# APPLICATION DEVELOPMENT FOR THE NETWORK NODES OF SOFTWARE-DEFINED NETWORKS

---

A Thesis

Presented to

the Faculty of the Department of Computer Science

University of Houston

---

In Partial Fulfillment

of the Requirements for the Degree

Masters of Science

---

By

Praveen Mala

December, 2012

# APPLICATION DEVELOPMENT FOR THE NETWORK NODES OF SOFTWARE-DEFINED NETWORKS

Praveen Mala

APPROVED:

Dr. Jaspal Subhlok, Chairman
Dept. of Computer Science

Dr. Deniz Gurkan
College of Technology

Dr. Lennart Johnsson
Dept. of Computer Science

Dean, College of Natural Sciences and Mathematics

# APPLICATION DEVELOPMENT FOR THE NETWORK NODES OF SOFTWARE-DEFINED NETWORKS

---

An Abstract of a Thesis

Presented to

the Faculty of the Department of Computer Science

University of Houston

---

In Partial Fulfillment

of the Requirements for the Degree

Masters of Science

---

By

Praveen Mala

December, 2012

# Abstract

Open Flow is a protocol that enables software-defined networking towards flexible, scalable, and programmable network architectures. In classical switches the control plane and the data plane are built into the operating system and the implementation is vendor specific with no opportunity for programmability. Open Flow protocol enables the separation of the control plane from the switch and moves it to a central server called controller where all the routing decisions are made. The controller is a programmable unit with a centralized visibility of the network. This architecture opens new opportunities for application development on the controller according to the user needs such as security and quality of service (QoS).One bottleneck observed in this approach is the communication between the switch and the controller due to slow processing central processing units (CPUs) of the switch. The other problem is the rigidity observed in the TCAMs (ternary content addressable memory elements) due to ASIC (application-specific integrated circuit) limitations which cannot perform flexible match functions such as Layer5 and Layer7 match. A split data plane (SDP) architecture has been proposed to address this problem by introducing programmable data plane. A multicore network processing unit (NPU) is housed in the same switch platform together with traditional TCAM-matching section. SDP unit can support flexible match functions and programming. By introducing the programming at the switch level we are opening the network nodes for building applications which will change the way network programming has been realized. The focus of the thesis is the development of applications on SDP towards more flexible and higher performance networks.

# Acknowledgements

I would like to thank my advisors Dr. Jaspal Subhlok, Dr. Deniz Gurkan, and Dr. Lennart Johnsson for giving me this opportunity to work on cutting edge technology, their sound advice, and their great efforts during my research at the University of Houston. I am grateful to Rajesh Narayanan from DELL, Michael Blair Wever from Cavium and Fahd Gilani from Xflow Research for guiding me through my research and helping me with the technical difficulties. I am thankful to my friends at UH for their moral support and help, which made my study enjoyable. I would like to thank the faculty in the Department of Computer Science for their assistance and support.

# Contents

# List of Figures

# Chapter 1

# Introduction

Software Defined Networking(SDN) is an emerging technology standard enabling flexible, scalable, and programmable network architectures. SDN's rapid innovation potential is constrained by current merchant silicon hardware scale and flexibility limitations. These limitations can be solved by porting the widely used OpenvSwith (OVS) software stack to merchant platforms, which introduces a new OpenFlow switch architecture called split dataplane(SDP) [1]. The focus of this thesis is investigation of application development opportunities for dataplane programmability on SDP.

## 1.1 Motivation

The security of the home networks can be easily compromised due to complex configurations, which are difficult for a normal home user. Today security in internet is provided by ISPS by implementing the security policies at the network core, which

is not an efficient way of solving the problem [2]. The two major problems while implementing such security algorithms at network core [2] (ADS in this context) are low detection rates and inability to run them at line rates. A home network router is the ideal location in network for detecting and avoiding most of these problems since the problem can be solved right at its roots but today's home network routers are not programmable. The programmable SDN routers can solve the problem by implementing the anomaly detection algorithms at the controller. The major advantage of SDN is the standardized programmability.

## 1.2 Contribution of this Thesis

SDN/OpenFlow is emerging as one of the most promising and disruptive networking technologies of the recent years. SDN allows us to realize new capabilities and address persistent problems with networking one such problems which can be solved by the OpenFlow technology which was not possible in the legacy switches is the DOS attack detection and prevention [2]. The existing OpenFlow switch architectures allows us to mitigate this attack by implementing the anomaly detection algorithms at the controller of the OpenFlow switch [2]. This approach of programmability at the controller is prone to some performance problems [1] in achieving the data forwarding speeds at line rates. The SDP architecture [1] provides a new platform allowing us to introduce the programmability at the dataplane of the switch increasing the flexibility of the dataplane to achieve processing at high data rates. The focus of this thesis is to implement the SYN Flood detection algorithm at the dataplane of the switch to prove the programmability of the dataplane.

## 1.3 Thesis Outline

The thesis is divided in to following chapters. Chapter 2 introduces the OpenFlow switch architecture, SDN network overview, and the advantages of software-defined networking. In chapter 3 we discuss the previous attempts made in detecting the DOS attack at open flow switches how this can be improved. Chapter 4 explains the split dataplane architecture (SDP), its advantages and a detailed description of implementing the SYN-flood detection on the dataplane of SDP switch. Chapter 5 provides a walkthrough of the experimental setup and the test procedure. In chapter 6 we present the results as a proof of concept and chapter 7 we conclude the project.

# Chapter 2

# Background and Definitions

## 2.1 Regular Switch Architecture

In the existing routers the dataplane and the control plane are tightly coupled as shown in the figure 2.1 [3].

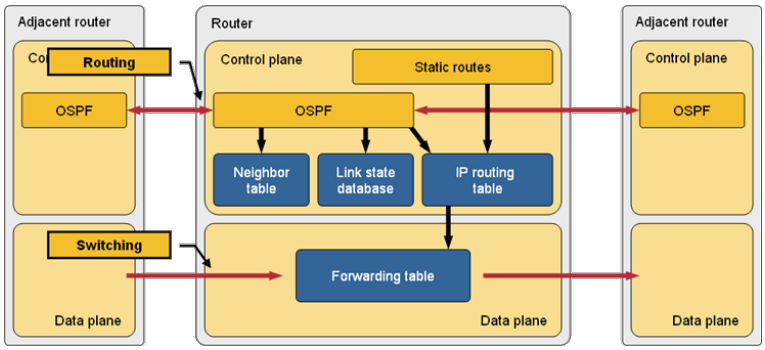


Figure 2.1: Relationships between Control and Dataplanes [3]

Dataplane performs the packet forwarding and the control plane manages the exchange of the routing information.

## 2.2 OpenFlow Switch Architecture

In a OpenFlow switch the control plane and the data plane are separated and the control plane is moved to a central location called the controller as shown in figure 2.2. OpenFlow Switch consists of a flow table, which performs packet lookup and forwarding, and a secure channel to an external controller. OpenFlow is the protocol which enables a secure communication between the switch and the controller [4]. OpenFlow allows to program the FlowTable in the different switches and routers of the OpenFlow network.

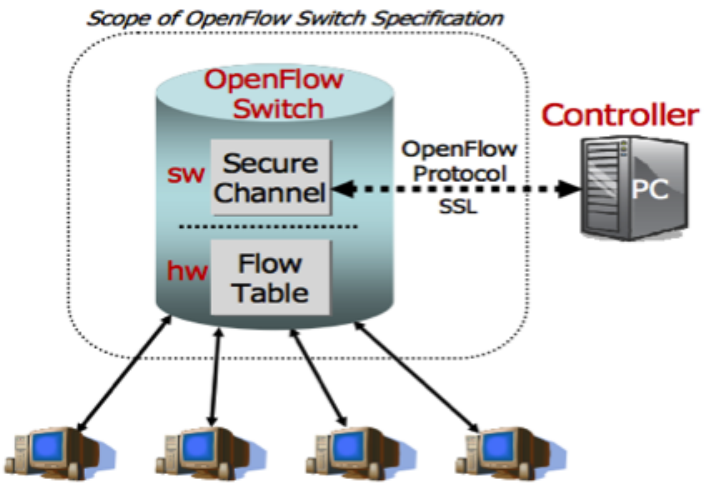


Figure 2.2: Controller and OpenFlow Switch

Flow table: Flow table consists of the Header fields counters and actions

Header fields: to match against packets

Counters: to update for matching packet

Actions: to apply to matching packets

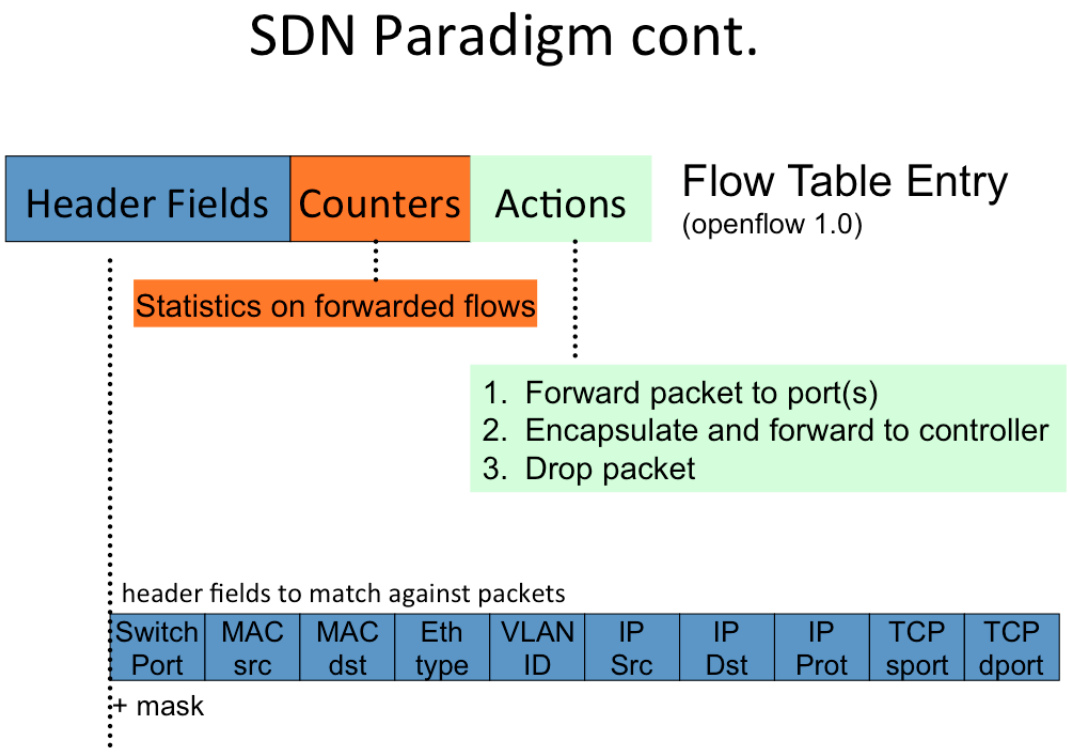


Figure 2.3: Flow Table Entry

Packet processing in a OpenFlow switch:

When a packet from the network reaches the OpenFlow, switch the Header fields of the packet are parsed to match against the flow table entries as shown in figure 2.3. If a flow entry exists in the flow table which matches the new packet it is forwarded according to the actions field of the flow else the packet is forwarded to the controller through the secure channel and the controller pushes the new flow with a corresponding action in to the flow table.

## 2.3 OpenFlow Controller Architecture

The OpenFlow controller acts as a central control plane for one or a group of OpenFlow switches. Any server with a controller instance can act as controller. All the control plane decisions such as routing are performed by the controller. There are many proprietary and opensource OpenFlow controllers available implemented in different languages ex:[Flood light (java),NOX(c++),POX(python).]. The user can pick any controller of choice. The beauty of the controller is its flexibility to modify the control plane modules according to the user requirements. We can also write applications on top of the controller or applications which interact with the controller very easily. The open source controllers have extreme online support making them very adoptable. Floodlight is chosen as the controller in this project.

## 2.4 SDN Network Overview

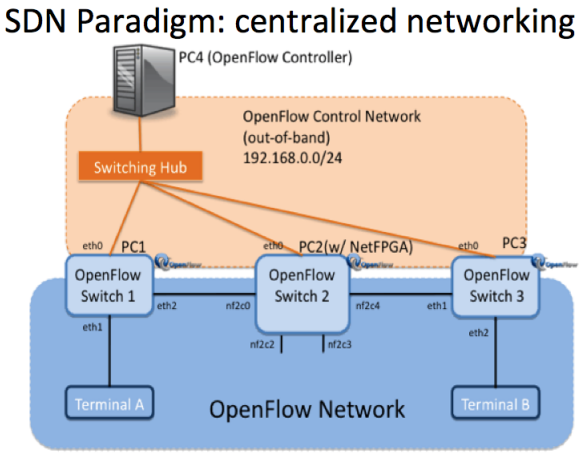


Figure 2.4: SDN Network [4]

Figure 2.4 [4] gives a overview of the SDN paradigm with centralized networking any OpenFlow enabled device can be connected to the controller with the OpenFlow protocol. This provides a centralized visibility of the network and provides unified control over multiple vendor specific devices.

## 2.5 Advantages of Software Defined Networking

Programmability: SDN allows to program the network in real time according to the network dynamic requirements. The controller is a programmable unit and any network programmer can develop applications on top of the controller such as encryption, Anamoly Detection, and peer-to-peer traffic detection according to network requirements.

Granular control: The network policies can be dynamically implemented in a very granular level due to the flow based control mechanism of the OpenFlow network.

Security: Most of the network failures are due to configuration or policy inconsistencies caused when a new node is added or removed from a network. The need for individual configuration and policy implementation on the network devices can be eliminated with SDN architecture which increases the reliability of the of the network. SDN allows a dynamic network architecture to ensure high performance, security, and reliability [5].

# Chapter 3

# Related Work

## 3.1 Traffic Anomaly Detection Techniques

A variety of anomaly detection algorithms [6] [7] [8] [9] have been proposed and implemented at the network core to detect the traffic anomalies, but these implementations have very low detection rates, or the systems with high detection rates have very high amount of false positives, making them not very adoptable for the current internet. The other problem with these implementations is they can not be executed at line rates . Packet and flow sampling [10] [11] [12] techniques are used to solve this problem, but this degrades the detection rates by eliminating the important traffic features. The above two problems can be solved if the anomalies can be detected close to the sources [2]. A SDN switch provides the opportunity to implement these anomaly detection algorithms on the switch allowing to detect the anomalies close to the sources.

## 3.2 Threshold Random Walk with Credit-based Rate Limiting TRW-CB

The TRW-CB [7] algorithm detects the SYN Flooding from a host by using the fact that the probability of a connection being a success should be much higher for a benign host than for a malicious one. This can be achieved by performing the likelihood ratio test. For each internal host the algorithm maintains a queue for connection initiations (TCP SYNs) which are yet to receive a response (SYN-ACK). When the connection receives a TCP RST or a connection timeout the connection dequeues it from the queue and increases the likelihood ratio of the host, which initiated the connection. When there is a successful connection the likelihood ratio is decreased. When the likelihood ratio for a particular host reaches a certain threshold then it is declared as infected.

## 3.3 TCP-SYN Flood Detection with TRW-CB on the NOX Controller

The TRW-CB [Threshold Random Walk with Credit-based Rate Limiting] [7] is implemented on the NOX controller to detect and prevent denial of service (DOS)attack with SYN Flooding.The algorithm either uses the connection initiations or replies to them . According to OpenFlow 1.0 any new packet, which does not match a flow entry in the switch, is forwarded to the controller. The TRW-CB instance on the controller monitors the packets if the connection is successful then the flows are

established in the switch to forward the rest of the packets in that session. Consider a internal host Host1 connected to an OpenFlow switch with a NOX controller and a TRW-CB instance is executing on the controller. Host 2 is a external webserver.

1) Host1 sends a TCP- SYN to a external host Host2. The packet reaches the switch, since there are no matching flows in the flow table the packets are forwarded to the controller.

2) The TRW-CB on the NOX controller forwards this packet through the switch without setting flows along with the normal processing by adding Host2 to a list of hosts contacted by Host1 and adds the connection request to the Host1 queue. The external host Host2 can respond in two ways with TCP SYN-ACK or Timeout

TCP SYN-ACK: When a SYN-ACK is received from Host2 to Host1 the switch again forwards to the controller due to no matching flows in the flow table. The TRW-CB instance on the controller installs two flows into the switch, one from Host1 to Host2 and the other from Host2 to Host1, the TRW-CB does the normal processing of removing the the connection request from Host1 queue and decrease host1 likelihood ratio.

Timeout: When a connection times out the TRW-CB does not install any flow entries in the switch and increases the likelihood ratio for Host1. When the likelihood ratio for a particular host reaches a threshold limit, the host is declared as infected.

## 3.4 OpenFlow Switch Hardware

The bench marking results from table 1 of [1] conforms four fundamental limitations of the merchant silicon in the existing openflow swithes.

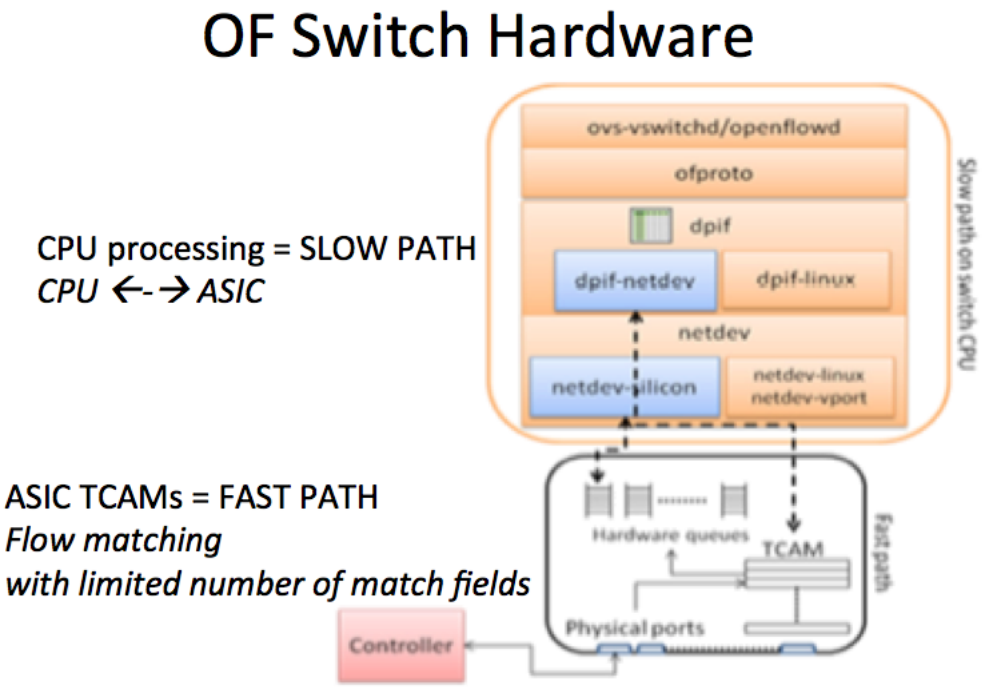


Figure 3.1: OF-Switch[1]

1) The flow action table entries encompass layer 2 and layer 3 till layer 5 (port numbers) with too many fields to match in the hardware.

2) Actions on the matched packets require may require non-traditional programmability expectations.

3) Certain flow definitions require the CPU processing of packets this effects CPU to ASIC bus speed and processing of flow tables on CPU.

4) Delay introduced by the OF-switch to controller communications for the new flow definitions.

To overcome the merchant silicon limitations and the controller-OF switch architecture constraints as shown in figure 3.2, is there a transitional architecture that would enable real network programmability?

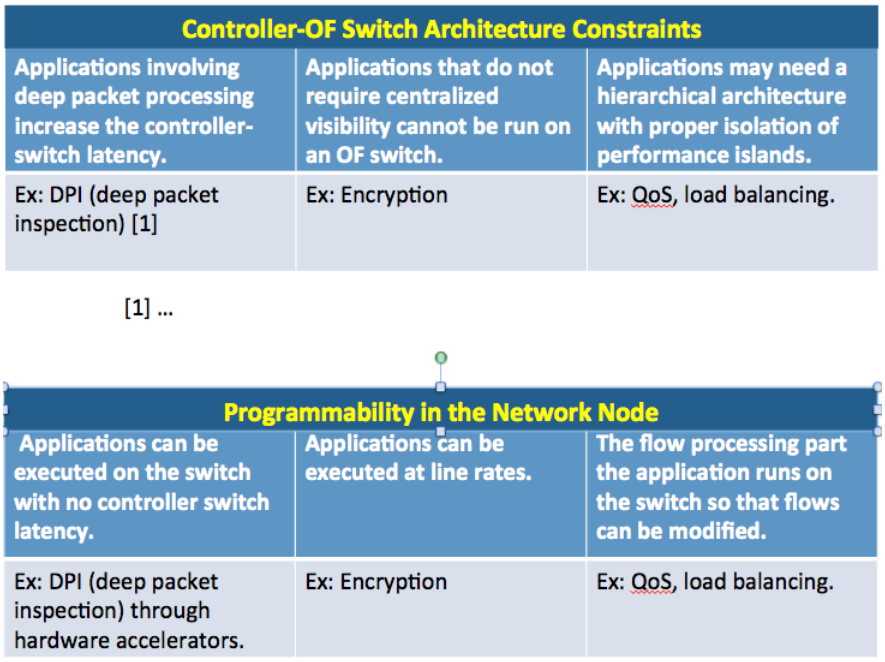


Figure 3.2: OpenFlow Switch Architecture Constraints

To address the above constraints of OpenFlow Switches, Dell introduced a innovative platform that integrated the merchant silicon (ASIC with TCAM entries) with a programmable network processor. This architecture is referred as SDP (split data plane [1]). To realize the problem with above limitations in a real world, we can consider a usecase such as encryption, compression, QOS mentioned above.

# Chapter 4

# Our Approach

## 4.1 Split Dataplane Architecture (SDP)

The SDP switch as shown in the figure 4.1 [1] consists of merchant silicon switch like the traditional switch deployments and programmable subsystem which contains a highend network processor unit with OVS instance. The switch and the subsystem are connected to the controller through OpenFlow API. The high adopted open-vswitch stack is ported on to multicore Network processing unit and the whole unit is integrated in to switch ASIC with a higig interface. A detailed explanation is available in the following sections.

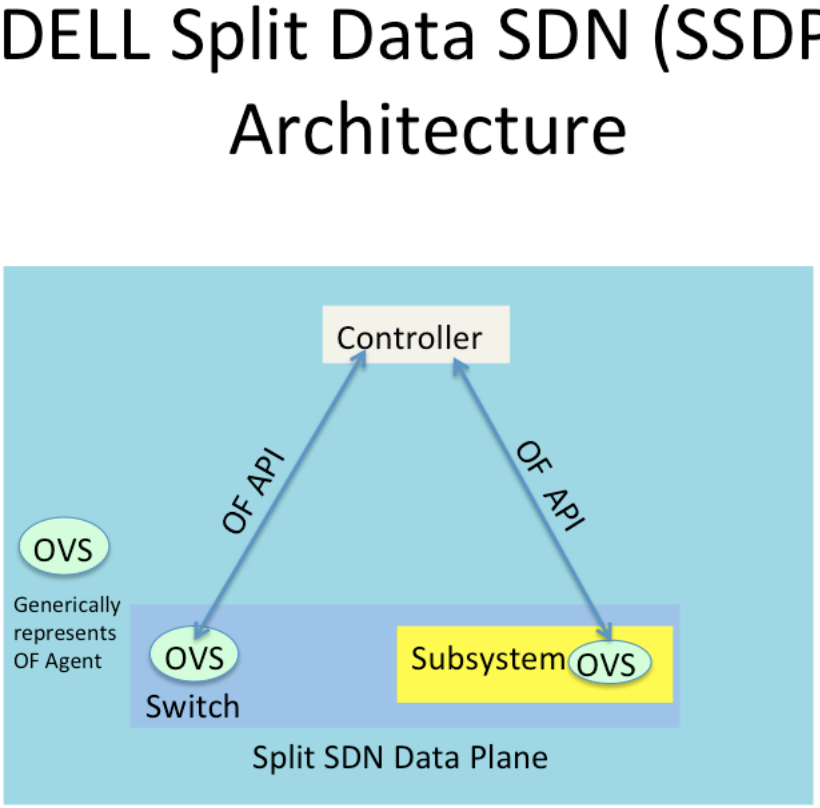


Figure 4.1: Split Data Plane SDP - DELL [1]

## 4.2 SDP MACROFLOW - MICROFLOW

The flow entries in the TCAM of the switch are referred to as the macro-flow and the software-based flowtables on the NPU on the switch are called micro-flows. The TCAM handles the flows, which require regular forwarding while the flows, which require higher granularity matching like layer 4, and layer 5 match are performed at the micro-flow section of the switch. The switch and the subsystem communicate through a 10gig Auxiliary unit interface (XAUI) interface this enables high speed data transfer between the micro-flow and macro-flow as shown in the figure 4.2. The

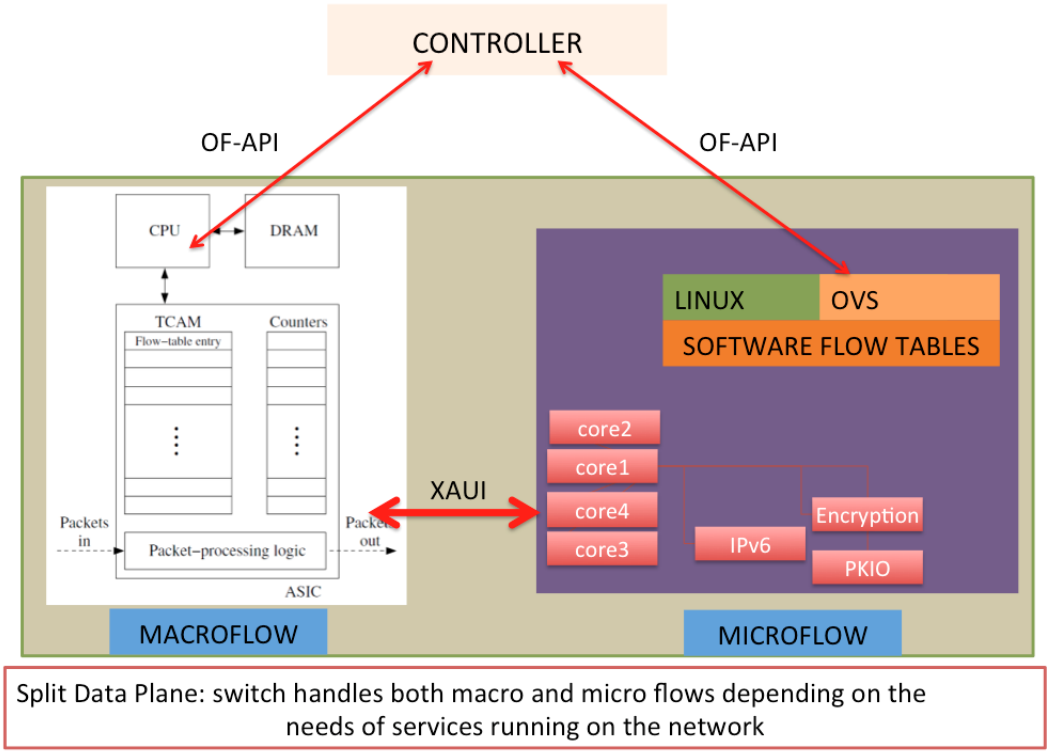


Figure 4.2: SDP Microflow-Macroflow

micro-flow section of the switch consists of OVS stack on a multicore networkprocessor with application specific hardware acceleration units. This hardware acceleration units allows to process the network traffic at high data rates while implementing algorithms such as encryption, compression and DOS detection. The OVS on the micro-flow section of the switch consists of the software flow tables and these flow tables , which are very flexible can be used for exotic flow entries . The micro-flow section of switch can also be used for flow entries which consume a lot of space on the switch such as the ipv6 entries.

## 4.3 TCP-SYN Flood Detection on Programmable Dataplane

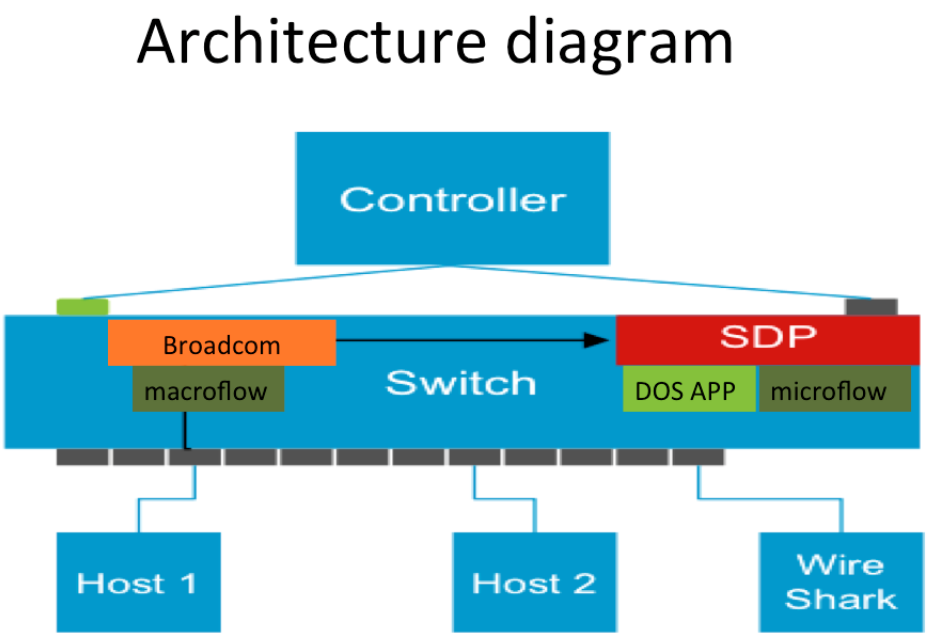


Figure 4.3: DOS(denial of service) Detection Architecture

The Architecture diagram of a proposed security analytics application ie DOS attack detection and prevention on SDP is illustrated in the figure 4.3. The DOS application on the SDP module can monitor the SYN requests and add the drop action field to the flows for the malicious host to prevent the attack. Host 1 is the client machine from where the TCP-SYN are generated. Host 2 is the Apache webserver where the TCP-SYN requests are received and replied.

The Architecture for launching dos attack and detecting it :

Multiple TCP packets with SYN bit set to 1 are generated from a packet generator (hping3) to create multiple sessions With the server. The packets are sent to a destination IP (webserver) with a source ip which is unreachable. The source port in the packet is changed for each of the requests sent by the script. Since the server keep the session open waiting for a ack which it never receives over period there are multiple tcp sessions opened on the server and when they reach a threshold beyond the server can handle the server rejects the next tcp sessions which causes the denial of service.

when a client sends a SYN request to a server it replies back with a SYN/ACK message to the client, [13]. Until the SYN/ACK is acknowledged by the client, The connection remains in half open state(SYN-RECV). The TCB (Transmission control block) data structure is used to save this state in the backlog queue to maintain all the halfopen connections. The memory of the backlog queue is finite which is typically 1024 connections once the backlog queue threshold is reached further connection requests are dropped.

## 4.4 Packet Flow in SDP

1) The macro-flow section of the switch is configured to forward every new packet to the controller which does not have the flow table entry in the switch.

2) At the Floodlight controller we write a application with java class Using Rest API which will detect a TCP packet and while writing a flow entry in the macro-flow section of the switch it adds one more destination port, i.e. port no 26 which is the

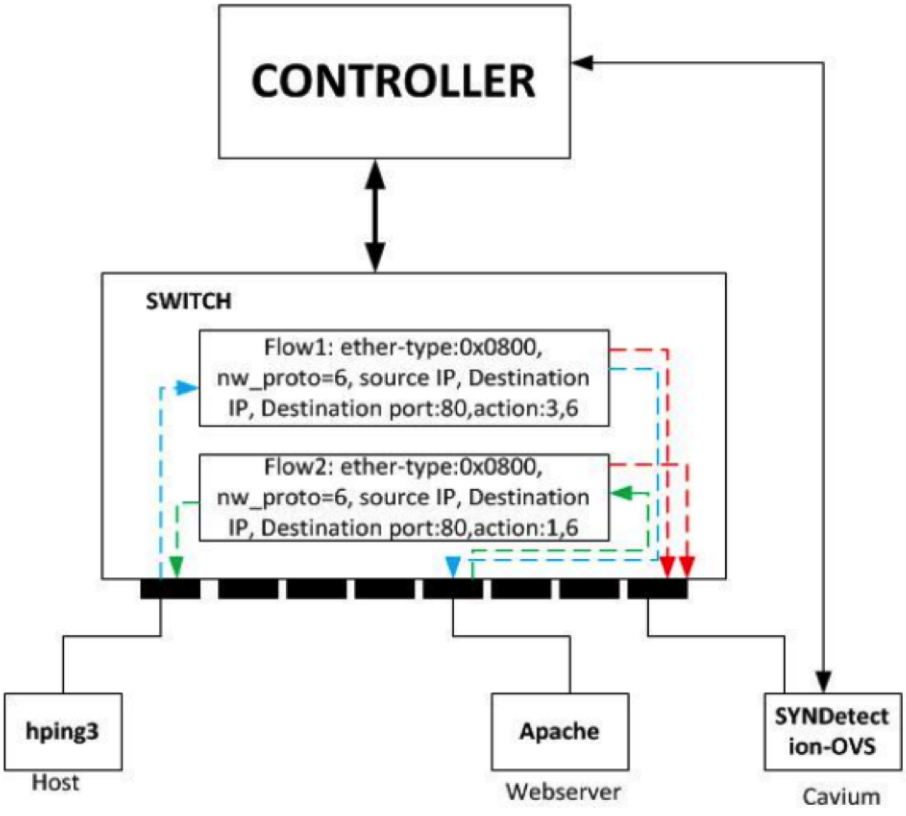


Figure 4.4: PacketFlow - TCP-SYN Flood Detection

XAUI interface .(in short we are mirroring the traffic on the Cavium card)

3) For communication between the client and the server; two flow entrys are made in to the macro-flow section of the switch one from client to the server(FLOW 1) and the other from server to client(Flow 2). Each flow entry is configured to check for the source ip, destination port The header part of the packet should be checked to find if it's a TCP packet(ether-type":"0x0800), destination ip, destination port :80. Flow 1 should check if it's a TCP frame (ether-type), source IP, Destination IP,

Destination port:80 and mirror the traffic on Cavium. Flow 2 should check if it's a TCP frame (ether-type), source IP, Destination IP, Source port :80 and mirror the traffic on Cavium.

4) A instance of the SYN Flood detection algorithm is executed on the Cavium processor. As all the TCP requests and replies are forwarded to the syn flood algorithm the algorithm does the following The algorithm parses every single packet to check if the SYN or SYN-ACK flag is set; If the SYN flag is set the Algorithm creates a counter corresponding to this source once and increments for further SYN requests from the particular source or if the SYN-ACK flag is set the algorithm decrements the counter corresponding to the particular host as show in figure 4.5.

5)When the counter value exceeds a certain threshold limit (25 in our case) the host

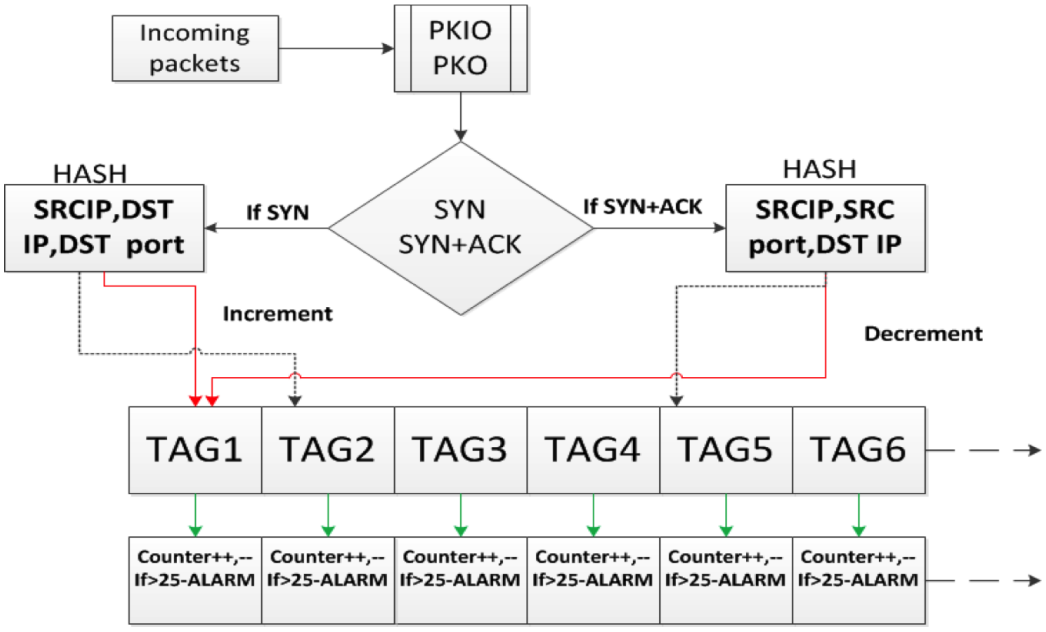


Figure 4.5: SYN Flood Algoritham.

is declared as infected and any further requests from the particular source is blocked.

6) To block the further packets from the particular host, a drop action field is added to the flow corresponding to the particular host(ip).

To implement this the simple executable on the network processor should inform the controller that a drop action field should be added to a particular flow. The curl library in the simple executable is used to push this drop action in to the flow from the OF-controller.

7) The further packets from that particular host are dropped right at the macro-flow section of the switch preventing the SYN Flooding.

# Chapter 5

# Experiment Setup

## 5.1 SDP Switch with a Client and a Server

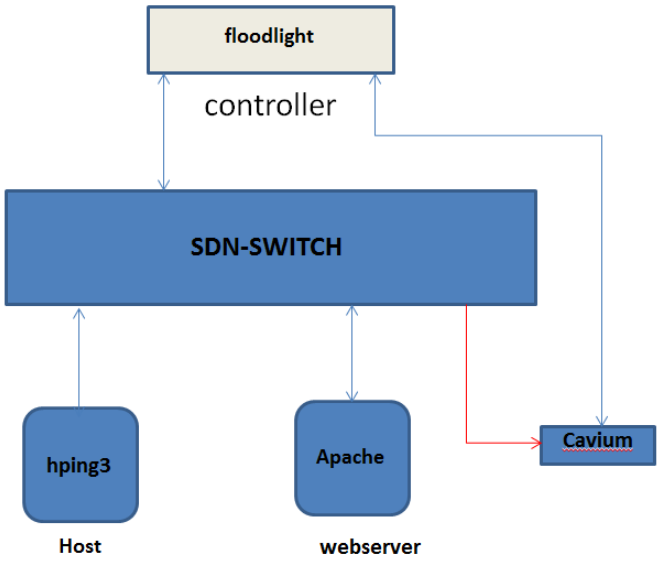


Figure 5.1: Experiment Setup for SYN Flood.

The experiment is setup according to the Architecture diagram shown in figure 5.1.

The hardware used for the experiment:

One openflow switch, three host machines and network processing unit are used for the experiment the specifications of each individual unit is explained below.

Openflow switch: hp procurve 3500 The hp 3500 switch support openflow 1.0 with 48 one gig ports.

Network processor: The Cavium CN5200 is a 4core mips64-based processor with hardware acceleration units such as encryption, compression, hashing, parsing specific to the packet processing.

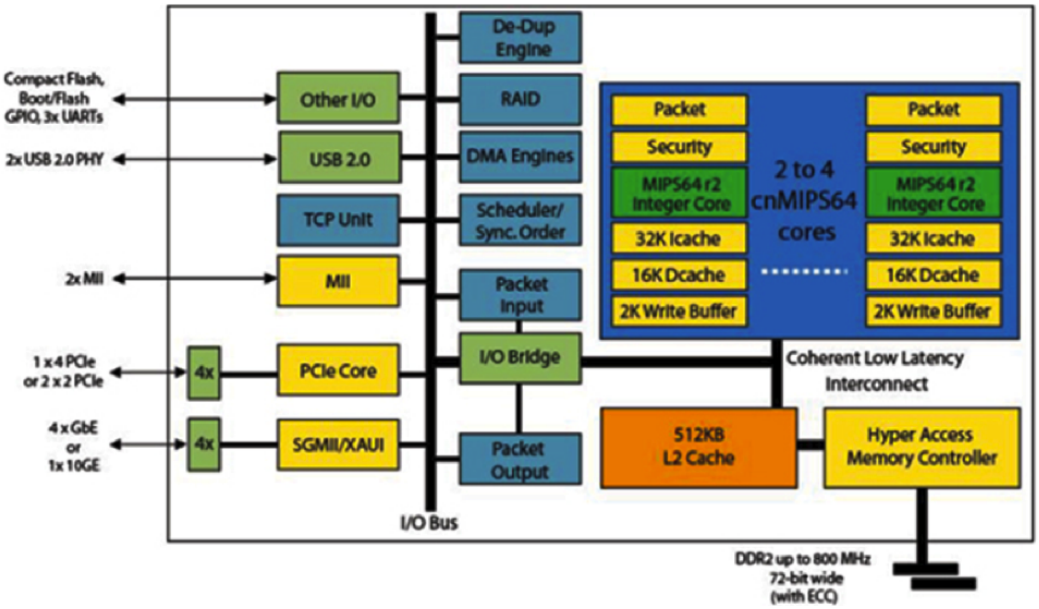


Figure 5.2: Cavium Reference Architecture

Three personal computers :

Hardware specifications of PC:

Host 1: The host is a regular pc with hping3 tool installed on a Fedora 16 operating

system.

hping3: hping is a tcp/ip packet generator and it can generate any kind of spoofed tcp packets with rate limiting. we are using it generate the TCP-SYN packets.

Host 2 (Apache webserver ): The Apache webserver is installed on a Ubuntu host machine. Apache webserver: Apache is a robust and most widely used webserver with a variety of features it has inbuilt open source intrusion detection and prevention engine for web application detection system known as ModSecurity some of the other features include secure socket layer and transport layer security it also has a inbuilt proxy module built within the system.

Host3 (Floodlight controller): The Floodlight controller is installed on regular pc with Fedora16 operating system.

Floodlight controller:

Floodlight is a java-based enterprise class open SDN controller with extensive online support new modules can be easily developed and added to the floodlight controller. The modules interact with the controller using rest API. The flows are automatically pushed by the controller into switch. If the user want to explicitly specify the flow definitions they can be done by writing a new controller module or by using CURL for static flow entries.

Sample static flow entries made through CURL using floodlight:

To list the flows: curl http://192.168.10.2:8080/wm/staticflowentrypusher/list/ 00:14:00:16:b9:0e:06:00/json

To Delete the flows : curl -X DELETE -d '"name":"flow-mod-1"' http://192.168.10.2 :8080/wm/staticflowentrypusher/json

To Push the Flows: curl -d '"switch": "00:14:00:16:b9:0e:06:00", "name":"flow-mod-1", "cookie":"0", "priority":"32768", "ingress-port":"30","active":"true","actions": "output=34,output=26"' http://192.168.10.2:8080/wm/staticflowentrypusher/json

## 5.2 Test Procedure

The experiment is set up as shown in the figure 5.1. After the hardware is setup and the basic connectivity between all the hardware devices is tested. The setup is tested for its normal operation. The http request is sent from the clients web browser(Host1) to the webserver(Host2). The web browser displays the sample page saying it works! as shown in figure 5.3 which is retrieved form the webserver.

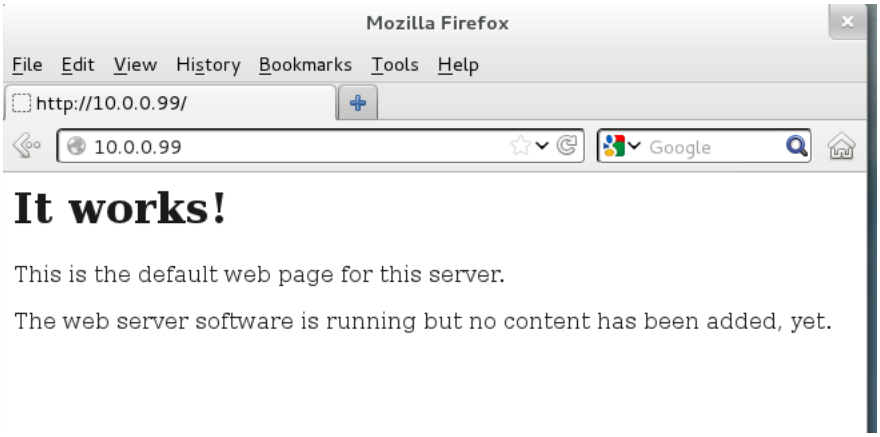


Figure 5.3: Host1 Webpage

We can check the series of messages exchanged between the client and the server on the wireshark either on the client or the server.

After checking the functionality the SYN Flood attack is launched from the host using hping3. The wireshark capture of the SYN Flooding is shown in figure 5.4.

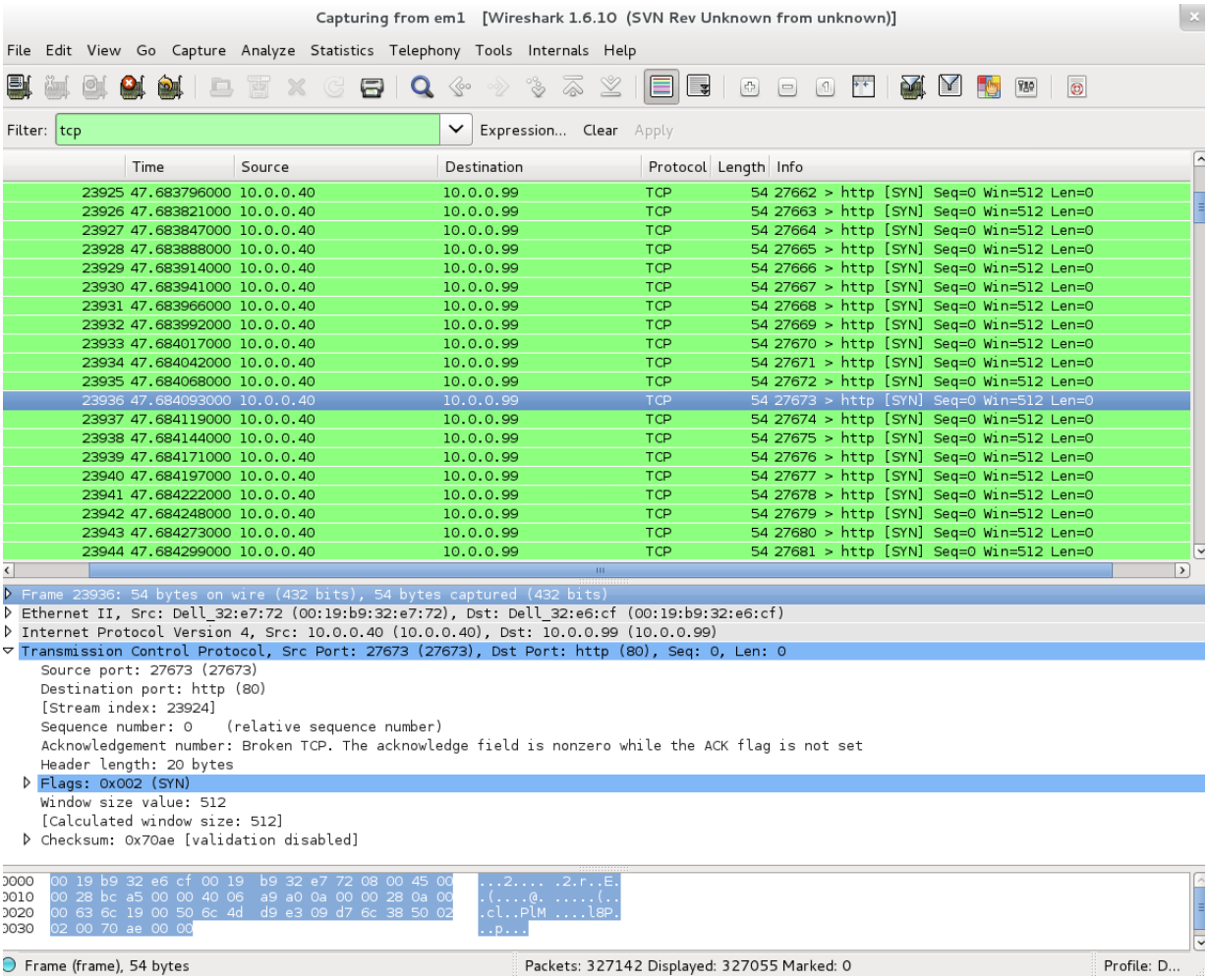


Figure 5.4: TCPSessionFloodingfromHost

## 5.3 Flow Definition

The flow definitions are made in the switch to mirror the traffic on to the cavium processor with TCP SYN Flood detection as shown in the figure below.The SYN

Flooding is detected by the SYN Flooding algorithm on the Cavium and alarm flag should be raised by the Cavium console.

```
HP-E3500yl-48G# show openflow 20 flows

 Openflow flows - VLAN 20


Flow 1
 Incoming Port      : 34                HW acceleration    : No
 Destination MAC    : 000000-000000     Source MAC         : 000000-000000
 VLAN ID            : 65535             VLAN Priority      : 0
 Source IP          : 0.0.0.0           Destination IP     : 0.0.0.0
 IP Protocol        : 0x0000            IP ToS bits        : 0
 Source Port        : 0                 Destination Port   : 0
 Duration           : 256886s secs      Priority           : 32768
 Idle Timeout       : 0 secs            Hard Timeout       : 0 secs
 Packet Count       : 34454             Bytes Count        : 2128242
 Actions            : output:30,output:26

Flow 2
 Incoming Port      : 30                HW acceleration    : No
 Destination MAC    : 000000-000000     Source MAC         : 000000-000000
 VLAN ID            : 65535             VLAN Priority      : 0
 Source IP          : 0.0.0.0           Destination IP     : 0.0.0.0
 IP Protocol        : 0x0000            IP ToS bits        : 0
 Source Port        : 0                 Destination Port   : 0
 Duration           : 256904s secs      Priority           : 32768
 Idle Timeout       : 0 secs            Hard Timeout       : 0 secs
 Packet Count       : 432               Bytes Count        : 47186
 Actions            : output:34,output:26


HP-E3500yl-48G#
```

Figure 5.5: Flow Entries

# Chapter 6

# Result

## 6.1 SYN Flood Detection on the dataplane of the switch

The SYN Flood algorithm on the data plane of the switch detects the SYN packets on the switch and when the counter value reaches the Threshold It displays the SYN Flooding message on the Cavium Console as shown in figure 6.1 proving the programmability of data plane.

```
Waiting for another core to setup the IPD hardware...Done
- # ifconfig pow0 192.168.1.2
- # PP0:~CONSOLE-> neither syn nor syn/ack
PP0:~CONSOLE-> Received 90 byte packet from Linux. Sending to PKO
PP0:~CONSOLE-> neither syn nor syn/ack
PP0:~CONSOLE-> Received 78 byte packet from Linux. Sending to PKO
PP0:~CONSOLE-> neither syn nor syn/ack
PP0:~CONSOLE-> Received 70 byte packet from Linux. Sending to PKO
PP0:~CONSOLE-> neither syn nor syn/ack
PP0:~CONSOLE-> Received 90 byte packet from Linux. Sending to PKO
PP0:~CONSOLE-> neither syn nor syn/ack
PP0:~CONSOLE-> Received 70 byte packet from Linux. Sending to PKO

- # PP0:~CONSOLE-> neither syn nor syn/ack
PP0:~CONSOLE-> Received 70 byte packet from Linux. Sending to PKO

- # ifconfig eth0 promisc up
Device eth0 entered promiscuous mode
- # PP0:~CONSOLE-> neither syn nor syn/ack
PP0:~CONSOLE-> Received 231 byte packet. Sending to Linux.
PP0:~CONSOLE-> neither syn nor syn/ack
PP0:~CONSOLE-> Received 60 byte packet. Sending to Linux.
PP0:~CONSOLE-> neither syn nor syn/ack
PP0:~CONSOLE-> Received 60 byte packet. Sending to Linux.
PP0:~CONSOLE-> neither syn nor syn/ack
PP0:~CONSOLE-> Received 60 byte packet. Sending to Linux.
PP0:~CONSOLE-> neither syn nor syn/ack
PP0:~CONSOLE-> Received 60 byte packet. Sending to Linux.
```

Figure 6.1: TCP SYN Flood Detection

# Chapter 7

# Conclusion and Future Scope

By considering the results from the above experiment we conclude that our hypothesis of programming the network data plane is possible. The Existing SDP architecture has two openflow connections to the Switch one from the Microflow section and the Other from the Macroflow section since this is a transitional architecture. The future scope of this work is to stabilize the connectivity between the controller and the SDP with single openflow connection.

# Bibliography


[1] R. Narayanan, S. Kotha, A. K. Geng Lin, S. Rizvi, W. Javed, H. Khan, and S. A. Khayam, "Macro-flows and micro-flows: Enabling rapid network innovation through a split sdn dataplane," in *European Workshop on Software Defined Networks*, DELL, 2012.

[2] S. Mehdi, J. Khalid, and S. Khayam, "Revisiting traffic anomaly detection using software defined networking," in *Recent Advances in Intrusion Detection*, pp. 161–180, Springer, 2011.

[3] I. Pepelnjak. `http://wiki.nil.com/Control_and_Data_plane`.

[4] `http://www.openflow.org/documents/openflow-spec-v1.0.0.pdf`.

[5] `http://www.bigswitch.com/sites/default/files/sdn_resources/onf-whitepaper.pdf`.

[6] J. Jung, V. Paxson, A. Berger, and H. Balakrishnan, "Fast portscan detection using sequential hypothesis testing," in *Security and Privacy, 2004. Proceedings. 2004 IEEE Symposium on*, pp. 211–225, IEEE, 2004.

[7] J. Mikians, P. Barlet-Ros, J. Sanjuas-Cuxart, and J. Solé-Pareta, "A practical approach to portscan detection in very high-speed links," in *Passive and Active Measurement*, pp. 112–121, Springer, 2011.

[8] J. Jung, R. Milito, and V. Paxson, "On the adaptive real-time detection of fast-propagating network worms," *Detection of Intrusions and Malware, and Vulnerability Assessment*, pp. 175–192, 2007.

[9] J. Twycross and M. Williamson, "Implementing and testing a virus throttle," in *Proceedings of the 12th USENIX Security Symposium*, vol. 285, p. 294, 2003.

[10] `http://www.endace.com/endace-high-speed-packet-capture-probes.html`.

[11] M. Kim, H. Kong, S. Hong, S. Chung, and J. Hong, "A flow-based method for abnormal network traffic detection," in *Network Operations and Management Symposium, 2004. NOMS 2004. IEEE/IFIP*, vol. 1, pp. 599–612, IEEE, 2004.

[12] R. Fujimaki, T. Yairi, and K. Machida, "An approach to spacecraft anomaly detection problem using kernel feature space," in *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, pp. 401–410, ACM, 2005.

[13] H. Wang, D. Zhang, and K. Shin, "Detecting syn flooding attacks," in *INFOCOM 2002. Twenty-first Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3, pp. 1530–1539, IEEE, 2002.