Figure 5.4, step 2). Then, the server controller refers the incoming authentication packet header with the *CAuth App Table* (Figure 5.4, step 3). This time, server controller found a match for the authentication packet with the previously created flow information. Since there is exactly genuine first packet attempt to the server, the second time controller receive the packet with the same flow *5-tuples* information, it update the Pi_{hit} counter to 2.

After that, the server controller free the previously buffered authentic DNS query packet by installing the flow entry into the switch in order to forward the DNS query to the respected DNS server (Figure 5.4, step 4). Subsequently, *CAuth* authenticate the first attempt query packet to use the DNS server services (Figure 5.4, step 5). In our work, we simply drop the duplicate packet that we used for our authentication mechanism. The authentication packet that created originally by the server controller and redirect back by the client controller are used only for the server to authorize the DNS query packet to reach the DNS server.

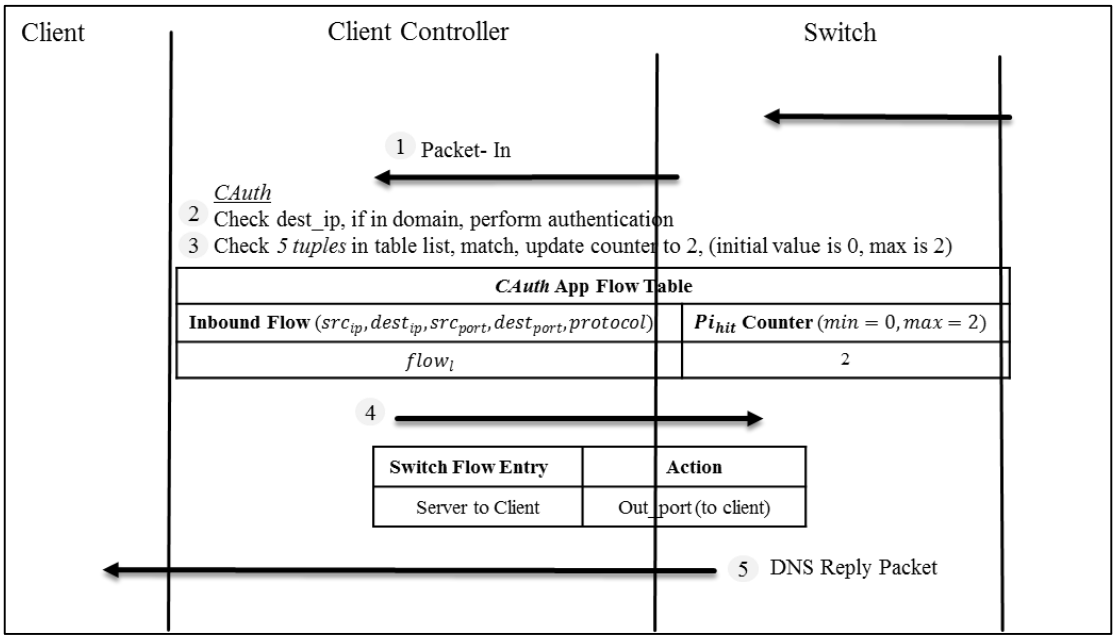

Figure 5.5. Client controller forward the DNS reply packet to client

When the DNS server process the received query, it will find the related query information as usual and reply with the DNS reply packet. Here, we did not modify any application packet for the *CAuth* authentication process. The server controller will forward the DNS reply packet to the client based on the header information. When the client network receives the authentic DNS reply packet from server, again there will be no match found in the switch and the packet sent to controller (Figure 5.5, step 1).

This time the client controller check that *5-tuple* information is already created previously in the *CAuth App Flow Table* (Figure 5.5, step 3). From this, the client controller update the Pi_{hit} counter and install the flow entry in switch to enable the DNS reply packet to reach the client successfully (Figure 5.5, step 4). We provide *CAuth* authentication method pseudo-code for server and client network as in Figure 5.6 and 5.7.

```
Initialization: In_flow, Out_flow, flow_l, Pi_hit = 0/*(Pi_hit min = 0, max = 2), In_flow, Out_flow table record incoming and outgoing flow 5 tuple data structure */
Input : p: PacketIn

For PacketIn
    If dest_ip is SERVER,
    Check flow_l in In_flow
        If empty /*not created yet*/
            flow_l = {src_ip, dest_ip, src_port, dest_port, protocol}/*Extract 5_tuple from PacketIn into In_flow table if the 5 tuple not in table yet*/
            flow_id = buffer_id  /*Extract buffer ID from PacketIn into In_flow table*/
            update Pi_hit=1 for flow_l
            For flow_l, flow_id
                replicate flow_l to (flow_l)_a
                (flow_l)_a = flow_l (src_ip = dest_ip, dest_ip = src_ip, src_port = dest_port, dest_port = src_port) /* modify(flow_l)_a tuple*/
                send(flow_l)_a to outport /*install entry for modified packet to out from network)
        ElseIf not empty /*flow tuples info is already in switch*/
                if (flow_l)_a then check match flow_l /*matching the tuples in incoming flow table */
                if match then update Pi_hit=2 for flow_l
                if Pi_hit=2  then
                    forward flow_l, flow_id to server /*forward buffered flow waiting in switch to server*/
                    drop/ignore (flow_l)_a
                    delete flow_l, flow_id from In_flow table
                End if
        End if
    End if
```

Figure 5.6. *CAuth* server controller defence

Initialization: $In_{flow}, Out_{flow}, flow_l, Pi_{hit} = 0$/*($Pi_{hit}$min = 0, max = 2), In_{flow}, Out_{flow} table record incoming and outgoing flow 5 tuple data structure */
Input : p: *PacketIn*

For *PacketIn*
  If $dest_{ip}$ is in domain,
    If src_{ip} is from SERVER,
      Check $flow_l$ in In_{flow} /*to check whether the same flow info has been created in table or not*/
      If empty /*not created yet*/
        $flow_l = \{\text{src}_{\text{ip}}, \text{dest}_{\text{ip}}, \text{src}_{\text{port}}, \text{dest}_{\text{port}}, \text{protocol}\}$/*Extract 5_tuple from PacketIn into In_{flow} table if the 5 tuple not in table yet*/
        update Pi_{hit}=1 for $flow_l$
        For $flow_l$,
          replicate $flow_l$ to $(flow_l)_a$
          $(flow_l)_a = flow_l$ ($\text{src}_{\text{ip}} = \text{dest}_{\text{ip}}, \text{dest}_{\text{ip}} = \text{src}_{\text{ip}}, \text{src}_{\text{port}} = \text{dest}_{\text{port}}, \text{dest}_{\text{port}} = \text{src}_{\text{port}}$) /* modify$(flow_l)_a$ tuple*/
          send$(flow_l)_a$ to $outport$ /*install entry for modified packet out to SERVER)
      ElseIf not empty /*flow tuples info is already in switch*/
          if $(flow_l)_a$ then check match $flow_l$ /*matching the tuples in incoming flow table */
          if match then update Pi_{hit}=2 for $flow_l$
          if Pi_{hit}=2 then
            forward $flow_l$ to client
            delete $flow_l$ from In_{flow} table
          End if
      End if
  ElseIf $dest_{ip}$ is NOT in domain
    forward $flow_l$ to destination
  End if

Figure 5.7. *CAuth* client controller defence

5.4. Performance Evaluation

5.4.1. Emulation Parameter

To validate the effectiveness of the proposed method, we develop the algorithm to be part of POX controller module. To emulate a large scale of DNS spoofing attack, we use Mininet [67], a network emulator that illustrate SDN environment with the support of virtual hosts, switches and network controller. The Mininet emulator use the real code for both the Openflow and Open vSwitch code and with the great functionality, as it can easily connect to the real networks. For our test, we use Mininet version 2.1.0 with the *Openflow v1.3* supported. To emulate a real DNS traffic, well-known BIND 9.8.1 server configured in separate Linux

stack to serve as the DNS server. Then, we bridge the Mininet SDN network domain to the BIND server virtual machine.

For our experiments, we create two types of DNS enquirers, legitimate clients and spoof attackers. The clients, attackers and the DNS server are design to be on different SDN network domain and have their own POX controller. We simulate a large-scale botnet to launch DDoS attacks on the protected DNS server in SDN environment. We set the legitimate clients to issue DNS query rate, $\bar{x}_{norm}$ every 3 seconds whereas the bots, $\bar{x}_{attack}$ every 1 seconds. The reason why we set query rate higher for the botnet attacks is for the botnet to more active than the legitimate clients.

The DNS query issued with the Poisson exponential inter-arrival time distribution. For the DNS query packet, we employ Scapy [68], a packet manipulation program that able to forge DNS packets easily. In our test, we set the number of legitimate client, n_{users} = 100 and the number of botnet attackers, n_{bots} = 400 that issue the DNS query at their given ferocities. We configure the BIND server to act as recursive server so that whenever the clients send DNS query packet, the BIND server is assumed to be responsive, so the DNS response returned as soon as possible.

5.4.2. Emulation Results

This section describes the evaluation results of the spoofing DNS query attack scenario; we configure all botnet to send the query as soon as the emulation start. This condition is also true for the legitimate clients. We deliberately generate a high throughput of DNS query packet to test the effectiveness of our proposed *CAuth* mechanism that implemented as a module in POX controller at both DNS server and client side. In this experiment, we measure the detection time that the server controller takes to block the spoofed DNS query packet while authenticate the query that issued by the legitimate clients. The botnets and legitimate clients launch DNS query packet that have randomly generated source ports and query name so that every-time the DNS server receive these query, it will act as a recursive. In Figure 5.8, the vertical axis is the number of measured substance, which is the number of blocked bots and the number of authenticated queries from the legitimate clients. We purposely launch the botnets to attack the DNS server at *t*=30 seconds to investigate whether it has effect on the legitimate query packets.

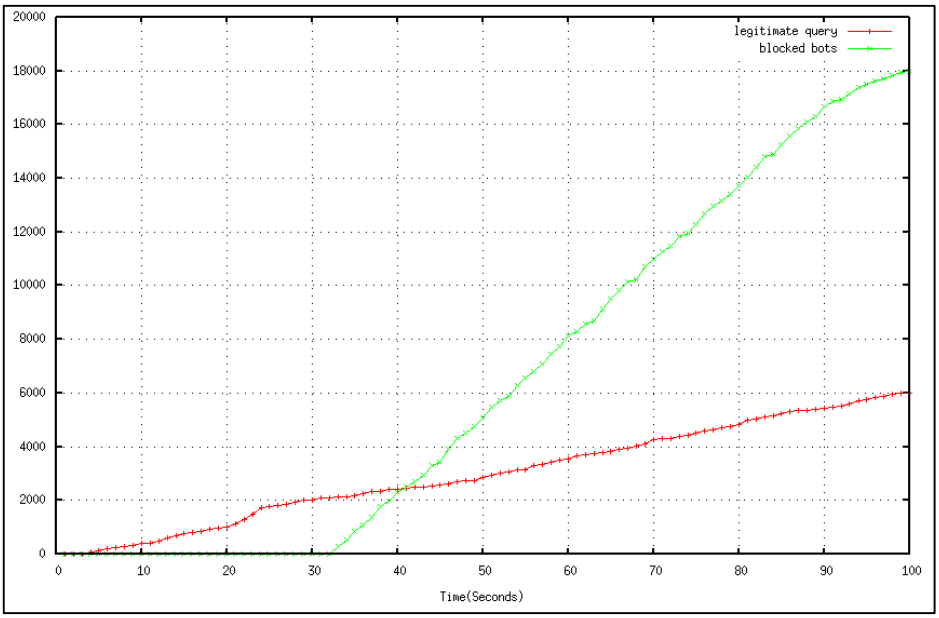

Figure 5.8. Emulation result

As can be seen at approximately t=33.2 seconds, the *CAuth* has start to perform the detection mechanism to block the spoof queries. This indicate that *CAuth* are able to classify and block the spoofed attempt at any time before the server controller forward the queries to the DNS server. Even the attack packet generated at every t=1 second, the server controller depends on the preconfigured timeout to drop the identified spoofed packet. In our case, the server controller decides to drop the incoming DNS queries packet if it did not receive the authentication packet reply from the clients in 2 seconds.

CAuth managed to block all botnets by roughly t=100 seconds. From the Figure 5.8 however, as the legitimate queries inter-arrival rate generated at every 3 seconds, even though *CAuth* can differentiate between the spoofing and legitimate DNS queries, it takes a significant amount of time to forward the legitimate packet to the DNS server. The reason of this behavior is that *CAuth* require additional one round trip time (RTT) in order to authenticate the clients to use the DNS services. The detection of botnet packets has no direct impact on the legitimate queries packets. Starting from near t=4.1 seconds onwards, *CAuth* able to forward all of the legitimate queries to the DNS server.

5.5. Discussion

As we can observe, once the DNS server network controller receive the query packet, it will send an authentication packet out to the clients that initiate the communication. Here, we assume that no spoofing botnet will receive the authentication packet sent by the server controller. Since the well-known objective of spoofing is to saturate the legitimate victim resources, the botnets will never receive any packet from the DNS server. In our works, the *CAuth* decide to drop any $flow_l, B_{id}$ that buffered and unprocessed DNS queries after the server controller did not receive the authentication packet back from the client network in time t=2 seconds. Another important issue to be highlight is the effect of the proposed method to the DNS provider bandwidth. Our method requires one additional RTT for a client to perform DNS query.

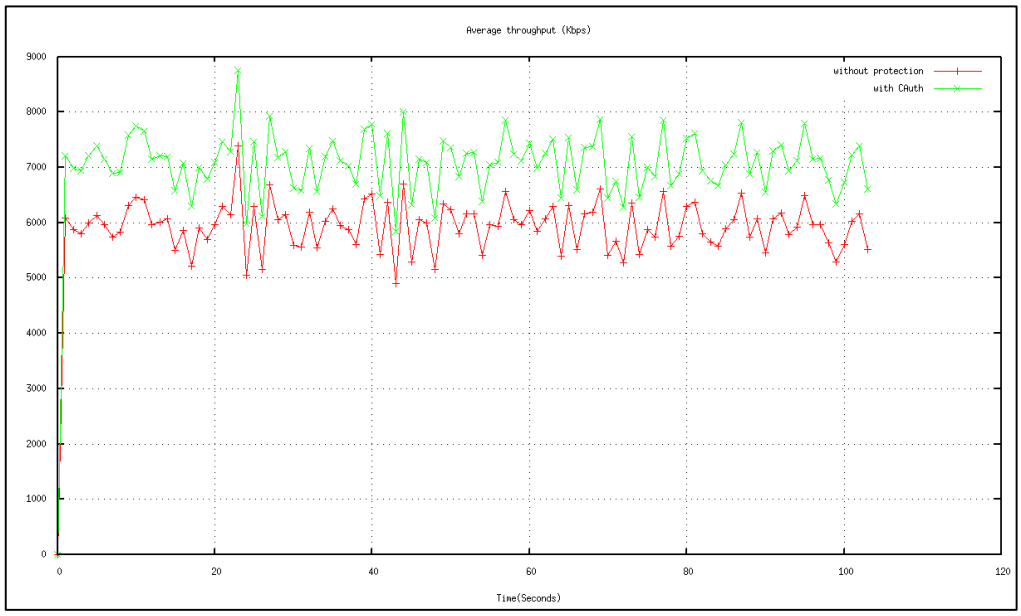

Figure 5.9 DNS provider bandwidth effect

As mentioned before, this additional delay used as an authentication packet in order for the DNS provider to recognize between legitimate and attack packets. From the same emulation, we study the average throughput of the DNS provider bandwidth. The purpose is to find the impact to the provider bandwidth. As in Figure 5.9, as expected, even though our method can differentiate between legitimate and attack packets and able to block the attacks before it

reaches the DNS server, our method consumes the usage of the bandwidth. Our method increases on average 1.2 times higher comparing when there is no protection introduced.

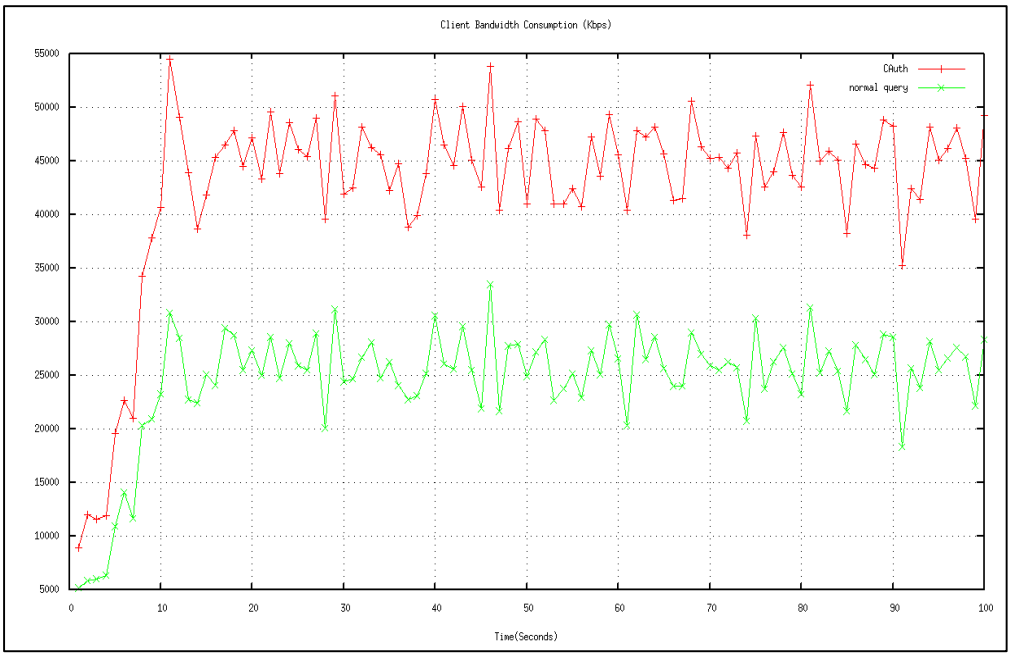

Figure 5.10 Client bandwidth effect

We also perform another emulation to find the effect of our method to the client network. Our method requires the server controller to send an authentication packet back to the source. This critical function introduced in order for the server network to make decision whether to accept or block the incoming packets that attempt to perform DNS queries. Botnet that launch the attack packets will never receive the authentication packet sent by the server network, but the legitimate client will. In this test, 100 hosts are configured to launch DNS query packets towards the DNS server where the inter-arrival time of each packet is every 1 seconds. From Figure 5.10, the client bandwidth consumption on average increased 1.7 times higher than a network without protection. We believe that this effect does not much impact the client side since we can guarantee that the client is secured from any spoofed packet attacks.

5.6. Summary

In this paper, we proposed an efficient spoofing detection mechanism to detect spoofed DNS query packets against DNS servers thus ensuring the DNS operator to be able to protect their server resources. The key thwart of our spoofing detection method is to block all queries from clients that did not reply back the authentication packet sent previously by the server controller. Our method authenticates the same queries that enter the server network domain for the second time. From the experiments, we can conclude that our method increases the bandwidth consumption on both the client and the server network. It is worth to note that, the UDP attack is fast since the size of the attack packets is small but it was not design to attack bandwidth but to consume the victim resources.

This scheme is well suited to a SDN-administered network since the client and server controllers need to collaborate for the process of creating the authentication packets. Furthermore, our scheme does not require high computation algorithm to detect the spoofed packet such as public key cryptography that usually used for authentication purpose. Moreover, we did not introduce any new protocols and all interaction between the client and server networks use standard Openflow protocol, make it as a lightweight spoofing detection method.

Chapter 6

6. Conclusion

Software Defined Networking (SDN) emerges as a future Internet network architecture. The core feature of SDN is to decoupled the controlling function of network from the forwarding function and the network is managed from a centralized network controller. Through dynamic, cost-effective and easily adaptable, making it supreme for the dynamic nature and high bandwidth of today's network applications. Thus, this thesis focused on two significant contributions of SDN: support IP fast rerouting to be able to restore the network below 50 milliseconds when there is a single link failure in the network and network security area which focusing on higher accuracy of anomaly detection strategies and low cost UDP based spoofing DDoS attack mechanism.

This thesis firstly illustrated the backgrounds, motivation and our goals. Secondly, related works according to fast rerouting and network security of SDN are reviewed. Then, architecture of SDN and summarized the features of Openflow is presented. Lastly, the corresponding methods were designed to solve these problems.

Any disruption to a link in a network has the potential to affect the important network traffic communications and applications. The ability to recover fast has always been a central objective in the current Internet. Routing protocol like OSPF are designed to update their routing information based on the topology changes after failure. This convergence process is a time consuming process since every router need to individually calculates new valid routing path and usually lead to routing instability. Furthermore, to achieve a carrier grade network, network service restoration is proposed to be sub-50 milliseconds which is not the case with the existing IP services fast rerouting. This is further proven with undeveloped IP fast reroute within current Internet infrastructure.

With the nature of SDN, where all of the control function managed centrally and the network node only do the forwarding task, fast rerouting below sub-*50 milliseconds* is not achieved. In Chapter 3, we present a fast rerouting mechanism for IP infrastructure in case of a

single link failure utilizing the Openflow protocol. By providing the pre-configured backup path in every switches for every flows, we ensure the affected path to reroute within *50 milliseconds* with the help of Openflow switch. In our work, we introduced the concept of local controller. Local controller is responsible for the deletion of affected flow in case of link failure and by doing this, the affected flow will reroute to the pre-configure path autonomously. In every Openflow switches, main path and backup path are differentiated by priority. Flow entry with higher priority number is the main path and lower priority number is the backup path.

However, when the link fail, the remaining flow that use the pre-configured path to reroute might use non-optimal backup path. Thus it will increase a significant delay to reach their destination. To avoid this problem, we combine the ability of the local controller with the Openflow switches to report of any event to the central controller. When the link fail, while the local controller performs the autonomous rerouting locally, the switch sends port status notification to the central controller and the central controller responsible to perform the new path calculation.

When the central controller receives the notification from the affected switch, it will recalculate the whole topology without the affected flow port. In our work, we ensure that every time the central controller receives the event notification from switch, it will recalculate and elect the new main and backup path for the affected path respectively. From the simulation, it shows that our hybrid mechanism able to lower the hop count for every affected flow to reach their target compared to packet that traverse to their destination using the pre-defined backup path. Thus, we ensure the affected flows to autonomously reroute using much more optimal path and also converge effectively below the carrier grade network restoration time.

Chapter 4 investigated intrusion detection mechanism for the network attacks to allow better defence against cyber-attack for any organization. In anomaly detection, sampling technique is usually used for the traffic analysis on finding any anomalous flow. While the use of sampling is capable to lessen the scalability problem of traffic monitoring, the insufficiency of sampled flow statistic may have led to inaccurate detection rate of anomaly. With increasing amount of network traffic, sampling techniques have become widely used, allowing monitoring and analysis of high-speed network. Despite of all benefits, sampling methods negatively influence the accuracy of anomaly detection techniques and other subsequent processing.

Furthermore, it is paramount to balance the trade-off between accuracy of anomaly detection and overhead introduced from the flow traffic measurements (e.g.-scalability). Sampling decision should have some intelligent ability to address some of the requirement such as to reach low false alarm and low computation convolution objective. If too frequent sampling was being used in detection, the load or the processing overhead is increased while too infrequent sampling will lead to loss of any important flow information to be analysed.

We presented our work on developing more accurate intrusion detection mechanism for the network attack in SDN paradigm, ultimately allowing better defence against the network cyber-attack for an organization. For the analysis, we choose 5 predefined feature set to be used in our experiment. *Flow size*, F_l (packet count) at t_i interval, *Flow byte*, B_l *(byte sizes*) at t_i interval, *number of different flows to the same Destination IP*, $n(D_{IP})\ at$ t_i interval, *number of flows to different Destination Ports*, $n(D_{Port}) at$ t_i interval *and number of different S-D pair*, $n(SD_{pair})\ at$ t_i interval is used. We utilized weighted K-Mean algorithm to differentiate the anomaly and the normal flow traffic. Moreover, to balance the tradeoff between accuracy and scalability, we tune the polling frequency for the network traffic to give more priority for the frequently used attack source port. For any source port numbers that does not identified as attack source port, we further lower down the polling frequency to stabilize the processing load for other flows.

From the simulation results, the proposed method prove that the adaptive query rate anomaly detection is able to detect the abnormal traffic behaviour much more accurate compared with static interval polling rate time. We show that by using the adaptive method, the False Positive is reduced significantly. The method proposed able to reduce 9% on false positive for the injected port scan attacks and further 11% lower false positive for injected DDoS attacks. Furthermore, by relaxing the polling rate for traffic that not classified by our method as abnormal, we only introduce small increment in CPU percentage compared to static polling rate that does not differentiate any type of flows. The proposed method improves the CPU utilization of the controller to 5% lower than compared mechanism.

As proven from our simulation, the classification K-Mean algorithm did not achieve 100% detection rate for the injected attack packets. To be exact, the algorithm only able to detect 97.82% from total manipulated attack packets. We strongly believe that this algorithm is not suitable to be used as any DDoS attack defence mechanism for classification. The important

achievement in this work is to prove that by giving higher polling frequency to any high probability attack flows from the network, we are able to detect more accurate attack flows as opposed to the previous related work [46,47] that use fix time periodic sampling for all type of flows.

In Chapter 5, a spoofing detection method was introduced. In UDP attack, specifically the DNS services, spoofing IP address of the victim is usually used to launch an enormous amount of attack packets that targeting the victim with the objective to make the victim services unavailable to the legitimate users. With the introduction of open resolvers such as OpenDNS and Google DNS, this services enable any IP addresses to perform domain lookup. Thus, the open resolver has problem on defining which is the legitimate and which one is attack DNS query packets. Since the DNS attack seldom utilize a large number of botnets army that spoofed the victim IP address, the DNS server network simply respond to those attack packets and redirect the legitimate packet back to the victim itself, thus the victim suffers from the unwanted DNS response packets that impact their application itself.

An effective defense against spoofing UDP based DDoS attacks on DNS servers requires source address spoofing detection. Assuming the SDN-managed network is implemented in where the DNS server reside can distinguish between spoofed DNS packets from real queries, it can selectively drop the spoofed packets and authenticate the legitimate DNS query packet with little collateral damage. In this paper, we present a novel spoofing detection mechanism, *CAuth*. In this mechanism, server and client side require to have their own network controller to collaborate between themselves to defeat the spoofing packets to the DNS server and at the same time, it can guarantee the client not to suffer from the effect of the attacks packet that targeted to them previously.

In general, once the DNS server side controller receive a DNS query packet, the controller decides to replicate the DNS query packet and modify the UDP and IP header of the replicated packet in order to destined this packet back to client. This replicated packet is used as an authentication packet in order for the server controller to differentiate which one is legitimate and which one is attacks DNS query packets. Legitimate client will receive this

authentication packet and once they receive it, the client controller decides to redirect this authentication packet back to DNS server network. It modifies the UDP and IP header information of this authentication packet and install entry for this packet to make sure this packet redirect back to the server network.

Once this authentication packet that redirected back by the client reach the server controller, then the server controller identified that the previous original packet that still buffered in the incoming switch is actually a legitimate query, thus it installs the flow entry for the packet to make sure the query reaches the DNS application server. Any spoofing packets will not receive the authentication packet that was sent by the server controller and the server controller decide to drop the attempted attack before it reach the DNS application server itself. In our work, if the server controller did not receive the authentication packet back from client network in 2 seconds, the server controller decides to drop the buffered DNS query packet.

Once the DNS response reach the client network, this time, the controller found out that this is the second time the client network receives the same flow information, thus, the client controller identified this is the actual DNS response that was sent by the server network. From this, client controller installs the entry for the incoming DNS response packet to route the packet back to the client itself.

From the emulation, *CAuth* able to block all of the manually launch DNS query attack packets without any impact on the legitimate query packets. Our mechanism able to identify efficiently between attack and legitimated DNS query packets, thus guarantee the client to be not affected. The detection time for spoofing packets is almost real-time since the controller decide to block the spoofing packet if the server controller did not receive the redirect authentication packet sent by the client controller within 2 seconds. From the simulation, the proposed method able to block spoof DNS query packet less than 3 seconds.

From this method, the detection is done autonomously and application friendly. However, the attack packets still will travel from their source to the destination which is the

DNS server network, thus this method is not bandwidth friendly. From the simulation, since the proposed authentication mechanism require additional one round trip time (RTT) to be able to differentiate between the legitimate and spoofing attack packets, the legitimate client takes around 1.1 seconds to get the DNS response packet from the DNS server.

Moreover, the client bandwidth is consumed 1.7 times higher, e.g.- without proposed authentication, throughput is 26000 Kbps, with proposed authentication method, throughput increased to 41500 Kbps. However, note that if there is no any protection used to defend the DNS server from attacks, the attacks packet will reach the DNS server and consume the DNS server resources itself, thus the DNS services will not be available to handle the legitimate DNS request. The proposed method is not the case. Even though it increases the network bandwidth overhead, the method proposed clearly able to identify the attack packets and drop the packets before it reach the DNS server resources. Thus the DNS server resources is well protected.

References

[1] C. Hedrick, "RFC 1058: The Routing Information Protocol (RIP)," *Internet Engineering Task Force (IETF) Request For Comments,* 1988.

[2] G. Malkin, "RFC 2453: Rip version 2," *Internet Engineering Task Force (IETF) Request For Comments,* 1998.

[3] J. Moy, "RFC 2328: OSPF Version 2," *Internet Engineering Task Force (IETF) Request For Comments,* 1998.

[4] D. Oran, "RFC 1142: OSI IS-IS Intra-domain Routing Protocol," *Internet Engineering Task Force (IETF) Request For Comments,* 1990.

[5] Y. Rekhter, T. Li and S. Hares, "RFC 4271: A Border Gateway Protocol 4 (BGP-4)," *Internet Engineering Task Force (IETF) Request For Comments,* 2006.

[6] Open Networking Foundation, "Software-Defined Networking: The New Norm for Networks," Open Networking Foundation, 2012.

[7] J Boyle, "RFC 4105 - Requirements for Inter-Area MPLS Traffic Engineering", *Internet Engineering Task Force (IETF) Request For Comments,* 2005.

[8] P. Psenak, S. Mirtorabi, A. Roy, L. Nguyen, and P. Pillay-Esnault, "MT-OSPF: Multi-topology (MT) routing in OSPF," IETF, RFC4915, June 2007.

[9] E Boschi, "RFC 5153 - IP Flow Information Export (IPFIX) Implementation Guidelines", *Internet Engineering Task Force (IETF) Request For Comments,* 2008.

[10] B Claise, "RFC 5476 - Packet Sampling (PSAMP) Protocol Specifications", *Internet Engineering Task Force (IETF) Request For Comments,* 2009.

[11] BARFORD P., KLINE J., PLONKA D., ET AL.: 'A signal analysis of network traffic anomalies'. Proc. 2nd ACM SIGCOMM Workshop on Internet Measurement, Marseille, France, November 2002, pp. 71–82

[12] YE N., EMRAN S., CHEN Q., ET AL.: 'Multivariate statistical analysis of audit trails for host-based intrusion detection', IEEE Trans. Comput, 2002, 51, (7), pp. 810–820

[13] LEE W., XIANG D.: 'Information-theoretic measures for anomaly detection'. Proc. IEEE Symp. Security and Privacy, Oakland, CA, USA, May 2001, pp. 130–143

[14] CERT, Denial of Service Attacks, June 4, 2001, [online] http://www.cert.org/tech tips/denial of service.html

[15] J. Mirkovic and P. Reiher, A taxonomy of DDoS attack and DDoS defense mechanisms, ACM SIGCOMM Computer Communications Review, vol. 34, no. 2, pp. 39-53, April 2004.

[16] S. Ranjan, R. Swaminathan, M. Uysal, and E. Knightly, DDoS-Resilient Scheduling to Counter Application Layer Attacks under Imperfect Detection, IEEE INFOCOM'06, 2006.

[17] P. Ferguson, and D. Senie, Network Ingress Filtering: Defeating Denial of Service Attacks that employ IP source address spoofing, Internet RFC 2827, 2000

[18] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker, Controlling high bandwidth aggregates in the network, presented at Computer Communication Review, pp.62-73, 2002.

[19] R. Chen, J. M. Park, and R. Marchany, RIM: Router interface marking for IP traceback, in IEEE Global Telecommunications Conference (GLOBECOM'06), 2006.

[20] B. Al–Duwairi, and G. Manimaran, Novel Hybrid Schemes Employing Packet Marking and Logging for IP Traceback, IEEE Trans. Parallel Distrib. Syst., vol. 17, no. 5, pp. 403-418, May 2006.

[21] S. Savage, D.Wetherall, A. Karlin, and T. Anderson, Practical Network Support for IP Traceback, Technical report, Department of Computer Science and Engineering, University of Washington, 2000.

[22] S. Kent, and R. Atkinson, Security Architecture for the Internet Protocol, IETF, RFC 2401, November 1998.

[23] Richard Wang, Dana Butnariu, and Jennifer Rexford. OpenFlow-based Server Load Balancing Gone Wild. In *Proceedings of the 11th USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, Hot-ICE'11, pages 12–12, Berkeley, CA, USA, 2011. USENIX Association.

[24] Ankur Kumar Nayak, Alex Reimers, Nick Feamster, and Russ Clark. Resonance: Dynamic Access Control for Enterprise Networks. In *Proceedings of the 1st ACM Workshop on Research on Enterprise Networking*, WREN '09, pages 11–18, New York, NY, USA, 2009. ACM

[25] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. Hedera: Dynamic Flow Scheduling for Data Center Networks. In

Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation, NSDI'10, pages 19–19, Berkeley, CA, USA, 2010. USENIX Association.

[26] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "Enabling fast failure recovery in OpenFlow networks," in 8th Int. Workshop on the Design of Reliable Communication Networks (DRCN), Oct. 2011, pp. 164–171.

[27] Sgambelluri, A.; Giorgetti, A.; Cugini, F.; Paolucci, F.; Castoldi, P., "OpenFlow-based segment protection in Ethernet networks," Optical Communications and Networking, IEEE/OSA Journal of, vol.5, no.9, pp.1066,1075, Sept. 2013

[28] "Cisco Global Cloud Index, Forecast and Methodology, 2012-2017"Web site, http://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/Cloud_Index_White_Paper.html

[29] M. Shand, "IP Fast Reroute Framework," IETF Internet Draft (work in progress), June 2005, draft-ietf-rtgwg-ipfrr-framework-03.txt

[30] MCKEOWN, N., ANDERSON, T., BALAKRISHNAN, H., PARULKAR, G., PETERSON, L., REXFORD, J., SHENKER, S., AND TURNER, J. OpenFlow: Enabling Innovation in Campus Networks. SIGCOMM CCR 38, 2 (2008), 69–74.

[31] Hundessa, L.; Pascual, J.D., "Fast rerouting mechanism for a protected label switched path," *Computer Communications and Networks, 2001. Proceedings. Tenth International Conference on* IEEE Computer Communications and Networks (2001), vol., no., pp.527,530, 2001

[32] P. Chimento and J. Ishac. Defining Network Capacity. RFC 5136 (Informational), February 2008

[33] Openflow specification version 1.1.0, http://www.openflow.org/documents/openflow-spec-v1.1.0.pdf

[34] A. Kvalbein, A.F. Hansen, T. Cicic, S. Gjessing, and O. Lysne, "Fast IP network recovery using multiple routing configurations," Proc. INFOCOM, pp.23–29, April 2006.

[35] A. Varga and R. Hornig. An overview of the OMNeT++ simulation environment. In International Conference on Simulation Tools and Techniques for Communications, Networks and Systems, Mar 2008.

[36] A. Varga. INET Framework for the OMNeT++ Discrete Event Simulator. http://github.com/inet-framework/inet, 2012.

[37] A. Medina, A. Lakhina, I. Matta, and J. Byers, “BRITE: An ap- proach to universal topology generation,” Proc. IEEE MASCOTS, pp.346–353, Aug. 2001.

[38] S. Jain, A. Kumar, S.Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J.Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Holzle, S. Stuart, and A. Vahdat, “B4: Experience with a Globally-deployed Software DefinedWAN,” SIGCOMM Comput. Commun. Rev., vol. 43, no. 4, pp. 3–14, Aug. 2013

[39] P. Barford and D. Plonka, “Characteristics of Network Traffic Flow Anomalies,” Proc. 1st ACM SIGCOMM Internet Measurement Wksp, San Francis- co, CA, Nov. 2001, pp. 69–74.

[40] MAI J., SRIDHARAN A., CHUAH C.N., ET AL.: ‘Impact of packet sampling on portscan detection’, IEEE J. Sel. Areas Commun., 2006, 24, (12), pp. 2285–2298

[41] MAI J., SRIDHARAN A., CHUAH C.N., ET AL.: ‘Is sampled data sufficient for anomaly detection?’. Internet Measurement Conf., Rio de Janeiro, Brazil, October 2006, pp. 165–176

[42] DUFFIELD N.G., LUND C.: ‘Predicting resource usage and estimation accuracy in an IP flow measurement collection infrastructure’. ACM SIGCOMM Internet Measurement Conf., Miami, FL, USA, October 2003, pp. 179–191

[43] ESTAN C., VARGHESE G.: ‘New directions in traffic measurement and accounting’. Proc. SIGCOMM’02, Pittsburgh, PN, USA, August 2002, pp. 323–336

[44] ANDROULIDAKIS G., CHATZIGIANNAKIS V., PAPAVASSILIOU S., ET AL.: ‘Understanding and evaluating the impact of sampling on anomaly detection techniques’. IEEE Military Communications Conf.,Washington, DC, USA, October 2006

[45] Rodrigo Braga, Edjard Mota, Alexandre Passito, Lightweight DDoS flooding attack detection using NOX/OpenFlow, in: LCN ‘10 Proceedings of the 2010 IEEE 35th Conference on Local, Computer, 2010, pp. 408–415.

[46] Syed Akbar Mehdi, Junaid Khalid, Syed Ali Khayam, Revisiting traffic anomaly detection using software defined networking, in: RAID’11 Proceedings of the 14th International Conference on Recent Advances in Intrusion Detection, 2011, pp. 161–180.

[47] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, and V. Maglaris, “Combining openflow and sflow for an effective and scalable anomaly detection and mitigation mechanism on sdn environments,” Computer Networks, vol. 62, no. 0, pp. 122 – 136, 2014.

[48] POX.'An Openflow Controller', Online Referencing, http://www.noxrepo.org/pox/about-pox/ (2008, accessed May 2015).

[49] Y. Gu, A. McCallum, and D. Towsley, "Detecting anomalies in network traffic using maximum entropy estimation," in Proc. Internet Measurement Conference, 2005

[50] RAMADAS, M., OSTERMANN, S., AND TJADEN, B. C., "Detecting anomalous network traffic with self-organizing maps." In Proceedings of the Conference on Recent Advances in Intrusion Detection. 2003, 36–54

[51] Ed. Belson David, "The State of the Internet," Volume 6, Number 2, Akamai Internet Quarterly Report, Online Referencing, http://www.akamai.com/dl/documents/akamai_soti_q213.pdf (2013, accessed March 2015).

[52] Open Networking Foundation, "OpenFlow switch specification, version 1.3.", Online Referencing, https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf (2012, accessed April 2015).

[53] Mininet, "An Instant Virtual Network on your Laptop", Online Referencing, http://mininet.org (2012, accessed April 2015).

[54] Ben Plaff et al., Extending networking into the virtualization layer, in: 8th ACM Workshop on Hot Topics in Networks (HotNets-VIII), New York, City, 2009.

[55] CAIDA, "The CAIDA UCSD Anonymized Internet traces 2013.", Online Referencing, http://www.caida.org/data/passive/passive_2013_dataset.xml (2013, accessed August 2015).

[56] Tcpreplay, Online Referencing, http://tcpreplay.synfin.net(accessed June 2015).

[57] SCAPY, Online Referencing, http://hg.secdev.org/scapy (accessed June 2015).

[58] Arbor Networks. Q1 2015 Infrastructure Security Report. [Online]. Available: http://preview.tinyurl.com/kvacqcv

[59] G. Kambourakis, T. Moschos, D. Geneiatakis, and S. Gritzalis, "Detecting DNS Amplification Attacks," in Workshop on Critical Information Infrastructures Security (CRITIS), vol. 5141. Springer, 2008, pp. 185–196.

[60] F. Guo, J. Chen, and T. Chiueh, "Spoof detection for preventing DoS attacks against DNS servers," in IEEE ICDCS, 2006, pp. 37–37.

[61] J. Ioannidis and S. M. Bellovin, "Implementing pushback: Router-based defense against ddos attacks," in Proc. of the Symposium on Network and Distributed Systems Security (NDSS 2002), 2002.

[62] A. Yaar, A. Perrig, and D. Song, "SIFF: A Stateless Internet Flow Filter to Mitigate DDoS Flooding Attacks," in Proc. of IEEE Symposium on Security and Privacy, 2004, 2004.

[63] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: Enabling innovation in campus networks. ACM SIGCOMM Computer Communications Review, 38(2):69–74, 2008

[64] POX. [Online]. Available: http://www.noxrepo.org/pox/about-pox/

[65] OpenFlow Switch Specification, Version 1.3.0 (Wire Protocol 0x04). [Online]. Available: https://www.opennetworking.org/images/stories/downloads/specification/openflow-spec-v1.3.0.pdf

[66] Mininet. (2013, Mar). An Instant Virtual Network on your Laptop (or other PC). [Online]. Available: http://mininet.org

[67] SCAPY. <http://hg.secdev.org/scapy>.

[68] A. Kvalbein, A.F. Hansen, T. Cicic, S. Gjessing, and O. Lysne, "Fast IP network recovery using multiple routing configurations," Proc. INFOCOM, pp.23–29, April 2006.

[69] V. Sharma and F. Hellstrand, "Framework for multi-protocol label switching (MPLS)-based recovery," IETF, RFC 3469, Feb. 2003.

[70] Atkins, D., Austein, R., "Threat Analysis of the Domain Name System (DNS)", RFC 3833, Auguest 2004.

[71] A. Herzberg and H. Shulman. DNS Authentication as a Service: Preventing Amplification Attacks. In Computer Security Applications Conference, 2014. ACSAC'14. Annual, December 2014

[72] Jianning Mai, Chen-Nee Chuah, Ashwin Seidharan, et al. Is sampled data sufficient for anomaly detection//Proceedings of the 6th ACM SIGCOMM on Internet Measurement. New York: ACM Press, 2006, pp. 165-176.

[73] Daniela Brauckhoff, Bernhard Tellenbach, Arno Wagner, et al. Impact of Packet Sampling on Anomaly Detection Metrics Proceedings of the 6th ACM SIGCOMM on Internet Measurement. New York: ACM Press, 2006, pp. 159-164.

[74] S. Ramamurthy and B. Mukherjee, “Survivable WDM Mesh Networks, Part I - Protection,” in Proc. of IEEE INFOCOM, vol. 2, pp. 744-751, Mar. 1999.

[75] V. Sharma and F. Hellstrand, “Framework for Multi-protocol Label Switching (MPLS)-based Recovery,” in IETF, RFC 3469, Feb. 2003.

[76] J. Moy, “OSPF Version 2,” IETF RFC 2328, Apr. 1998.

[77] C. Alaettinoglu, V. Jacobson, and H. Yu, “Towards Millisecond IGP Conver- gence,” in IETF Internet draft 2000.

[78] D. Frigioni, A. Marchetti-Spaccamela, and U. Nanni, “Incremental Algo- rithms for Single-Source Shortest Path Trees,” in Proc. of Foundations of Software Tech. and Theoretical Comp. Sci., Dec. 1994, pp. 113-24.

[79] P. Narvaez, “Routing Reconfiguration in IP Networks,” Ph.D. dissertation, MIT, June 2000.

[80] A. Kvalbein, T. Cicic and S. Gjessing, “Post-Failure Routing Performance with Multiple Routing Configurations,” in Proc. of INFOCOM, May 2007.

[81] R. Takahashi, S. Tembo, K. Yukimatsu, S. Kamamura, T. Miyamura, and K. Shiomoto, “Dispersing Hotspot Traffic in Backup Topology for IP Fast Reroute,” in Proc. of ICC, Jun. 2011.

[82] J. Wang, and S. Nelakuditi, “IP Fast Reroute with Failure Inferencing,” in Proc. of INM’07, at ACM SIGCOMM, Aug. 2007.

[83] Kang Xi, H. J. C. “ESCAP: Efficient SCan for Alternate Paths to Achieve IP Fast Rerouting” in Proc. of IEEE Globecom 2007.

[84] A. Tam, K. Xi, and H. J. Chao, “A Fast Reroute Scheme for IP Multicast,” in Proc. of IEEE Globecom, Dec. 2009.

[85] Kang Xi, C. H. J., C. Guo, “Recovery from Shared Risk Link Group Failures Using IP Fast Reroute,” in Proc. of IEEE ICCCN Aug. 2010.

[86] Tan, P.-N., Steinbach, M., and Kumar, V. 2005. Introduction to Data Mining. Addison-Wesley.

[87] S.T. Zargar, J. Joshi, and D. Tipper. A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks. IEEE Communications Surveys & Tutorials, 15(4):2046–2069, January 2013.

[88] S. Kent, and R. Atkinson, Security Architecture for the Internet Protocol, IETF, RFC 2401, November 1998.

[89] S. Kent, and R. Atkinson, IP Authentication Header, IETF, RFC 2402, November 1998

[90] H. Wang, C. Jin, and K. G. Shin, Defense Against Spoofed IP Traffic Using Hop-Count Filtering, IEEE/ACM Trans. Netw., vol. 15, no. 1, pp.40-53, February 2007.

[91] M. Abliz, Internet Denial of Service Attacks and Defense Mechanisms, University of Pittsburgh, Department of Computer Science, Technical Report. TR-11-178, March 2011

Published Papers

(1) N. M. SAHRI and Koji OKAMURA, “Fast failover mechanism for software defined networking: Openflow based,” in Proceedings of the Ninth International Conference on Future Internet Technologies, ser. CFI ’14. New York, NY, USA: ACM, 2014, pp. 16:1–16:2. http://dx.doi.org/10.1145/2619287.2619303

(2) N. M. SAHRI and Koji OKAMURA, “Openflow Path Fast Failover Fast Convergence Mechanism,” Proceedings of the 38th Asia Pacific Advanced Network (APAN) on Research Network Workshop, pp. 23-28 ISSN: 2227-3026, DOI: http://dx.doi.org/10.7125/APAN.38.4, August 2014.

(3) N. M. SAHRI and Koji OKAMURA, “Load Sensitive Forwarding for Software Defined Networking – Openflow Based,” ACSIJ Advances in Computer Science: International Journal, Vol. 3, Issue 6, No.12, ISSN: 2322-5157, Journal published November 2014

(4) N. M. SAHRI and Koji OKAMURA, "Adaptive Anomaly Detection for SDN", Proceedings of the 40th Asia Pacific Advanced Network (APAN) on Research Network Workshop, pp. 55-59 ISSN: 2227-3026, DOI: http://dx.doi.org/10.7125/40.9 , August 2015.

(5) N. M. SAHRI and Koji OKAMURA, "Collaborative Spoofing Detection and Mitigation - SDN based looping authentication for DNS services", COMPSAC 2016: The 40th IEEE Computer Society International Conference on Computers, Software & Applications. Atlanta, Georgia, USA - June 10-14, 2016

(6) N. M. SAHRI and Koji OKAMURA, “Protecting DNS services from IP spoofing - SDN collaborative authentication approach” in Proceedings of the 11th International Conference on Future Internet Technologies, ser. CFI ’16. Nanjing, China: ACM, 2016

(7) N. M. SAHRI and Koji OKAMURA, "Adaptive Query Rate for Anomaly Detection with SDN", International Journal of Computer Science and Network Security, Vol. 16, No. 6, 2016, ISSN: 1738-7906, Journal published June 2016

(8) N. M. SAHRI and Koji OKAMURA, " *CAuth* – Protecting DNS application from spoofing attacks", International Journal of Computer Science and Network Security, Vol. 16, No. 6, 2016, ISSN: 1738-7906, Journal published June 2016