IOSR Journal of Engineering (IOSRJEN) **www.iosrjen.org**
ISSN (e): 2250-3021, ISSN (p): 2278-8719
Vol. 10, Issue 7, July 2020, ||Series -II|| PP 06-15

Study of Controllers in SDN

[1]Dr. Girish Ashok Kulkarni, [2]Mrs. Sulbha Manoj Shinde

[1]Head and Associate Professor, Department of Electronics and telecommunication Engineering,Shri Sant Gadge Baba College of Engineering and Technology, Bhusawal, India

[2] Research Scholar, Department of Electronics and telecommunication Engineering,Shri Sant Gadge Baba College of Engineering and Technology, Bhusawal, India

Received 09 July 2020; Accepted 25 July 2020

Abstract: Software Defined Networks offer flexible and intelligent network operations by splitting a traditional network into a centralized control plane and a programmable data plane. The intelligent control plane is responsible for providing flow paths to switches and optimizes network performance. The controller in the control plane is the fundamental element used for all operations of data plane management. Hence, the performance and capabilities of the controller itself are extremely important. Furthermore, the tools used to benchmark their performance must be accurate and effective in measuring different evaluation parameters. There are dozens of controller proposals available in existing literature. However, there is no quantitative comparative analysis for them. In this article, we present a comprehensive qualitative comparison of different SDN controllers. We also discuss the two categories of the controller along with some popular available controller. For each controller , we discussed the architectural overview, design aspects and so on. This paper points to the major state-of-the-art controllers used in industry and academia. Our review work covers major popular controllers used in SDN paradigm.

Keywords: *software defined networks (SDN), SDN controller*

I. INTRODUCTION

Software-Defined Networking (SDN) is an emerging architecture that is dynamic, manageable, cost-effective, and adaptable, making it ideal for the high-bandwidth, dynamic nature of today's applications. This architecture decouples the network control and forwarding functions enabling the network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services. The OpenFlow® protocol is a foundational element for building SDN solutions.

The SDN Architecture is:

- Directly Programmable

Network control is directly programmable because it is decoupled from forwarding functions.

- Agile

Abstracting control from forwarding lets administrators dynamically adjust network-wide traffic flow to meet changing needs.

- Centrally Managed

Network intelligence is (logically) centralized in software-based SDN controllers that maintain a global view of the network, which appears to applications and policy engines as a single, logical switch.

- Programmatically Configured

SDN lets network managers configure, manage, secure, and optimize network resources very quickly via dynamic, automated SDN programs, which they can write themselves because the programs do not depend on proprietary software.

- Open Standards-Based And Vendor-Neutral

When implemented through open standards, SDN simplifies network design and operation because instructions are provided by SDN controllers instead of multiple, vendor-specific devices and protocols.

Software-Defined Networking (SDN) is a network architecture approach that enables the network to be intelligently and centrally controlled, or 'programmed,' using software applications. This helps operators manage the entire network consistently and holistically, regardless of the underlying network technology.

Enterprises, carriers, and service providers are being surrounded by a number of competing forces. The monumental growth in multimedia content, the explosion of cloud computing, the impact of increasing mobile usage, and continuing business pressures to reduce costs while revenues remain flat are all converging to wreak havoc on traditional business models.

To keep pace, many of these players are turning to SDN technology to revolutionize network design and operations.

SDN enables the programming of network behavior in a centrally controlled manner through software applications using open APIs. By opening up traditionally closed network platforms and implementing a common SDN control layer, operators can manage the entire network and its devices consistently, regardless of the complexity of the underlying network technology.

SDN has got an architecture which comprises of mainly three layers. They are Application Layer,Control Layer and Infrastructure Layer as mentioned in Fig 1. The Application Layer can program explicitly in this layer to communicate with the network. Further it also helps to get the abstract view of the network by collecting the data from the control plane for taking decisions. It consists of an abstract view of business applications to program explicitly. Control Layer is the logical entity where all the instructions are received from other networking components. It has the controller who can control and extract information about the network from the hardware components and make communication possible. Infrastructure layer consists of switches that is used to forward the packets and has an inbuilt flow table to check the incoming and outgoing packet from one host to another. A southbound API connects the controller and the switch and the Northbound API connects the controller and the applications to make communication possible.

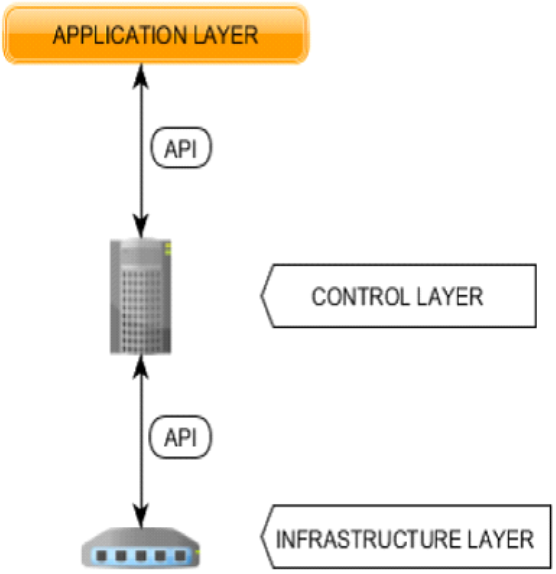

Figure 1: SDN Architecture

There are four critical areas in which SDN technology can make a difference for an organization.

1) Network programmability: SDN enables network behavior to be controlled by the software that resides beyond the networking devices that provide physical connectivity. As a result, network operators can tailor the behavior of their networks to support new services, and even individual customers. By decoupling the hardware from the software, operators can introduce innovative, differentiated new services rapidly—free from the constraints of closed and proprietary platforms.

2) Logically centralize intelligence and control: SDN is built on logically centralized network topologies, which enable intelligent control and management of network resources. Traditional network control methods are distributed. Devices function autonomously with limited awareness of the state of the network. With the kind of centralized control an SDN-based network provides, bandwidth management, restoration, security, and policies can be highly intelligent and optimized—and an organization gains a holistic view of the network.

3) Abstraction of the network: Services and applications running on SDN technology are abstracted from the underlying technologies and hardware that provide physical connectivity from network control. Applications will interact with the network through APIs, instead of management interfaces tightly coupled to the hardware.

4) Openness: SDN architectures usher in a new era of openness—enabling multi-vendor interoperability as well as fostering a vendor-neutral ecosystem. Openness comes from the SDN approach itself. The open APIs support a wide range of applications, including cloud orchestration, OSS/BSS, SaaS, and business-critical networked apps. In addition, intelligent software can control hardware from multiple vendors with open programmatic interfaces like OpenFlow. Finally, from within the SDN, intelligent network services and applications can run within a common software environment.

A key advantage of SDN technology is the ability for network operators to write programs that utilize SDN APIs and give applications control over network behavior. SDN allows users to develop network-aware

applications, intelligently monitor network conditions, and automatically adapt the network configuration as needed.

II. SDN CONTROLLER

An SDN controller is an application in a software-defined networking (SDN) architecture that manages flow control for improved network management and application performance. The SDN controller platform typically runs on a server and uses protocols to tell switches where to send packets.

SDN controllers direct traffic according to forwarding policies that a network operator puts in place, thereby minimizing manual configurations for individual network devices. By taking the control plane off of the network hardware and running it instead as software, the centralized controller facilitates automated network management and makes it easier to integrate and administer business applications. In effect, the SDN controller serves as a sort of operating system (OS) for the network.

The controller is the core of a software-defined network. It resides between network devices at one end of the network and applications at the other end. Any communication between applications and network devices must go through the controller.

The controller communicates with applications -- such as firewalls or load balancers -- via northbound interfaces. The Open Networking Foundation (ONF) created a working group in 2013 focused specifically on northbound APIs and their development. The industry never settled on a standardized set, however, largely because application requirements vary so widely.

The controller talks with individual network devices using a southbound interface, traditionally one like the OpenFlow protocol. These southbound protocols allow the controller to configure network devices and choose the optimal network path for application traffic. OpenFlow was created by ONF in 2011.

- *Pros and cons of SDN controllers*

One major benefit of SDN controllers is that the centralized controller is aware of all the available network paths and can direct packets based on traffic requirements. Because of the controller's visibility into the network, it can automatically modify traffic flows and notify network operators about congested links.

Companies can -- and should -- use more than one controller, adding a backup for redundancy. Three seems to be a common number among both commercial and open source SDN options. This redundancy will enable the network to continue running in the event of lost connectivity or controller susceptibility.

The controller acts as a single point of failure, so securing it is pivotal to any software-defined network. Whoever owns the controller has access to the entire network. This means network operators should create security and authentication policies to ensure only the right people have access.

- *SDN controller vendors*

Vendors that offer SDN controllers include the following:

- Big Switch Networks
- Cisco
- Cumulus Networks
- Hewlett Packard Enterprise
- Juniper Networks
- Nuage Networks
- Pica8
- Pluribus Networks
- VMware

III. CONTROLLER CATEGORIES

Software Defined networking makes use of two types of controllers which are Centralized and Distributed.

- *Centralized Controller*

Centralized Controllers implement all control plane logic at a single location. In such controller, the single server takes care of all control plane activities. The main benefit of such controller is simplicity and management as they provide a single point of control. However, they suffer from scalability issue because each server has limited capacity to deal with data plane devices.

- *Distributed Controller*

In comparison to centralized controllers, distributed controllers have advantages in case of scalability and high-performance during increase demand of requests.

Some of the popular SDN controllers are POX[1],Ryu[2],OpenDayLight[3],NOX[4],ONOS and so forth. They would manage and configure the available switches dynamically according to the necessity of the user. These controllers would control all the operations for the forwarding of the packets from the source to the

destination using interfaces like NorthBound API and SouthBound API. NorthBound API such as REST, acts as a interface between the Application Layer and Control Layer which is used for the implementation of business policy over application layer. This also uses service policy to state traffic behaviour. The interface between the Control Layer and the Infrastructure Layer is the Southbound API that has got a forwarding rule after installation of the controller. Some of the southbound API are Opflex [5], OpenFlow [6], NETCONF[7], POF [8], ForCES [9] etc.

IV. POX CONTROLLER

POX provides a framework for communicating with SDN switches using either the OpenFlow or OVSDB protocol. Developers can use POX to create an SDN controller using the Python programming language. It is a popular tool for teaching about and researching software defined networks and network applications programming.1

POX can be immediately used as a basic SDN controller by using the stock components that come bundled with it. Developers may create a more complex SDN controller by creating new POX components. Or, developers may write network applications that address POX's API. In this tutorial, we only briefly discuss programming for POX.

V. OPEN DAY LIGHT CONTROLLER

The OpenDaylight controller is JVM software and can be run from any operating system and hardware as long as it supports Java. The controller is an implementation of the Software Defined Network (SDN) concept and makes use of the following tools:

Maven: OpenDaylight uses Maven for easier build automation. Maven uses pom.xml (Project Object Model) to script the dependencies between bundle and also to describe what bundles to load and start.

OSGi: This framework is the back-end of OpenDaylight as it allows dynamically loading bundles and packages JAR files, and binding bundles together for exchanging information.

JAVA interfaces: Java interfaces are used for event listening, specifications, and forming patterns. This is the main way in which specific bundles implement call-back functions for events and also to indicate awareness of specific state.

REST APIs: These are northbound APIs such as topology manager, host tracker, flow programmer, static routing, and so on.

The controller exposes open northbound APIs which are used by applications. The OSGi framework and bidirectional REST are supported for the northbound APIs. The OSGi framework is used for applications that run in the same address space as the controller while the REST (web-based) API is used for applications that do not run in the same address space (or even the same system) as the controller. The business logic and algorithms reside in the applications. These applications use the controller to gather network intelligence, run its algorithm to do analytics, and then orchestrate the new rules throughout the network. On the southbound, multiple protocols are supported as plugins, e.g. OpenFlow 1.0, OpenFlow 1.3, BGP-LS, and so on. The OpenDaylight controller starts with an OpenFlow 1.0 southbound plugin. Other OpenDaylight contributors begin adding to the controller code. These modules are linked dynamically into a Service Abstraction Layer (SAL).

The SAL exposes services to which the modules north of it are written. The SAL figures out how to fulfill the requested service irrespective of the underlying protocol used between the controller and the network devices. This provides investment protection to the applications as OpenFlow and other protocols evolve over time. For the controller to control devices in its domain, it needs to know about the devices, their capabilities, reachability, and so on. This information is stored and managed by the Topology Manager. The other components like ARP handler, Host Tracker, Device Manager, and Switch Manager help in generating the topology database for the Topology Manager.

VI. NOX CONTROLLER

NOX is the original OpenFlow controller. It serves as a network control platform, that provides a high level programmatic interface for management and the development of network control applications. Its system-wide abstractions turn networking into a software problem.

Following are various NOX versions:
NOX — initially developed by Nicira Networks and now owned by VMware, along with OpenFlow — was first introduced to the community in 2009. This was later divided into multiple different lines of development:
NOX classic: This is the version that has been available under the GPL since 2009.
NOX: The "new NOX." Only contains support for C++ and has lesser applications than the classic; however, this version is faster and has better codebase.
POX: Typically termed as NOX's sibling. Provides Python support.

Figure 2 below depicts the architecture of NOX. The NOX core provides helper methods, such as network packet process, threading and event engine, in addition to OpenFlow APIs for interacting with Open Flow switches, and I/O operations support.

At the top, we have applications: Core, Net and Web. However, with the current NOX version, there are only two core applications: OpenFlow and switch, and both network and web applications are missing. The middle layer shows the in-built components of NOX. The connection manager, event dispatcher and OpenFlow manager are self-explanatory, whereas the dynamic shared object (DSO) deployer basically scans the directory structure for any components being implemented as DSOs.

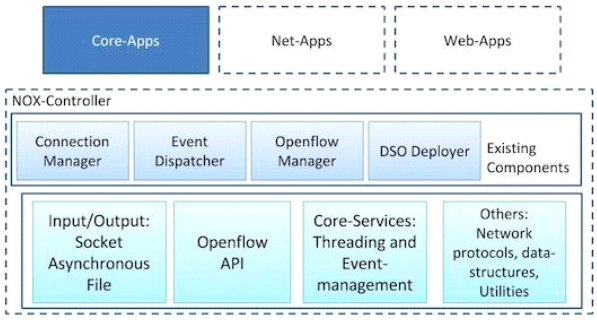

Figure 2: NOX architecture

The event system is another important concept of the NOX controller.
An event represents a low-level or high-level event in the network. Typically the event only provides the information, and processing of that information is deferred to handlers. Many events roughly correlate to something which happens on the network that may be of interest to a NOX component. These components, typically, consists a set of event handlers. In this sense, events drive all execution in NOX.

VII. ONOS

Open Network Operating System (ONOS®) is the leading open source SDN controller for building next-generation SDN/NFV solutions.
ONOS was designed to meet the needs of operators wishing to build carrier-grade solutions that leverage the economics of white box merchant silicon hardware while offering the flexibility to create and deploy new dynamic network services with simplified programmatic interfaces. ONOS supports both configuration and real-time control of the network, eliminating the need to run routing and switching control protocols inside the network fabric. By moving intelligence into the ONOS cloud controller, innovation is enabled and end-users can easily create new network applications without the need to alter the dataplane systems.
The ONOS platform includes:

- A platform and a set of applications that act as an extensible, modular, distributed SDN controller.
- Simplified management, configuration and deployment of new software, hardware & services.
- A scale-out architecture to provide the resiliency and scalability required to meet the rigors of production carrier environments.

VIII. CISCO OPFLEX

Cisco OpFlex is a southbound protocol in a software-defined network (SDN) designed to facilitate the communications between the SDN Controller and the infrastructure (switches and routers). The goal is to create a standard that enables policies to be applied across physical and virtual switches/routers in a multi-vendor environment.

On the surface, Cisco OpFlex sounds a lot like OpenFlow, an open standard that enables the SDN Controller to interact with the infrastructure, however, it is quite different in terms of the scope of its capabilities.

While OpenFlow centralizes all the network control functions on the SDN Controller, Cisco OpFlex focuses primarily on the policies. Cisco believes this focus will remove the potential for the controller to become the bottleneck of the network, supporting greater resiliency, availability and scalability, by pushing some of the intelligence out to the devices, using established networking protocols.

Basically, policies are defined within a logical, centralized policy repository in the controller, and the OpFlex protocol is used to communicate and enforce those policies within a set of distributed policy elements

on the switches/routers/etc. The protocol allows bidirectional communication of policy, events, statistics and fault information, so potential adjustments can be made to address changes in the environment.

To work, an agent must be embedded in the switches and routers to support the Cisco OpFlex protocol. As a result, Cisco is working on an open source OpFlex agent that can be used across platforms. Microsoft, IBM, F5, Citrix, Red Hat, Canonical, and AVI Networks have already committed to embedding this agent in their solutions.

IX. OPEN FLOW

An Open Flow Controller is a type of SDN Controller that uses the OpenFlow Protocol. An SDN Controller is the strategic point in software-defined network (SDN). An OpenFlow Controller uses the OpenFlow protocol to connect and configure the network devices (routers, switches, etc.) to determine the best path for application traffic. There are also other SDN protocols that a Controller can use such as OpFlex, Yang, and NetConf, to name a few.

SDN Controllers can simplify network management, handling all communications between applications and devices to effectively manage and modify network flows to meet changing needs. When the network control plane is implemented in software, rather than firmware, administrators can manage network traffic more dynamically and at a more granular level. An SDN Controller relays information to the switches/routers (via southbound APIs) and the applications and business logic (via northbound APIs).

In particular, OpenFlow Controllers create a central control point to oversee a variety of OpenFlow-enabled network components. The OpenFlow protocol is designed to increase flexibility by eliminating proprietary protocols from hardware vendors.

When choosing an SDN Controller, IT organizations should evaluate the OpenFlow functionality supported by the Controller, as well as the vendor roadmap. IT organizations should understand existing functionality and ensure newer versions of OpenFlow and optional features are supported (for example, IPv6 support is not part of the OpenFlow v1.0 Standard, however, it is part of the v1.3 Standard).

- *Sampling of OpenFlow Controllers include:*
- NOX: NOX is a Network Operating System that provides control and visibility into a network of OpenFlow switches. It supports concurrent applications written in Python and C++, and it includes a number of sample controller applications.
- Beacon: Beacon is an extensible Java-based OpenFlow Controller. It was built on an OSGI framework, allowing OpenFlow applications to be built on the platform to be started/stopped/refreshed/installed at run-time, without disconnecting switches.
- Trema: Originally named Helios, Trema is an extensible OpenFlow Controller built by NEC in the programming lanuages of Ruby and C, targeting researchers. It also provides a programmatic shell for performing integrated experiments.
- NEC ProgrammableFlow: Trema is the foundation for the Programmable Flow from NEC. ProgrammableFlow automates and simplifies network administration for better business agility, and provides a network-wide programmable interface to unify deployment and management of network services with the rest of IT infrastructure. Programmable Flow was the first to be certified by the Open Networking Foundation.
- Lumina SDN Controller: In December 2017, Lumina released the Lumina SDN Controller 7.1.0, which supports OpenDaylight Nitrogen (the seventh OpenDaylight platform). It also has support for OpenDayLight's Karaf 4, which allows users to choose the controller's protocols and services.
- BigSwitch: Its Big Cloud Fabric controller creates a virtual private cloud based on SDN controller abstractions and open networking hardware switches.

Netconf

Network Configuration Protocol, better known as NETCONF, gives access to the native capabilities of a device within a network, defining methods to manipulate its configuration database, retrieve operational data, and invoke specific operations. YANG provides the means to define the content carried via NETCONF, for both data and operations. Together, they help users build network management applications that meet the needs of network operators.

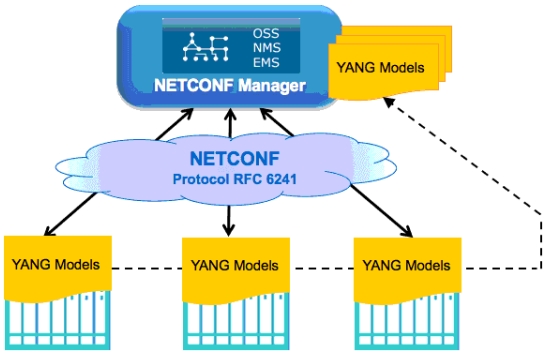

Figure 3: Network diagram of a NETCONF management system

The motivation behind NETCONF and YANG was, instead of individual devices with functionalities, to have a network management system that manages the network at the service level that includes:

- Standardized data model (YANG)
- Network-wide configuration transactions
- Validation and roll-back of configuration
- Centralized backup and restore configuration

Businesses have used SNMP for a long time, but it was being used more for reading device states than for configuring devices. NETCONF and YANG address the shortcomings of SNMP and add more functionalities in network management, such as:

- Configuration transactions: NETCONF configurations work based on atomic transactions consisting of multiple configuration commands required to move a network from state A to state B. The order of the configuration snippets within a transaction does not matter and the success of a transaction is based on the success of all the command snippets. If any single command fails, the entire transaction becomes a failure. So, there is no intermediate erroneous state, either it's at state A (if any one command of the transaction fails) or at state B (if the transaction is successful as a whole).
- Network-wide orchestrated activation: There is a distinction between the distribution of a configuration to all the networking devices and the activation of it. For example, if the operator wants to configure a VPN over a network of devices all at one time, NETCONF provides the flexibility to distribute the configuration, validate it, lock all device configurations, commit the configuration, and unlock. This set of actions will result in enabling a VPN over the entire network at the same time, in an orchestrated, synchronized way.
- Network-level validation and roll-back: Each NETCONF server keeps a "Candidate database" (in parallel to "Running config database"). Using this candidate data store, a NETCONF manager can implement a network-wide transaction by sending a configuration to the candidate of each device, validating the candidate, and if all participants are fine, telling them to commit the changes. If the results are not satisfactory, the manager can ask to roll-back all devices.

Save and restore configurations: NETCONF Manager can take a backup of the networking device configuration whenever needed and restore it by sending the saved configuration to any networking device.

Pof controller

As one of the initial implementations of SDN, OpenFlow [10] provides a powerful tool set for network operators to program and manage their networks adaptively [11]. However, its protocol-dependent nature still limits the programmability of the forwarding plane. Specifically, OpenFlow defines the matching fields in flow tables according to existing network protocols (e.g., Ethernet and IP). Therefore, OpenFlow switches need to understand the protocol headers to parse packets and perform flow matching, which may cause serious compatibility issues when new protocols try to add or remove header fields. Hence, it is desirable that the network programmability can be further enhanced such that the forwarding plane is protocol-independent and can be dynamically reprogrammed to support new protocol stacks seamlessly. Following this idea, recent studies have proposed a few new SDN technologies, such as protocol-oblivious forwarding (POF) [12] and programming protocol-independent packet processors (P4) [13]. The basic idea behind POF and P4 are similar, as they both try to decouple

network protocols from packet forwarding and make the forwarding plane reconfigurable, programmable, and future-proof. More specifically, POF introduces a protocol-independent instruction set, which allows a network operator to define the protocol stack and packet processing procedure in a much more flexible manner than that in the current OpenFlow specifications, while P4 designs a high-level language to program an SDN switch more flexibly for protocol innovations. As POF defines the southbound interface that can be treated as a promising enhanced version of Open Flow.

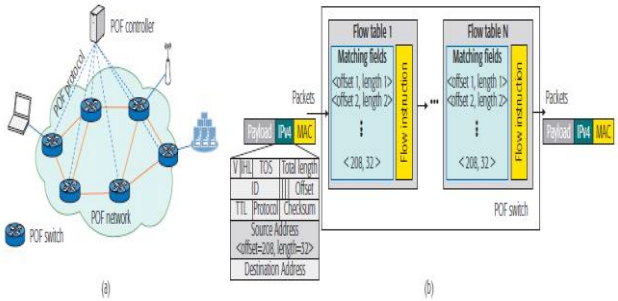

Figure 4. Overview of POF: a) architecture of a POF-based network; b) packet forwarding procedure in POF.

Figure 4 shows the overview of POF. The network architecture in Fig. 4a indicates that POF also uses a centralized controller to manage the packet forwarding in the switches. However, to make the forwarding plane protocol-independent, POF uses the forwarding procedure in Fig. 4b. Specifically, the packet parsing and flow matching in POF are based on a sequence of generic key assembly and table lookup instructions [12]. Hence, POF switches do not need to know the packet formats in advance. Actually, the search key of a matching field is defined as a tuple , where offset indicates the matching field's start bit location in a packet, and length tells the field's length in bits. For instance, in Fig. 4b, the IPv4 Source Address field in an IPv4-over-Ethernet frame is denoted as POF also includes a generic flow instruction set (POF-FIS) to facilitate POF switches to parse, edit, and forward packets arbitrarily [14]. It is known that in OpenFlow, the instructions and actions are also protocol-dependent; for example, actions like push, pop, and set are all subject to a specific packet field. While with POF-FIS, all the instructions and actions become protocol-independent. Hence, for packet forwarding, all a POF switch needs to do is to extract the matching fields from packets based on certain tuples , perform flow table lookups, and then execute the associated instructions defined in POF-FIS. On top of these innovations, POF defines four types of flow tables to enhance the programmability of the forwarding plane, which are the maskedmatch (MM) table, the longest-prefix-match (LPM) table, the extract-match (EM) table, and the direct table (DT). These types of tables occupy different memory sizes and can be searched with specific table lookup algorithms. Note that a flow entry in all the tables consists of both matching field(s) and related instruction(s), except for DT, whose flow entries only include instructions. By leveraging these tables, the forwarding procedure in a POF switch can be abstracted as a data path pipeline. To handle the situation in which the switch needs to store the flow information temporarily, POF defines a metadata memory for each switch and introduces several metadata-related instructions.

Forces controller

The ForCES provides a framework and defines open API/protocols that clearly separate control and forwarding planes. Although many such API/protocols has been developed and/or proposed (.e.g. OpenFlow and RestAPI), the real strength of ForCES lies with its model which enables the description of new data-path functionality without changing the protocol between the control and forwarding plane (Haleplidis et al, 2015) [15]. In contrast, OpenFlow requires implementation of defined protocol at controller and Forwarding plane (e.g. OpenFlow agent at networking device). ForCES extend the notion of elastic routing architecture with belief that data plane packet processing may need additional functionalities programmed in the future while the switches are deployed in the network. This abstraction innately differentiate it from other existing SDN protocols/APIs such as OpenFlow.

In ForCES framework, the elastic routing architecture or network element (NE) comprises of two parts FE (Forwarding Element) and CE (Control Element). Both of these elements are implemented through ForCES Agent which includes a set of protocols and models.

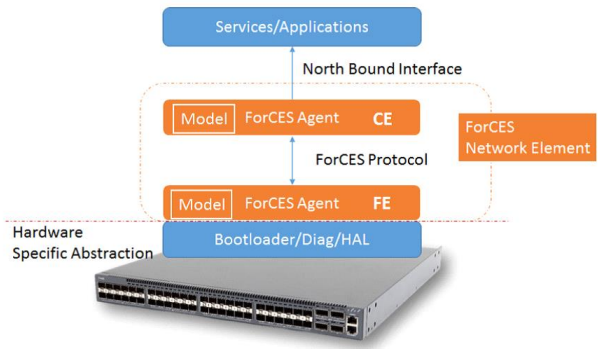

Figure 5: ForCES Framework

The ForCES framework allows developers to define their own FE abstraction models for implementation. This does not constrained CE to control and manage ForCES-modeled FEs. More importantly, ForCES framework is also protocol agnostic meaning a vendor may choose to use a traditional networking protocol to communicate between CEs and FEs. In theory, CE can implement routing protocols like OSPF and BGP and packets that cannot be handled by FEs are redirected to CEs (Haleplidis et al, 2015) [15].

X. CONCLUSION

Software Defined Network reforms the existing network design by introducing the control in a centralized way. Bringing routing control functionality at the centralized location relax the forwarding devices working. All the intelligence of SDN comes from the controller that acts like the brain of the system. The capability of the controller can be defined by the number of requests it can handle from the switches. Various modules inside the controller take care of network discovery, path discovery, flow pushing functionality The centralized controller provides simplified architecture, efficient handling of request messages but it fails to address the scalability issue. On the other hand, Distributed controllers perform well at scalability issue and give maximum throughput with high availability but they require proper message exchange procedure in the cluster. Both categories contain controllers from open source as well as dedicated vendors. Open source controllers like ONOS, Beacon, OpenDaylight provide rich community support to go through the SDN concept in brief. We can also categorize the controllers based on their uses in industry and academia. Selection of controller depends upon the various criteria like a single thread, Multithread etc. The choice of controller for academia can differ from industry.

FUTURE SCOPE

The authors intend to include in future, the study of and working on various protocols (other than overflow protocol) used in SDN and to compare some more SDN controllers such as ONOS, FloodLight, RYU and other SDN controllers. Also, several other topologies for controllers under SDN environment having more complex scenarios are required to be explored and the results have to be evaluated for some more parameter such as throughput and latency which are too suitable performance indicators to measure performance of a SDN.

REFERENCES

[1]. McCauley, M. (2012). POX, http://www.noxrepo.org/

[2]. Nippon Telegraph and Telephone Corporation, RYU network operating system, 2012, http://osrg.github.com/ryu

[3]. OpenDaylight, Linux Foundation Collaborative Project, 2013,http://www.opendaylight.org\

[4]. Gude al, N. (2008). NOX: Towards an operating system networks. ACM SIGCOMM Computer Communication Review,vol. 38, no. 3, pp. 105–110.

[5]. Smith M. (2014). OpFlex control protocol, Internet Engineering Task Force, http://tools.ietf.org/html/draft-smith-opflex-00

[6]. McKeown , "OpenFlow: Enabling innovation in campus networks," ACM SIGCOMM Computer Communication Review, vol. 38, no. 2, p. 69–74, 2008.

[7]. Enns, R., Bjorklund, M., Schoenwaelder, J., Bierman, A. (2011). Network configuration protocol (NETCONF). Internet Engineering Task Forc,http://www.ietf.org/rfc/rfc6241.txt

[8]. Song, H. ,Protocol-oblivious forwarding: Unleash the power of SDN through a futur proof forwarding plane.Proceedings of ACM SIGCOMM Workshop Hot Topics Software Defined Netw II. p. 127–132, 2013.

[9]. Doria et al, Forwarding and control element separation(ForCES) protocol specification. Internet Engineering Task Force. (2010), http://www.ietf.org/r/fc/rfc5810.txt.

[10]. N. McKeown et al., "OpenFlow: Enabling Innovation in Campus Networks," Comp. Commun. Rev., vol. 38, Feb. 2008, pp. 69–74.

[11]. N. Xue et al., "Demonstration of OpenFlow-Controlled Network Orchestration for Adaptive SVC Video Manycast," IEEE Trans. Multimedia, vol. 17, Sept. 2015, pp. 1617–29.

[12]. H. Song, "Protocol-Oblivious Forwarding: Unleash the Power of SDN through a Future-Proof Forwarding Plane," Proc. ACM HotSDN 2013, Aug. 2013, pp. 127–32.

[13]. P. Bosshart et al., "P4: Programming Protocol-independent Packet Processors," Comp. Commun. Rev., vol. 44, July 2014, pp. 87–95.

[14]. J. Yu et al., "Forwarding Programming in Protocol-Oblivious Instruction Set," Proc. ICNP 2014, Oct. 2014, pp. 577–82.

[15]. Haleplidis, E., Salim, J.H., Halpern, J.M., Hares, S., Pentikousis, K., Ogawa, K., Wang, W., Denazis, S. & Koufopavlou, O., 2015. Network Programmability With ForCES. IEEE COMMUNICATION SURVEYS & TUTORIALS, VOL. 17, NO. 3, THIRD QUARTER 2015.

Dr. Girish Ashok Kulkarni, et. al. "Study of Controllers in SDN." *IOSR Journal of Engineering (IOSRJEN)*, 10(7), 2020, pp. 06-15.