IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 25, 2018, accepted May 29, 2018, date of publication June 11, 2018, date of current version July 19, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2846236

Controllers in SDN: A Review Report

MANISH PALIWAL, DEEPTI SHRIMANKAR, AND OMPRAKASH TEMBHURNE
Department of Computer Science and Engineering, Visvesvaraya National Institute of Technology, Nagpur 440010, India
Corresponding author: Manish Paliwal (paliwalmanish1@gmail.com)

ABSTRACT Software-defined networking (SDN) is a networking scenario which changes the traditional network architecture by bringing all control functionalities to a single location and making centralized decisions. Controllers are the brain of SDN architecture, which perform the control decision tasks while routing the packets. Centralized decision capability for routing enhances the network performance. Through this paper, we presented a review report on various available SDN controllers. Along with the SDN introduction, we discuss the prior work in the field. The review states how the centralized decision capability of the controller changes the network architecture with network flexibility and programmability. We also discuss the two categories of the controller along with some popular available controller. For each controller, we discuss the architectural overview, design aspects, and so on. We also evaluate the performance characteristics by using various metrics, such as throughput, response time, 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.

INDEX TERMS OpenFlow, software defined networks (SDN), topology abstraction, pending raw-packet threshold (PRT), model-driven service abstraction layer (MD-SAL).

I. INTRODUCTION

The networking scenario in today's world is growing rapidly. Day by day the IP network is getting large and complicated. However, the large IP network allows strong connectivity among the users all over the world but at the same time presents the challenges of management of networks. When we talk about the functionality of the IP network, router plays an important role in it. The routing algorithms running inside the router perform the necessary functioning to route the network packets towards appropriate destinations. The advantage of router functioning comes from the fact that how efficiently it forward the packets. The routing algorithms inside the router constitute the control plane of the router, which sometimes called the brain of the router. The forwarding devices (Switch, Routers etc.) well connected to each other by the physical media, constitute the data plane of the network [1].

The current IP/MPLS networks are designed in such a way that the control plane and data plane are tightly bundled together which make the overall architecture complicated. This design also leads to the problem of up gradation in control logic to meet the future demands of users. It also makes the network management very hard. Specifically, if we talk about the routing algorithms of architecture, they are completely organized in a distributed fashion. Each router makes an independent routing decision irrespective of others for building a path selection decision. The tight integration of control and data plane makes the management of the system hard. Often the network administrator has to deal with the network through vendor-specific low-level commands to perform certain changes. On the other side, the complicated routing algorithm makes the overall cost of the system high. The cost represents not only the purchasing cost of the system but also the operational cost and management cost [2].

Software Defined Networking (SDN) is a new emerging paradigm which allows breaking of integration of planes. The concept of SDN was first introduced in June 2009 at Stanford University, US which was the result of work around OpenFlow technology started in 2008. The control logic of routing devices placed in a centralized controller with the hope that we can have a single point of control. In contrast to this, data plane consists of routers which are connected together in some topological fashion and perform the forwarding of data packets based on the decision taken by the central controller. Controller and the Data plane communicate through the protocol (specifically Open Flow [3]). With the introduction of the OpenFlow, the major advantages come in the fact of simpler modification in the network, global knowledge of network architecture. SDN allows simple high-level policies to modify the network as the device level dependency

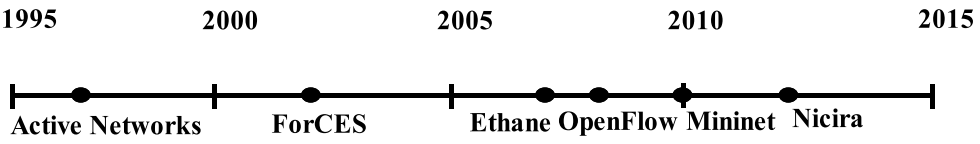

FIGURE 1. Prerunners of SDN at timescale.

is eliminated to some extent. Now the network administrator can operate the different vendor-specific devices from a single software console. The controller is designed in such a way that it can view the whole network globally. This controller design helps a lot to introduce a new functionality or program as it just needs to be placed in the centralized controller [3].

From the start of the SDN, many industrial communities worked around the Open source standardization of the technique and they came with some solution like OpenDaylight, OpenStack etc. In March 2011 IT & Networking giants like Cisco, Facebook, Google, Verizon, and Microsoft etc. collaborated with each other to form a working group on the widespread and open source adoption of the SDN architecture. The working group called as Open Networking Foundation (ONF). Each member of ONF is responsible for a specific activity for the promotion of SDN. For example, Architecture & Framework group deals with the architectural aspects of the SDN and defining the various components of it. Configuration & Management group deals with Operation, Administration, and Management of OpenFlow Protocols [3].

There are several review articles available in the literature which address design issues and key challenges in the different field of SDN. Sood *et al.* [4] reviewed for challenges and opportunities Software-defined wireless networking in IoT. Kobo *et al.* [5] addressed the challenges and design requirement for SD-WSN. Review work on various DDoS attacks on SDN controller carried out by Zubaydi *et al.* [6]. Neghabi *et al.* [7] stated various load balancing approaches in SDN through their article. Kreutz *et al.* [8] presented a comprehensive survey on overall design aspects along with challenges and opportunities in SDN. While working with SDN architecture, it is a major point of concern that which controller should be selected for deployment. Every controller has its own pros and cons along with its working domain. This paper is motivated by this need of controller selection. This article is different in a certain way as instead of briefly discussing and describing the overall architecture and design aspects of SDN it specifically focuses on the control plane aspects. It brings the all controller design aspects at a single point so that network administrator can select right controller as per the requirements of his SDN.

We organized our survey into 6 different sections. In section 1 Introduction, we start with the formal introduction of the technology along with the major areas where this technology has major effects. Section 2 discusses the historical background of the SDN. Section 3 introduces the basic building blocks of SDN architecture. Section 4 describes the main idea of the article by presenting the different controller strategies along with their suitable classification. The performance analysis is carried out in section 5. Finally, we conclude our review in section 6 by giving a summary of the article.

II. PRERUNNERS OF SDN

The idea of breaking the control and data plane is not introduced the first time rather it is the result of several efforts made of separation of planes like Network Control Point [9], ForCES [10], Ethane [11], Active Networking [1] etc. Fig. 1 illustrates the evolution of different technologies that lead to the development of SDN at the time scale starting from 1995 to 2015. Active Networking was the first attempt in this direction as they suggested that network element should have the capability to perform computation and modification of packet. Programmable Switch and capsules are the two distinct approaches suggested by the active networks. However, it doesn't provide a clear picture of separation. Soon NCP came into light which defined clean separation image. It was initially meant for telephone network and introduced by AT&T networks. The idea suggested by NCP leads to several innovations in the field.

ForCES was another major effort introduced in 2003. ForCES separated the control logic of individual data plane devices and made them available at the centralized location. However, this centralization was not complete one which we supposed to have because each control element interacted with their corresponding data plane element. So we can call it as partial centralization. In ForCES architecture, Forwarding Element typically implemented in hardware and responsible for filtering and forwarding approaches. On the other hand, Control Element works with the coordination between individual devices in the network for communicating the forwarding and Routing Information of all devices. Fig. 2 illustrates

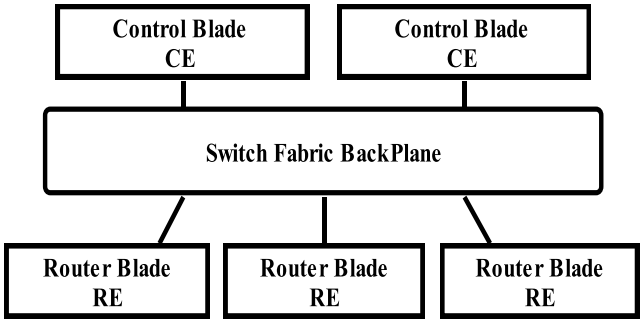

FIGURE 2. ForCES architecture.

the architecture of ForCES. The Router Blade of the underlying forwarding devices interacts with the corresponding Control Blade module through the Switch Fabric Backplane.

Ethane [11] was also one of initiative in this direction. It was introduced in a paper entitled as Rethinking Enterprise Network Control in 2007. Some of the features supported by Ethane were binding between packets and their origin, centralization of registration and topology management. In Ethane architecture we call controller as Ethane Controller. Fig. 3 illustrates the Ethane architecture. In the architecture, all the network elements are directly connected to the Ethane Controller. The Ethane controller links to other modules like Registration, Policies, and Binding etc. Registration module makes an enrollment for the new network element. Policies define the filtering criteria upon which the network traffic should be monitored and forwarded. Network Topology module prepares an annotated graph of network elements along with the link characteristics. The module can range from simple to complex which carries the additional network information.

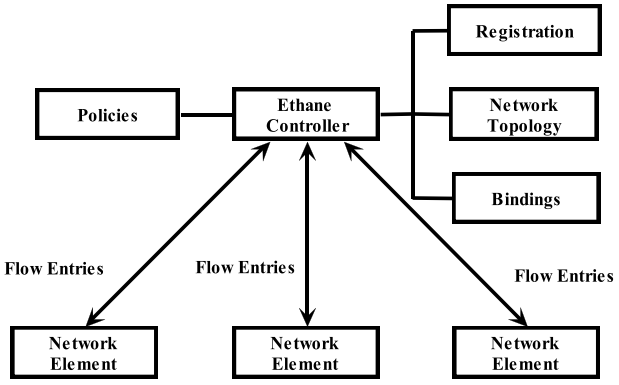

FIGURE 3. Ethane architecture.

OpenFlow [3] provides a set of protocols for effective communication between the control plane and data plane. It is supported by open source community. It first made its appearance in 2008 by Martin Casado at Stanford University. The appearance of OpenFlow was one of the main factors which gave birth to Software Defined Networking. Open Flow provides an open source platform for Research Community. It consists of a rich set of protocol specifications at the controller and switching element side.

III. ARCHITECTURAL ELEMENTS OF SDN

We divide the elements of SDN based on their corresponding plane. Fig. 4 gives a brief idea about the SDN element distribution. This section introduces each element along with their importance in SDN architecture.

A. INFRASTRUCTURE

The infrastructures at data plane consist of networking element i.e. routers, switches etc. which form the data plane. These devices perform the data forwarding based on the

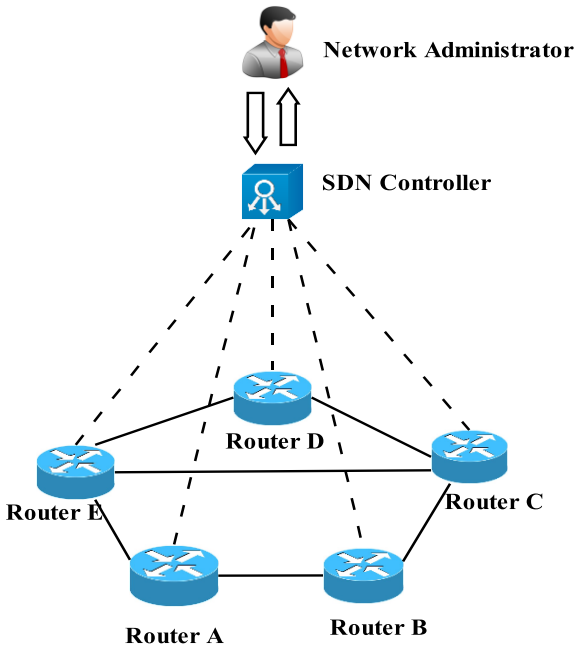

FIGURE 4. Architectural overview of SDN.

control decision. The control logic is the driving force behind these devices. The difference lies in the fact that the control logic is not distributed among these devices rather it is available at the centralized server. In today's scenario, we have a wide range of networking devices with OpenFlow supports like Open Vswitch [2], [12] from open source community, ofsoftswitch13 [13] from Ericsson, pica8 3920 [14] from pica8, contrail-vrouter [15] from juniper networks, RackSwitch [16] from IBM etc. On the other hand, Control plane consists of the server or cluster of the server (Distributed SDN) which performs the task of the controller. The controller usually termed as network brain because all the logic functionality implemented in it. The controller performs various tasks like building flow entries inside the routing devices, keeping track of packet information (flow statistics) etc.

The flow entries stored in a table (called as Flow table) can be considered similar to the Routing table in traditional networking architecture. Flow table consists of three portions matching rules, Action and Counter. Matching rules are the set of different fields of TCP/IP header portions like MAC address, Source IP, Destination IP, VLAN ID etc. A specific value of these fields for a packet constitutes a flow. Second field Action comprises of operation which has to be performed on the packet for processing. A typical action can be anything e.g. forward packet to the specified port, drop the packet, send to normal processing pipeline etc. The default action for each packet is forward to the controller. It means if the router doesn't contain the entry for the particular flow then it passes the packet towards controller for further processing. Once this packet gets process by the controller, it gets sent back to that router along with the flow entries. Now the router

processes the packet according to that particular flow entry. The last field Statistic stores various counting information like how many packets get pass from the specific port, how many packets for a destination address get process etc. This information can be grouped based on per flow, per table, per port basis [13], [15].

B. SOUTHBOUND INTERFACE

The southbound interface provides a means of communication between the controller and switching devices. It installs the appropriate flow rules in the switch forwarding table decided by the controller. OpenFlow is the most widely deployed southbound standard from open source community. OpenFlow provides various information for the Controller. It generates the event-based messages in case of port or link changes. The protocol also generates a flow based statistic for the forwarding device and passes it to the controller. Packet IN message sends in the case when a router doesn't know how to handle the new incoming packet. However, this is not the only choice for the southbound interface. Other interfaces include Open vSwitch Database (OVSDB) [17], ForCES [10], OpFlex [18] etc.

OVSDB is a southbound API designed to provide additional management capabilities like networking functions. With OVSDB we can create the virtual switch instances, set the interfaces and connect them to the switches. We can also provide the Quality of Service (QoS) policy for the interfaces. OpFlex southbound API contrasts to the OpenFlow API to some extent. OpFlex allows forwarding devices to deal with some of the management functionality. Initially, it abstracts the policies from the underlined plane and decides which functionality to be placed where.

C. NETWORK OPERATING SYSTEM

Network Operating System is the brain of the SDN, implemented in the controller. The NOS should provide the basic functionalities similar to a basic operating system. These functionalities include execution of the program, management of input/output operation, provide security and protection mechanism etc. Beside this, a network operating system provides networking functionalities like topology related function, shortest path forwarding etc. In contrast to the traditional IP network, the NOS in SDN is implemented in a logically centralized way. Based on the architecture and design aspects we can define two categories of NOS which are centralized and distributed. Some of the controller available in SDN domain are Onix [19], PANE [20], Fleet [21], ONOS [22], Meridian [23] etc.

D. NORTHBOUND INTERFACE

The northbound interface provides connectivity between the controller and the network applications running in management plane. As we already discussed that southbound interface has OpenFlow as open source protocol, northbound lacks such type of protocol standards. However with the advancement of technology now we have a wide range of northbound API support like ad-hoc API's, RESTful APIs etc. Soon we hope for the common standard for northbound API as SDN is growing day by day. The selection of northbound interface depends on the programming language used in application development. NOSIX [24] was the first approach towards the northbound interface implementation which was independent of programming language and controller aspects. The emergence of common northbound API is a critical task as the requirement of each networking application can vary. For example, a security application can have requirement different from the routing applications. The Northbound interface working group from ONF community are working already on the common standardization of northbound API.

E. NETWORKING APPLICATIONS

Network applications available at management layer are responsible for implementing the control logic, which provides an appropriate command to be installed in the data plane. The network applications are broadly divided into five categories, which are Traffic Engineering, mobility and wireless, measurement and monitoring, security and dependability and data center networking. Various types of networking applications can be implemented at management level e.g. load balancing, traffic optimization, QoS Enforcement, Predict application workloads, Fine-grained access control etc [20].

IV. CONTROLLER CATEGORIES

Software Defined networking makes use of two types of controllers which are Centralized and Distributed. Fig 5 describes the classification of the various controllers into two categories.

A. 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.

1) BEACON

Beacon [25] is one of the most popular open source centralized controller in SDN. It was designed on Java platform by Stanford University and Big Switch Network in 2010. We are not claiming that it was first and only available open source controller because NOX [26] was originally introduced as open source in 2008. However, initially, it was available for the single threaded environment. But soon it came up with multithreading solution named NOX-MT [27] in 2011. Beacon got so much popularity because of its importance in the research community. There are three main objectives of Beacon development - provide developer side productivity, achieve high performance and increase runtime capability to stop and start application. From programming

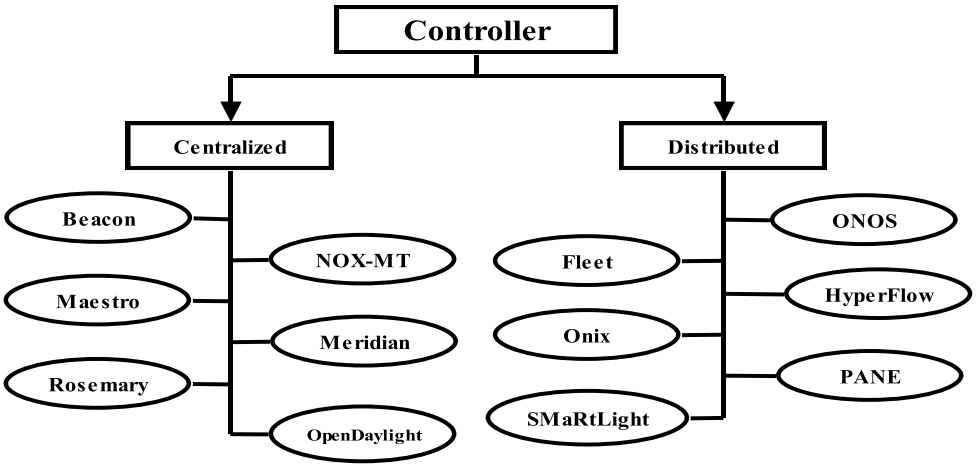

FIGURE 5. Controller classification.

language perspective C#, Java and Python were selected for beacon implementation. Lack of official support for different operating system platform eliminated C# as a choice for implementation. Similarly, python was eliminated because of the lack of true multi-threading support. So they only left with java which provides effective memory management and proper segmentation fault and memory leaks handling.

From developer productivity point of view, Beacon provides a rich set of libraries for application development. For example, for routing purpose, they provide IRoutingEngine interface which is helpful for designing different routing modules. Shortest path routing [28] is one of the well-known examples of routing module which can be implemented through this interface. For network topology, it provides ITopology interface which contains a set of operation to retrieve information related to link discovery, link registration/deregistration. To get the benefit of OpenFlow protocol, it uses OpenFlowJ API which is the Java-based implementation of OpenFlow 1.0 specification. So beacon can interact with the OpenFlow switches through IBeaconProvider interface. Code reusability is an important property of object-oriented languages. Beacon achieves this through a library called Spring. Through this, it can create multiple instances of an object and bind them together into a single entity. Beacon also provides other facilities like Device Management, web application development etc. through various available interfaces.

To deal with the runtime modularity feature, beacon provides Open Services Gateway initiative (OSGi) specifications [29]. Like OSGi specification, Equinox provides facility to create a new instance of the application at runtime. Now not only you can start and stop existing application at runtime but also create a completely new application. One more specification is OSGi service registry which allows new services to register themselves so that user can pick any service from the pool based on their requirement.

Performance is a major challenge in any networking design. From controller perspective, we define the performance based on two parameters - time required to process a single request and the number of input packet request it can handle in per unit time. Proper event handling can lead to effective performance in any architecture. In Beacon, they have provided a pipelining of messages. A shared queue is implemented which contains the packets from all switching elements. Any worker threads in the controller can pick available message request and execute it. For that purpose, IOFMessageListener should register with IBeaconProvider [25].

2) ROSEMARY

Sometimes it happens that network application interferes with the controller program and allows it to malfunction. Rosemary [30] offers the feature of controller resiliency in which third-party applications can perform interference. It is possible that they perform maliciously themselves but cannot harm the controller functioning. In Rosemary, the NOS is designed in such a way that it provides some sort of security corresponding to OpenFlow applications. It also makes a sandbox like structure around each application. The NOS architecture specially designed as Micro-NOS architecture [31]. The three design pillars of this architecture are-

i) Schedule each network application separately in different address space other than the controller.

ii) Implement a resource monitoring system so that we can track the resource consumption pattern of each application to find out its behavior.

iii) Introduce a permission structure for each micro NOS instance so that we can allow constraints for each instance regarding libraries, resources etc.

During the design of Rosemary, two important issues were robustness and security. Researchers found that network application doesn't support the robustness property because

they work in the same privilege zone in which network OS applications are running. Also, it lacks resource management facility as each application can demand any number of resources for task executions. It was monolithic type architecture. These two factors allow enforcement of separate process context for network applications. From a security point of view, they found that network applications don't need any authentication procedure for initialization of task execution. Access control mechanisms did not adapt properly so resource access could be done in an unsolicited way. These two factors allow NOS to incorporate security paradigm in the design architecture [31].

Based on these issues the design, of rosemary separates the network application's context from the controller context. They made a compartment for each shared module of NOS applications. Each time when an application needs resources it first contacts the NOS kernel. NOS kernel performs some scheduling activity like fair share scheduling to grant the resources to the application [30].

While performing the tasks, NOS should ensure that it should create a clear view of network application e.g. shared resources, functions, libraries etc. The abstraction should be minimal so that it can achieve maximum performance. The design also ensures that proper balancing should be there between robustness and performance. Sometimes it happens that enforcement of too many constraints leads to latency overhead and the system cannot accept a large number of requests. The NOS takes care of this by implementing the light weighted architecture.

3) MAESTRO

The main factor for the design of any system is high performance with parallelism. Consider the case of Datacenter network design [32]. Datacenter network should be capable of accepting multiple requests at the same time. It should have proper scheduling strategies [33] to schedule the requests from multiple users. Design should also be inspired by the fault tolerance [34] issues so that it can continue work even in the case of partial failure.

The same issues are important while designing any controller. Maestro [35] deals with these issues. It is a java based multithreaded controller from Rice University. It explores additional throughput optimization technique to achieve maximum performance with exploiting parallelism. If we observe the basic working of any controller then we find that first packet is sent to the controller each time. After performing a security check, the controller carries path calculation and push appropriate flow entries in the data plane. This scenario performs well when a number of packet request are small. But in case of datacenter like a scenario where around 10 million requests arrive per second, it is not a good choice.

To deal with this, Maestro introduces the concept of batching. Multiple requests are grouped in a single batch from users. Once a thread is free it can pick any available pending request from the batch and start executing it. The availability of threads depends upon the number of cores. Maestro allows multiple flow requests execution by different worker threads. Through this, we can achieve the parallelism which is first design issue in maestro design. There are some design considerations in threads. First, how we can divide the available requests among the threads? One solution is to distribute the requests fairly among the available cores/threads by introducing a dedicated task queue for each thread. However, there can be a drawback of this as the idle thread cannot take care of requests for other threads. Also, demand for each request can be different with varying number of CPU cycles. The second issue in design is core binding. Many times it happens that when we move the actively running code from one CPU to another CPU, the complete processing fails if both cores don't share a common cache. In such case, manual synchronization is needed of core states and cache to continue execution in a new environment. It is called core binding which is an overhead for the system. Alternate solution for this is to perform all execution of one task at single core irrespective of execution time. Thread binding is also a solution in which the thread first checks its own dedicated task queue for execution. Once it is empty then it accesses shared queue to pick any available pending request. Each queue should specify some threshold while growing too large in size. In Maestro there is provision for task priority. Different tasks have different priorities e.g. output stage has the highest priority and input stage has low priority process. The reason for assigning lowest priority to input stage is that they are shared in the raw packet task queue. Tasks, which are in flow process stage, possess medium priority.

Maestro introduces threshold to raw packet task queue which is called Pending Raw-Packet Threshold (PRT) [35]. PRT has importance in deciding the pending task quantity in the queue. When the tasks in the pending queue have more number of requests then the PRT, the incoming tasks are paused to maintain the queue size. Similarly, if the numbers of tasks are small then input requests are resumed. Now the question is how to decide the size of PRT. The PRT should have enough size so that raw packet task queue cannot be completely empty.

Maestro also provides a rich set of interfaces and libraries. Discovery is one of application, which continuously sends probing messages to switches, to find the status of the newly joined switch. When the discovery routine finds the return LLDP probing message, then it can decide that from where it is getting the message so that it can determine the topology of the system. IntradomainRoutingapplication allows update in routing table once the topology gets changed. Authentication application takes care of security check constructs. Once it passed then the RouteFlow application can determine the appropriate path for the request. How does one ensure that the RouteFlow application is using correct routing table during the update procedure? For this, maestro allows execution of path selection of current request through older routing table to maintain the consistency in the result. Once the update takes place it will apply for upcoming requests [35].

4) NOX-MT

Throughout the development of SDN architecture, the main concern was about the controller performance. Two main questions of interest are-

i) How fast the controller processes input data path request?

ii) What is the capacity of the controller i.e. how many requests it can handle efficiently?

SDN community got various suggestions from researchers about this. Kandula *et al.* [36] presented a cluster of about 1500 servers which faces 100k flow request per second. Benson *et al.* [37] denoted that a network having 100 switches can experience 10M flow requests per second in the worst case.

NOX [27] was originally introduced as single threaded open source control platform to study the performance characteristics of SDN architecture. Later a multithreaded version of NOX was introduced as NOX-MT [27] which uses I/O batching for optimization purpose. During the packet forwarding, NOX performs a mapping between a MAC-switch tuple and port number for each switch and store them into a hash table data structure. The read operation only needs to go through the hash table. A new source MAC address requires an update in the hash table. The numbers of new source MAC addresses events are restricted by the total number of hosts and switches in the network. They are directly proportional to the product of a number of hosts and number of switches.

The result obtained from NOX-MT shows improvement in the performance by a factor of 33 in comparison to NOX. Although, NOX-MT shows a significant improvement in performance but still did not address some of the difficulties of NOX e.g. heavy use of Dynamic memory allocation, redundancy in multiple copies for each request etc.

5) MERIDIAN

Meridian controller was originally designed for the applicability of SDN architecture in a cloud environment. The idea was to build a service level network that can support features like policy abstraction, high connectivity in the cloud. SDN fits perfectly to the cloud architecture either in Infrastructure as a Service (IaaS) or Platform as a Service (PaaS).

The Meridian cloud network architecture, which is SDN architecture for cloud networking, is mainly organized as three different layers. The first layer is Abstracted API layer responsible for exposing the required abstracted details for the network model. For example, Topology abstraction is one kind of functioning in which abstracted view of the topology of underlying network is presented to the networking applications. The level of abstraction can differ from application to application. Cloud orchestrator requires a detailed and complete topology information as it needs to decide that where the virtual machine should be placed. On the other hand, Control application requires topology along with different path information for controlling the networking devices. Second layer Network Orchestration layer collaborates with the abstraction layer to convert the logical command into their corresponding actions. Orchestration layer has an additional responsibility to generate the network services e.g. routing action, shortest path computation etc. It does proper coordination with other layers so that application requests can be mapped successfully to the underlying physical devices. Network Driver layer is the lowest layer of the model which serves as an interface between the controller and various networking tools. The layer consists of plugins or drivers which allow devices to work accordingly to the command issued [23].

Meridian was originally based on the design of Floodlight controller. Floodlight is a java based controller which supports various network service e.g. path discovery, link level information discovery etc. Certain modules are added in the design of Floodlight to achieve then required goal of Meridian controller. Meridian service model organizes the whole system in five different parts. First, endpoints are the basic network entities. For a Virtual machine, we can consider it as a virtual interface. Endpoints are combined together to form a group which possesses some common properties. Services are provided as the functionalities e.g. new routing module, traffic filtering module etc. Two or more groups can be combined together along with required services and termed as a segment. Formally it is defined as three tuples set $\{g1, g2, ser\}$ where g1 and g2 are the two groups and the ser is the required services. Finally, all these subnets linked together to form a virtual network. In virtual networks different services can be placed on the different subnets and a connectivity link is provided between them. A typical example of all these elements is Web services model. In this model, servers are placed at tier 1 subnets which are called web server. Tier 2 subnet contains the application which interacts with the web server to use the services. Finally, there is application subnet link to the database subnet which is presented on tier 3 [23].

6) OPENDAYLIGHT

The opendaylight project started with the concept of model-driven software engineering (MDSE) approach. Its architecture is inspired by the Beacon and makes use of Open Service Gateway Interface (OSGi). MDSE consists of a framework which defines the models and relationships among them. The different models communicate with each other by data modelling language. These models are platform independent to support the different business policy needs. NETCONF and RESTCONF are used as a model-driven network management protocol. The basic operations supported by NETCONF are Create, Retrieve, Update and Delete. Besides this, it also supports Remote Procedure Call (RPC) operation. The data encoding technique used in NETCONF is based on XML to support data configuration and operation. Another configuration protocol is RESTCONF which is similar in some aspect to the typical REST-like protocol. This protocol is responsible for providing programmatic interface over the HTTP [38].

YANG is used as a modelling language so that models can communicate with each other. Initially, YANG was used to configure the models but later it was used to describe the other network constructs i.e. services, policies, protocols etc.

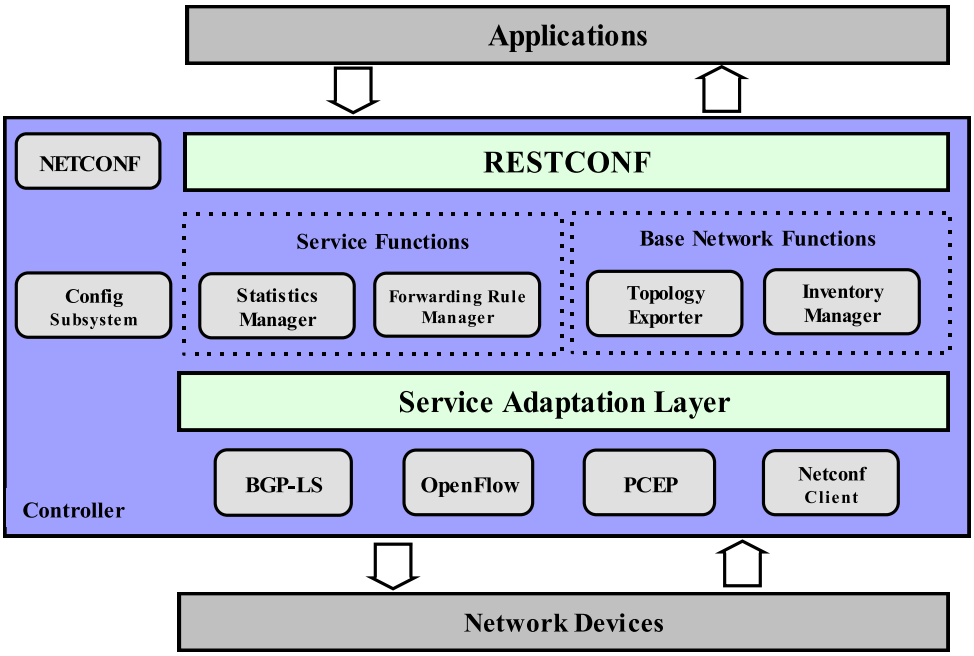

FIGURE 6. OpenDaylight architecture (network view).

The data structure used by the YANG is a tree structure. The internal structure of tree can be further complex i.e. lists and unions [39].

While developing the OpenDaylight controller certain considerations are taken. The controller should be flexible. It should provide a common configuration platform for different application development. The system should support runtime modularity for the addition of models at runtime. The modularity should meet the performance and scalability feature [38].

Fig. 6 describes the network view of OpenDaylight controller architecture. The OpenDaylight architecture consists of a set of North Bound and South Bound plugins which are separated by the Service Adaption Layer (SAL). At the Southbound side, the plugins are Openflow, Netconf Client, PCEP etc. Similarly, North Bound plugins consist of Topology exporter, Forwarding Rule manager, Statistics Manager etc. To meet the objective of the OpenDaylight, the SAL is modified using the model-driven software engineering concept and termed as Model Driven Service Adaption Layer (MDSAL). Following are the certain points regarding the MDSAL.

i) An RPC is a call from a consumer to the provider which is processed either locally or remotely. The call connection is of one to one type.

ii) A Notification is a reply expected by the consumer from the provider side.

iii) The Data store is a tree-like logical structure described by the YANG schemas.

iv) A Path is the location of the specific leaf in the tree.

Fig. 7 describes the architecture of MDSAL. It consists of two different brokers for data handling. DOM Broker deals with the runtime activity of the architecture. Binding aware broker deals with the JAVA APIs for plugins. BA-BI Connector works as a mediator between the DOM broker and Binding aware broker. To implement the dynamic late binding, the BA-BI connector works along with Codec Registry and Codec Generator.

B. DISTRIBUTED CONTROLLER

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

1) HYPERFLOW

HyperFlow [40] is the first distributed control plane designed for OpenFlow. The original design of HyperFlow is inspired by the NOX [26]. The design is distributed because of physical availability of different controllers but they form a logically centralized environment. There are certain issues pointed out during the design e.g. When the switches increase in their quantity than the traffic increases towards the centralized controller. In such condition, centralized control becomes a bottleneck. To handle such scenario we need multiple controller replicas physically distributed over a geographical area. This large network size gives rise to long flow setup latency for switches. The processing power of individual controller is also a significant issue.

FlowVisor [41] has a similar design to HyperFlow but it allows resource slicing so that each slice takes care by corresponding controller instance. HyperFlow uses public/subscribe message system to send the event messages towards another controller. During message passing, it is

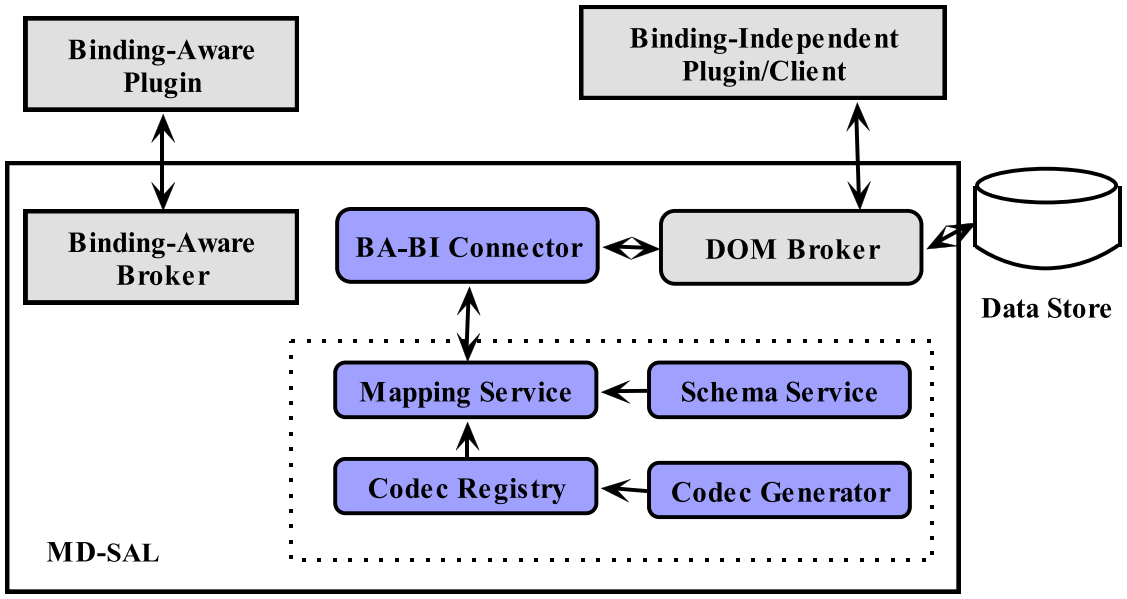

FIGURE 7. MD-SAL design.

necessary that we have persistent storage of events because there may be a chance of reordering of event list for a controller during network partitioning into slices. To adopt such feature HyperFlow makes use of WheelFS [42] which is a distributed file system for distributed applications. When the network gets partitioned WheelFS continues its functioning in each partition. Controllers in one partition don't receive any messages from other partition controllers.

In HyperFlow design, each controller can only program the switches which are directly controlled by it. While to control other it publishes a message which contains the source controller identifier, target switch identifier, and local command identifier. Each controller continues to send periodic messages to show its presence in the network. If any controller fails to send a message within three advertisement intervals then it is assumed to be failed. In this condition, the switches associated with failed controller needs to migrate to some other controller to continue operation. The control application should not depend on the temporal ordering of events until they belong to same switch or links because different controllers see different ordering of events. HyperFlow also supports the Authoritative controller to ensure the correct operation of each case [40].

2) SMARTLIGHT

SMaRtLight [43] is designed to address the fault tolerance issue in the network. Three aspects of failure are discussed in it which are a switch or link failures in the data plane, switch-controller connection failure in the control plane and failure of the controller. The controller is designed by extending the Floodlight [44] controller. Lease management application, One to one mapping between switches and data store connection, caching support for data store are major changes which are carried out in Floodlight for SMaRtLight design.

To achieve fault tolerance, we assign the job of overall network management to a single controller which is called primary controller while other controllers act as backups. In the case of failure of primary a smooth transition is performed to choose one of the back-ups as a new primary. The controller stores all the application related data in a shared data store which is implemented through Replicated State Machine (RSM) [45]. Shared data store keeps all the state related information to the application like Network Information Base (NIB). During the transaction from the fail controller, the new controller gets a complete update from the data store.

The design also keeps a cache for fast access to state information. Now we don't need to access data store again and again for a simple read operation. The cache is also free from synchronization aspects because at a time only one controller is using it.

The system is arranged in such a way that switches can directly connect to a controller but not to data stores. Controllers are available in between the data store and switches so they can communicate with both. Based on message passing paradigm we can say that process p is connected to process q if the request sends by p can be answered in a predefined time interval by process q [43]. System design gives three main facts to detect the failure of the component.

i) If a switch is connected to all correct controllers and it has not crashed then it is working correctly.

ii) If a controller is connected to all data servers and it has not crashed then it working correctly.

iii) If a data store is well connected to all correct data servers and running the recovery protocols then we can say that it is working correctly.

Initially, all the switches have controller's role as EQUAL. Once a Primary gets selected the primary replica changes its

status in all switches as MASTER. Eventually, this leads to change the role of other controllers as SLAVE in all switches. Few points to be noted while interacting the controller replicas as at any point of time there can be only one primary replica in the system. Once primary replica fails some correct controller replicas can claim to become the primary. Each replica must generate an acquireLease(id;L) message which should be sent to the data server. The parameter id defines the id of replica and L defines the lease time require. The reply message from data store contains the id of primary replica. If any controller receives id its own from data server then it becomes eligible to be MASTER otherwise, nothing will happen and lease time will be updated for the current primary. Data store is designed in such a way that it works on key-value store interface which supports a basic operation like put, get, remove, list, etc. It also supports a cache for fast access to state information [43].

3) ONOS

Open Network Operating System (ONOS) [46] is the Open source, Distributed Controller designed for SDN environment. It is mainly designed to address the scalability, availability, and performance issue. The major challenges identified are-

i) Achieve high throughput: about 1M requests per second.

ii) Latency should be in 10-100 ms range for event processing

iii) State size for network which is order of 1TB

iv) Achieve high availability 99.99% for services

Similar to HyperFlow it supports logically centralized but physically distributed controller architecture. Two prototype specifications are defined for the ONOS. Prototype 1 focuses on building network architecture which provides a global network view with fault tolerant and scalability features. It was originally based on open source single instance controller, Floodlight [44]. Prototype 2 focuses on improving the performance of overall system. To achieve this, they emphasized on a number of remote operation and time required to process them. This should be kept as small as possible.

To achieve high performance in the system, RAMCloud data store [47] is used which provides latency in the range 15-30 ms for read/write operation. Topology cache support is provided to reduce the time for most frequent read operations. Faster lookup can be possible through in memory topology view. The polling issue is addressed by implementing public-subscribe event notification and communication system based on Hazelcast [48]. Network View API is also simplified and contains major three areas - Topology abstraction, Path installation system, and events. Table 1 discusses the various characteristic aspects of the controller in centralized and distributed domain.

4) FLEET

Fleet [21] is one of the first controllers which addressed the malicious administrator problem. The idea is to prevent the controller from malfunctioning because of malicious administrator configuration. The administrator can damage the routing, forwarding, network ability of controller. It has been observed that human errors are responsible for 50% to 80% network outages [49]. The malicious network administrator can easily degrade the performance of the system by misconfiguring the controller.

The objective of Fleet design is to prevent the k malicious administrators among n administrators from further affecting routing, forwarding and availability in the system. It is assumed that the number of network administrators for a network is restricted to at most 10. Switches in the system are installed with authentication scheme so that they can verify controller. The non-malicious administrators are grouped and they follow a single routing policy among them. Besides this, all administrators have proper communication to each switch for message exchange. The probability of compromise, Protocol Overhead and Recovery time are the metrics considered for measurement. The first metric is used to find out the possibility that amongst the given controllers a group of k controllers will have different network configuration compared to non-malicious controllers [21]. Protocol overhead specifies computational overhead which is carried out by the system from the time when failure first introduced in the network and a fix is performed on the same. Total time duration denotes the third parameter recovery time.

The basic building block of Fleet architecture is Administrator Layer and Switch intelligence Layer [21]. These layers are arranged in logically centralized manner but physically they are distributed across various controller instances. Switch intelligence layer interacts with its corresponding switch and operates on each switch. Two versions of Fleet design are suggested by the researchers. They are- single configuration approach and multi-configuration approach. In single configuration approach, all the administrators agree upon a single threshold value for making a high-level routing decision which is installed in corresponding switches. On the other hand, the multi-configuration approach allows a set of n different routing configuration decisions from the different administrator and select anyone for particular switch flow based on metrics.

5) ONIX

SDN requires a common control platform which allows implementing various control functions like routing, access control, traffic engineering etc. ONIX [19] is introduced as distributed controller offering such a common control platform written in C++. There are certain challenges for designing common control platform.

The control platform should ensure that it provides various functionality for management application in various contexts. Scalability is the need for any network architecture so control paradigm should meet this requirement.

i) It should be reliable to handle failure in the system.

ii) It should provide a simplified structure for building management application.

TABLE 1. Characteristics comparison of controllers.

Controller	Architecture	Language	Prototype Basis	Flows/second	Northbound API	Modularity	Productivity	Partner	License
Beacon	Centralized Multithread	Java	Developer Side Productivity, High Performance	12.8M	Ad-hoc API	Fair	Fair	Stanford University	GPLv2
Fleet	Distributed	---------	Malicious Administration	---------	Ad-hoc API	Fair	Fair	Carnegie Mellon University	--------
Hyperflow	Distributed	C++	Scalability, Response Time	30K	---------	Fair	Fair	University of Toronto	--------
Onix	Distributed	C, Python	Common Control platform for Consistency and Durability	2.2M	---------	Fair	Fair	Nicira Networks, Google, NEC, ICSI	Commercial
NOX-MT	Centralized Multithread	C++	Throughput Optimization	1.8M	Ad-hoc API	Low	Fair	Nicira	GPLv3
ONOS	Distributed	Java	High Availability, low Latency	1M	RESTful API	High	Fair	At&T, Ciena,Cisco, Ericsson,Fujitsu, Huawei,Intel, Ntt Communication	--------
PANE	Distributed	----------	API for SDN Control, Security	----------	PANE API	Fair	Fair	Brown University	--------
Rosemary	Centralized	----------	Controller Resiliency	----------	Ad-hoc API	Fair	Fair	Atto Research, KAIST, CSL-SRI International	--------
SMaRtLight	Distributed	Java	Robust SDN design	367K	RESTful API	Fair	Fair	University of Lisbon	--------
Maestro	Centralized Multithread	Java	Parallelism	4.8M	Ad-hoc API	Fair	Fair	RICE, NSF	LGPLv2.1
Meridian	Centralized Multithread	Java	Service level model for application networking in Cloud	---------	Extensible API layer	High	Fair	IBM T. J. Watson Research Center	--------
OpenDaylight	Centralized	Java	Modular Controller Framework	106K	REST, RESTCONF	High	Fair	Linux Foundation With Memberships Covering Over 40 Companies, Such As Cisco, IBM, NEC	EPLv1.0

iii) Control plane functionality should not enforce additional burden on the overall functioning of the system to deal with performance and latency issues.

Onix allows its instance to be written in multiple languages which subsequently run in different processes. Currently, Onix supports C++, Python, and Java for implementation. Some of the management application built on the top of Onix instances are Multi-tenant virtualized data centers, Scale-out carrier-grade IP router, Distributed Virtual Switch etc. The NIB stores all the state information of the Switches and function supported by Onix API. Query, Create, destroy access attributes, notifications, synchronize, configuration and pull are certain functions supported by Onix API [19].

6) PANE

The idea of PANE [50] controller suggests that there should be a configuration API between user and control plane. In networking scenario, it happens that many times the required condition cannot be fulfilled instantly but we can reserve it for future. Configuration API does the same thing. It applies greater visibility and control over the network to make a required reservation. PANE deals with two problems-decomposition of control and visibility in the network, conflict resolution among the users and their requests. Decomposition of control and visibility can be resolved with the use of Privileges. Similarly, request conflict can be resolved by making use of conflict resolution operator and using the Hierarchical flow table.

Principals in PANE are end users or more specifically application running on their behalf. The principal can interact with three types of messages which are request, query and hints. The request message is used to take control over the resources e.g. bandwidth or access control. Query messages are used to gain information about the network states.

Hints messages indicate the future demand of systems or possible future behaviour of the system. The principal should be limited in their authority. For this, PANE introduces the concept of the share which is a combination of principal, privileges and flow group. A share indicates that which principal can issue which message for which flow. Based on the different shares it prepares a share tree. Share tree does not itself introduce any new policy in the system while it applies a constraint to the existing policies. Policies and share tree combined together to form a policy tree. It may be possible that two policies can conflict with each other over some criteria. To avoid such condition policy tree are well organized in Hierarchical Flow Tables [50].

The request in PANE is processed in step by step manner. First, a principal generates a request regarding requesting a resource or something else and passes it to the controller. It should be noted that only an authenticated principal can send a request message to the controller. PANE first checks for the integrity of the message and if the messages follow the specified criteria and are compatible with network state. If the request gets passed successfully than it gets added to the tree and controller installs appropriate policy in the network.

Conflict resolution in PANE takes place through the conflict resolution operator which is used in the Hierarchical Flow Table. For each of the two conflicting request three types of operators can be applied they are +D, +P and +S. The conflict request can have different types of relationship to each other like they can be siblings, they can be parent and child to each other or they can belong to the same share. Based on this relationship PANE applies the appropriate operator. +D operator is used to avoid conflict when both the requests belong to the same share. +P operator is used when the requests follow the parent-child relationship to each other. +S operator applies when both requests are siblings. Based on the operator PANE introduces very simple procedures as in the case of a parent-child relationship, child request overrides the parent request. Similarly, in the case of siblings, Deny request will be overridden by the Allow request. The +D and +S operator in the PANE have similar meaning while dealing with conflict resolution procedure [50].

Each request gets processed in either strict mode or Partial mode. In strict mode, it is necessary that the required condition should hold clearly for each packet. No relaxation is allowed in strict mode. For example, if an application is demanding for 50 Mbps bandwidth then it is necessary that result of HFT must allocate 50 Mbps bandwidth to it. On the other hand in partial mode, there can be a relaxation for the required condition. For a similar example, the required bandwidth 50 Mbps can be get relaxed with the 40 Mbps bandwidth. Each of these two modes has their own advantages and they purely depend on the network application behaviour. Another point of interest in PANE design is Network Information Base which stores the network element like switches, ports, queues etc. and their corresponding capabilities. NIB translates the logical action into their corresponding physical actions and holds the necessary information related to the switch characteristic vendor, version, and statistic details [50].

Just like other controllers, PANE also contains fault tolerance and Resilience procedure. Two types of failures can be possible in network one is the failure of networking elements i.e. links, ports, switches etc. Second is the failure of the controller itself. In the case of link failure or link modification, PANE controller recompiles the policy tree. As we know that the link gets updated so the outcome of the recompilation is not necessarily been available. If it is not available then in such scenario controller processes the request again and again to recreate a new policy tree and each of the principal gets informed regarding this. To handle the controller failure it stores the database instance into the log by periodically checking the database. If the controller gets a restart because of failure, the instance details copies from the log record so that the controller can continue its functioning like before [50].

V. PERFORMANCE ANALYSIS

This section discusses performance analysis among various controllers discussed in the previous section. Two metrics namely throughput and Response time are taken for analysis. Throughput defines the number of input requests which controller can handle per second. Access time defines latency which is a time period required by the controller to process the request.

Cbench [50] provides a platform to evaluate these parameters. The features offered by Cbench are a measurement of maximum and minimum response time for controller irrespective of a number of connected switches, throughput measurement in bounded environment i.e. bounded number of the packet on the fly, calculation of maximum throughput etc. It operates in two modes i.e. latency and throughput mode. In latency mode, each switch interacts with the single request at a time until the processing is over and subsequently proceeds for next request. Generally, low load condition is considered for this case. On the other hand throughput mode computes the maximum flow processed by the controller in unit time.

Controller's throughput performance is also evaluated in the Multithreaded and single threaded environment. For comparative analysis of controller, system configuration assumed to be having Intel Xeon E5-2870 processor, 32-64 RAM with Ubuntu 11.10 or higher VM. Fig. 8 defines responses of the controller in a single-threaded environment where Onix is leading with 2.2M requests per second. Amongst the discussed controllers Onix and Beacon are the only controllers which show throughput over 1M/s. Onix has double responses in the unit time period as compared to another controller like NOX and ONOS. Ryu [52], Hyperflow and POX [53] are amongst the controller having least response time. For application development, which requires higher throughput, Onix can be a preferred choice.

On the other hand, Fig. 9 shows the throughput performance in a multithreaded environment where Beacon again

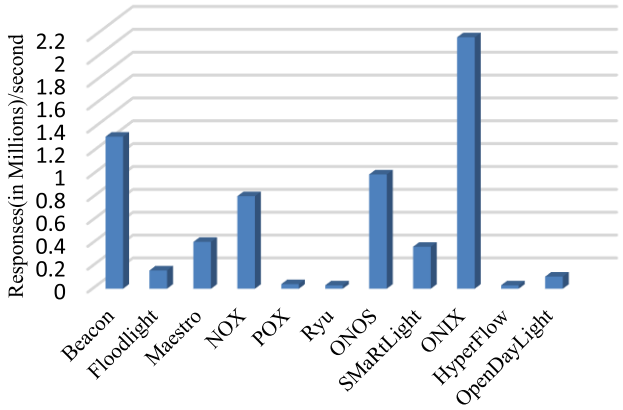

FIGURE 8. Throughput in single thread environment.

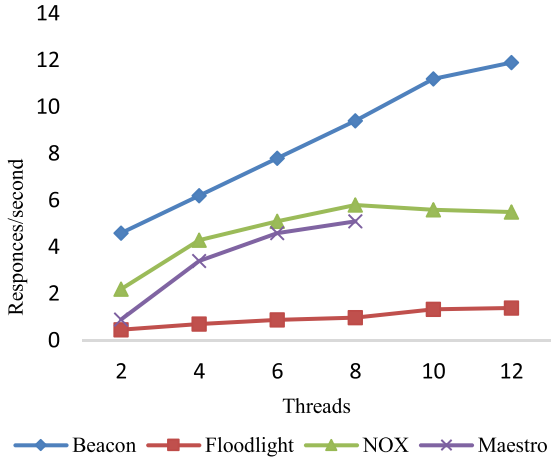

FIGURE 9. Throughput in multithread environment.

leads in the range 4.6 – 11.9M responses/second. From the graph, we can see steady growth in Beacon performance. Threads are taken from min 2 to max 12. In the graph Maestro shows the performance up to 8 threads only. However, its performance is much better than Floodlight which gives 1.39M responses/second when number of threads are 12. NOX have slightly high throughput performance compare to Maestro but comparatively very low to Beacon.

Fig. 10 shows the latency performance based on average response time for the single request. Onix has least response time (latency) amongst the controllers. POX controller shows the average response time in the interval of 100 to 200 which is slightly higher than other controllers. POX is the only controller which has response time over 100 μs. OpenDayLight, which shows latency higher than only Onix, is in the category of low latency controller. PANE, NOX, Maestro and Floodlight have latencies similar to each other.

We again took a set of controllers like Rosemary, NOX, Floodlight, and Beacon to evaluate the throughput. Fig. 11 shows that Rosemary is leading in the chart up to 10 threads and performs better than other controllers like Beacon etc. But when the number of threads goes beyond 10 then the throughput becomes stable and now beacon shows better performance over it. So it's clear from the figure that beacon

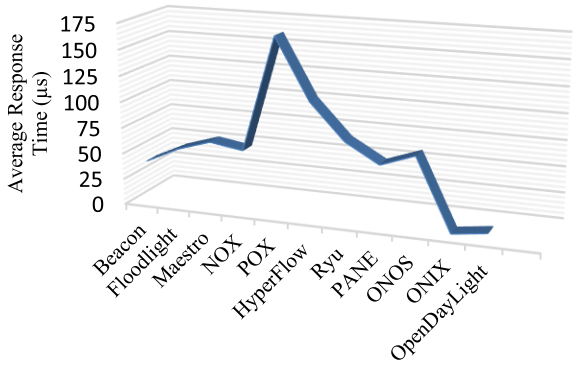

FIGURE 10. Latency in single thread environment.

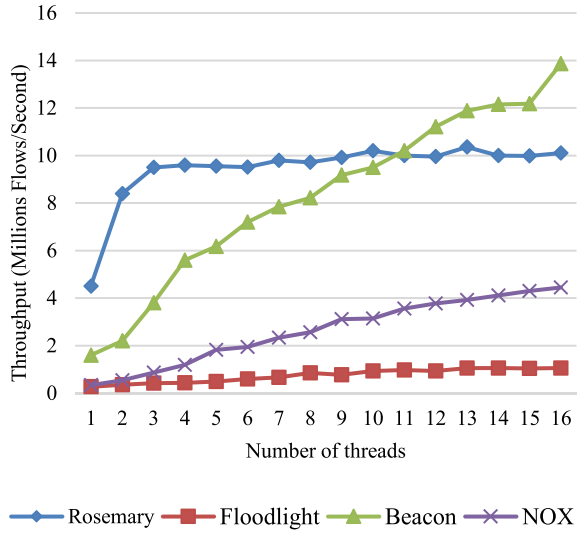

FIGURE 11. Throughput comparison of popular controller in multithread environment.

will be a preferred choice for a multithreaded environment with a high number of threads. On the other hand, Floodlight has throughput range in 0.27-1.06 M responses/second. Fig. 11 represents that initially floodlight and NOX behave similarly but when we gradually increase threads the difference of throughput among them becomes high. NOX shows throughput near about 4 times in comparison to Floodlight at 16 number of threads.

Based on the overall analysis Beacon shows throughput 5M/s-12.8M/s with having threads in the range from 2 to 12. NOX also have the near optimal throughput to Beacon from 5M/s-8M/s. Maestro lacks this performance by having throughput only 1M/s-5M/s. So in throughput scenario, Beacon performs well over NOX and Maestro. Similar case consider for latency calculation which shows a result that Beacon has 24.7 μs (25 μs), Maestro has 55 μs and NOX has 50 μs of latency.

VI. 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 etc.

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.

REFERENCES

[1] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden, "A survey of active network research," *IEEE Commun. Mag.*, vol. 35, no. 1, pp. 80–86, Jan. 1997.

[2] B. Pfaff, J. Pettit, K. Amidon, M. Casado, T. Koponen, and S. Shenker, "Extending networking into the virtualization layer," in *Proc. Workshop Hot Topics Netw.*, 2009, pp. 1–6.

[3] N. McKeown *et al.*, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Apr. 2008.

[4] K. Sood, S. Yu, and Y. Xiang, "Software-defined wireless networking opportunities and challenges for Internet-of-Things: A review," *IEEE Internet Things J.*, vol. 3, no. 4, pp. 453–463, Aug. 2016.

[5] H. Kobo, A. Abu-Mahfouz, and G. Hancke, "A survey on software-defined wireless sensor network: Challenges and design requirements," *IEEE Access*, vol. 5, pp. 1872–1899, 2017.

[6] H. Zubaydi, M. Anbar, and C. Wey, "Review on detection techniques against DDoS attacks on a software-defined networking controller," in *Proc. PICICT*, 2017, pp. 10–16.

[7] A. Neghabi, N. Navimipour, M. Hosseinzadeh, and A. Rezaee, "Load balancing mechanisms in the software defined networks: A systematic and comprehensive review of the literature," *IEEE Access*, vol. 6, pp. 14159–14178, 2018.

[8] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[9] D. Sheinbein and R. Weber, "800 service using SPC network capability," *Bell Syst. Tech. J.*, vol. 61, no. 7, pp. 1737–1744, 1982.

[10] A. Doria *et al.*, *Forwarding and Control Element Separation (ForCES) Protocol Specification*, document RFC 5810, Internet Engineering Task Force, 2010.

[11] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking control of the enterprise," in *Proc. Conf. Appl. Technol. Architect. Protocols Comput. Commun.*, 2007, pp. 1–12.

[12] (2013). *Open vSwitch*. [Online]. Available: http://vswitch.org/

[13] E. Fernandes and C. Rothenberg, "OpenFlow 1.3 software switch," in *Proc. Brazilian Symp. Comput. Netw. Distrib. Syst.*, 2014, pp. 1021–1028.

[14] (2013). *Pica8 3920*. [Online]. Available: http://www.pica8.org/documents/pica8-datasheet-64x10gbe-p3780-p3920.pdf

[15] Juniper Networks, Inc. (2013). *Contrail Virtual Router*. [Online]. Available: https://github.com/Juniper/contrail-vrouter

[16] IBM System Networking. (2013). *RackSwitch G8264*. [Online]. Available: http://www-03.ibm.com/systems/networking/switches/rack/g8264/

[17] B. Pfaff and B. Davie, *The Open vSwitch Database Management Protocol Internet Engineering Task Force*, document RFC 7047, 2013.

[18] M. Smith, *OpFlex Control Protocol Internet Engineering Task Force*, Internet Draft, 2014.

[19] T. Koponen *et al.*, "Onix: A distributed control platform for large-scale production networks," in *Proc. 9th USENIX Conf. Oper. Syst. Design Implement.*, 2010, pp. 1–6.

[20] A. Ferguson, A. Guha, C. Liang, R. Fonseca, and S. Krishnamurthi, "Participatory networking: An API for application control of SDNs," in *Proc. ACM SIGCOMM Conf.*, 2013, pp. 327–338.

[21] S. Matsumoto, S. Hitz, and A. Perrig, "Fleet: Defending SDNs from malicious administrators," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 103–108.

[22] U. Krishnaswamy, *ONOS: An Open Source Distributed SDN OS*, 2013.

[23] M. Banikazemi, D. Olshefski, A. Shaikh, J. Tracey, and G. Wang, "Meridian: An SDN platform for cloud network services," *IEEE Commun. Mag.*, vol. 51, no. 2, pp. 120–127, Feb. 2013.

[24] M. Raju, A. Wundsam, and M. Yu, "NOSIX: A lightweight portability layer for the SDN OS," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 28–35, 2014.

[25] D. Erickson, "The beacon openflow controller," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw.*, 2013, pp. 13–18.

[26] N. Gude *et al.*, "NOX: Towards an operating system for networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 105–110, 2008.

[27] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *Proc. 2nd USENIX Conf. Hot Topics Manage. Internet Cloud Enterprise Netw. Services*, 2012, p. 10.

[28] C. Demetrescu and G. F. Italiano, "A new approach to dynamic all pairs shortest paths," *J. ACM*, vol. 51, no. 6, pp. 968–992, 2004.

[29] *Open Service Gateway Initiative*. Accessed: Nov. 11, 2017. [Online]. Available: https://www.osgi.org/

[30] S. Shin *et al.*, "Rosemary: A robust, secure, and high-performance network operating system," in *Proc. 21st ACM Conf. Comput. Commun. Secur.*, Scottsdale, AZ, USA, 2014, pp. 78–89.

[31] M. Accetta *et al.*, "Mach: A new kernel foundation for unix development," in *Proc. USENIX Conf.*, 1986, pp. 93–112.

[32] A. Tavakoli, M. Casado, T. Koponen, and S. Shenker, "Applying NOX to the datacenter," in *Proc. 8th ACM Workshop Hot Topics Netw. (HotNets-VIII)*, 2009, pp, 1–6.

[33] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. USENIX NSDI*, 2010, p. 19.

[34] R. N. Mysore *et al.*, "PortLand: A scalable fault-tolerant layer 2 data center network fabric," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 39–50, 2009.

[35] Z. Cai and A. Cox, "Maestro: A system for scalable OpenFlow control," Rice Univ., Houston, TX, USA, Tech. Rep., 2011.

[36] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *Proc. IMC*, 2009, pp. 202–208.

[37] T. Benson, A. Akella, and D. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. IMC*, 2010, pp. 267–280.

[38] J. Medved, R. Varga, A. Tkacik, and K. Gray, "OpenDaylight: Towards a model-driven SDN controller architecture," in *Proc. IEEE 15th Int. Symp. World Wireless, Mobile Multimedia Netw.*, Jun. 2014, pp. 1–6.

[39] M. Bjorklund, *YANG—A Data Modeling Language for the Network Configuration Protocol (NETCONF)*, document RFC 6020, Internet Engineering Task Force, 2010.

[40] A. Tootoonchian and Y. Ganjali, "HyperFlow: A distributed control plane for OpenFlow," in *Proc. Internet Netw. Manage. Conf. Res. Enterprise Netw.*, 2010, p. 3.

[41] R. Sherwood *et al.*, "FlowVisor: A network virtualization layer," Deutsche Telekom Inc. R&D Lab, Stanford, Nicira Netw., Stanford, CA, USA, Tech. Rep. 1, 2009.

[42] J. Stribling *et al.*, "Flexible, wide-area storage for distributed systems with wheelfs," in *Proc. 6th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, Boston, MA, USA, 2009, pp. 43–58.

[43] F. Botelho, A. Bessani, F. Ramos, and P. Ferreira, "SMaRtLight: A practical fault-tolerant and controller," Cornell Univ. Liberary, Netw. Internet Archit., Ithaca, NY, USA, Tech. Rep., 2014.

[44] Project Floodlight. (2012). *Floodlight*. [Online]. Available: http://floodlight.openflowhub.org/

[45] F. Botelho, F. M. V. Ramos, D. Kreutz, and A. Bessani, "On the feasibility of a consistent and fault-tolerant data store for SDNs," in *Proc. EWSDN*, 2013, pp. 38–43.
[46] P. Berde, "ONOS: Towards an open, distributed SDN OS," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 1–6.
[47] J. Ousterhout *et al.*, "The case for RAMClouds: Scalable high-performance storage entirely in DRAM," *SIGOPS Operat. Syst. Rev.*, vol. 43, no. 4, pp. 92–105, 2010.
[48] *Hazelcast Project*. Accessed: Oct. 22, 2017. [Online]. Available: http://www.hazelcast.org/
[49] *What's Behind Network Downtime?* Sunnyvale, CA, USA, 2008.
[50] A. Ferguson, A. Guha, C. Liang, R. Fonseca, and S. Krishnamurthi, "Participatory networking: An API for application control of SDNs," in *Proc. ACM SIGCOMM Conf.*, 2013, pp. 327–338.
[51] R. Sherwood and K. Yap. *Cbench: An Open-Flow Controller Benchmarker*. Accessed: Dec. 2, 2017. [Online]. Available: http://www.openflow.org/wk/index.php/Oflops
[52] Nippon Telegraph and Telephone Corporation. (2012). *RYU Network Operating System*. [Online]. Available: http://osrg.github.com/ryu/
[53] *POX*. (2012). [Online]. Available: http://noxrepo.org/

MANISH PALIWAL received the B.E. degree in computer science and engineering from UIT-RGPV, Bhopal, in 2012, and the M.Tech. degree from the Government College of Engineering at Amravati, Amravati, in 2015. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, Visvesvaraya National Institute of Technology, Nagpur, India.

DEEPTI SHRIMANKAR received the B.Tech. degree in computer technology and the M.Tech. degree in image processing from Nagpur University in 1997 and 2007, respectively, and the Ph.D. degree in parallel computing from the Visvesvaraya National Institute of Technology, Nagpur, India. She is currently an Assistant Professor with the Department of Computer Science and Engineering, Visvesvaraya National Institute of Technology.

OMPRAKASH TEMBHURNE received the B.E. degree in computer engineering and the M.Tech. degree from Rastrasant Tukadoji Maharaj Nagpur University in 2009 and 2012, respectively. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, Visvesvaraya National Institute of Technology, Nagpur.