IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 8, 2019, accepted March 21, 2019, date of publication April 4, 2019, date of current version April 18, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2909356

Traffic Load Balancing Using Software Defined Networking (SDN) Controller as Virtualized Network Function

SIKANDAR EJAZ[1], ZESHAN IQBAL[1], PEER AZMAT SHAH[2], BILAL HAIDER BUKHARI[3], ARMUGHAN ALI[3], AND FARHAN AADIL[3]

[1]Department of Computer Science, University of Engineering and Technology at Taxila, Taxila 42050, Pakistan
[2]Erik Jonsson School of Engineering and Computer Science, The University of Texas at Dallas, Richardson, TX 75275, USA
[3]Department of Computer Science, COMSATS University Islamabad, Attock Campus, Attock 43600, Pakistan

Corresponding author: Farhan Aadil (farhan.aadil@cuiatk.edu.pk)

ABSTRACT SDN and NFV are collaboratively recognized as the most promising bearing for flexible programmability of network control functions and protocols with dynamic usage of network resources. SDN provides the abstraction of network resources over well-defined APIs to achieve underlying topology-independent multiple tenant networks with required QoS and SLAs. NFV paradigm deploys network functions as software instances, namely, VNFs on commodity hardware using virtualization techniques. In this way, virtual IP functions, such as load balancing, routing, and forwarding or firewall, can operate as VNF in a cloud with a positive outcome in network performance. In this paper, we aimed to achieve traffic load balancing by using a virtual SDN (vSDN) controller as a VNF. With vSDN, when there is uneven and increased load, secondary vSDN controllers can be added to share this load. The need of secondary vSDN is determined and a copy vSDN with exactly the same configurations as original vSDN is created, which operates accurately and shares traffic load balancing tasks with an original vSDN controller. Both vSDN controllers are independently placed in the cloud with transparency assuring that every client in the network is familiar with the existence of the newly created secondary vSDN controller. We experimentally validated the load balancing in Fat-Tree topology using two vSDN controllers in a Mininet emulator. The results showed 50% improvement in average load, 41% improvement in average delay, and considerable improvements in terms of ping response, bandwidth utilization, and throughput of the system.

INDEX TERMS Load balancing, network function virtualization (NFV), software defined networking (SDN), virtual SDN controller (vSDN).

I. INTRODUCTION

Software Defined Networking is a constantly progressive technology that offers more flexible programmability support for network control functions and protocols. SDN provides logical central control model for implementation and maintenance of programmable networks by utilizing the concept of decoupling of data plane and control plane [1] over a well-marked and comprehensible controlling protocol like OpenFlow Figure 1. OpenFlow is one of the control plane protocols standardized as per Open Networking Foundation's (ONF) [2] recommendation for interfacing of components with their lower-level components in the network. It allows the policies, logical switch abstraction, configuration, outlining of high-level instructions and network resource administration to initiate functionalities in small timelines to hide the vendor-specific component details, enhancing the ability of hardware to use and exchange information in multi-vendor distributions and environments [3]. Controller in the SDN paradigm uses this solitary control protocol to provide abstraction of a wide variety of network functions including routing and forwarding technologies, traffic engineering, management and access control through an Application Programming Interface (API). A network hypervisor can be deployed from this abstraction to virtualize the network to achieve network protocol and underlying topology-independent multiple Virtual Tenant Networks (VTNs) [4] functioning at the same time with physical infrastructure.

The associate editor coordinating the review of this manuscript and approving it for publication was Tariq Ahamed Ahanger.

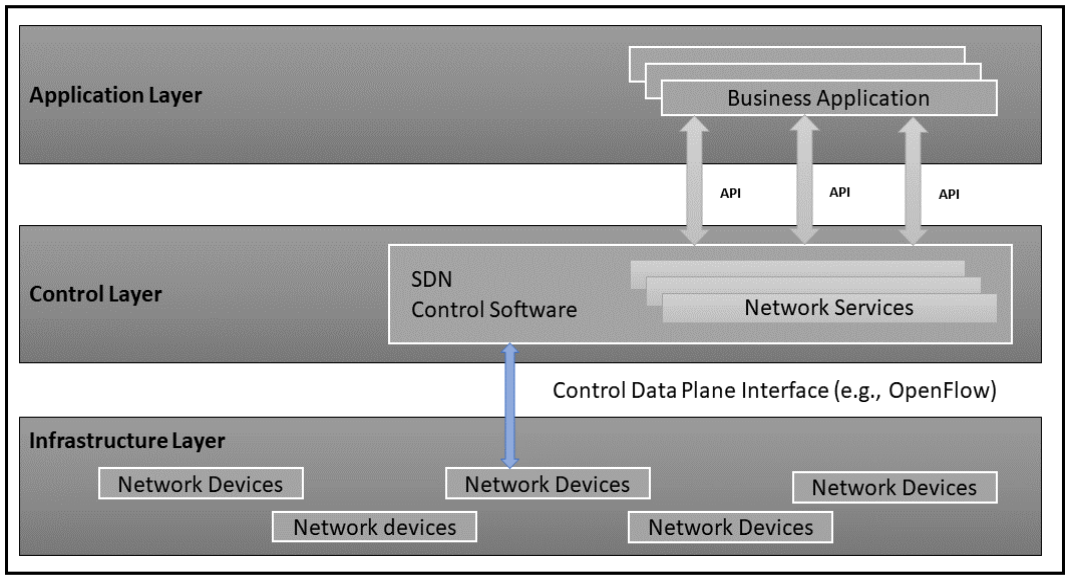

FIGURE 1. SDN three-layer reference model [5].

Separate controller instances independently handle network functions and ensure Service Level Agreement (SLA) and Quality of Service (QoS) in VTNs.

Proprietary characteristics of hardware components, cost and insufficiently skilled professionals make it difficult to bring and integrate new services to meet the user requirements. Combination of Network Function Virtualization (NFV) and associated technologies such as SDN and cloud computing are now capable of reducing these issues [6], [7]. NFV supports the separation of software control instances from hardware infrastructure for faster provisioning of network functions and services by means of software virtualization [8]. It employs the network functions on demand (no need of installation of new equipment for instantiation of virtual appliances), decouples them from location and virtualizes them on standardized commodity servers, switches and storage. This way, capital expenditures and energy consumption are decreased, and a lower-cost smart network infrastructure is achieved [9], [10] along with the benefits of changing innovation cycle for network operators such as rapid and efficient introduction of targeted and custom services according to user's needs. However, once network functions get virtualized and turned into Virtualized Network Functions (VNFs), NFV leads to raise some network performance related issues [8], [9] like throughput instability and unusual latency variations in just fewer network utilization. Therefore, smooth migration of tightly coupled large scale existing networks to NFV-based solutions with efficient deployment and accurate functioning of VNFs becomes a challenge. Similarly, the decoupling of control operations from location also generates the problem of effectual placement and dynamic on-demand instantiation of the virtual appliances.

A. BACKGROUND

Usually, SDN controller distributions for tenant networks are open-source implementations, such as Floodlight, OpenDaylight, Ryu, POX, ONOS and Trema etc. Each VTN contains independent SDN controller running on a dedicated host. So, SDN controller is essential to be physically deployed and configured at dedicated host at time of each dynamic VTN employment. This implementation of SDN controller adds delays of several days in required service provisioning. Virtualization of the SDN controller functions by means of NFV paradigm is supposed as a more sophisticated approach for utilization of network functions including load balancing, routing and forwarding, firewall and traffic engineering. NFV gives the idea of virtualizing the SDN controller and moving it into the cloud for dynamic deployment and required connectivity of autonomous SDN controller prototypes within minutes. Consequently, whenever a new VTN is deployed dynamically, the functionality of the whole network can be accomplished in a couple of minutes [11]. Moreover, this technique also offers supplementary advantages like reduction in hardware retainment pause and improvement in recovery time in catastrophe or failure conditions. A virtualized SDN controller [12] can be immediately and effortlessly moved among physical servers within a cloud of data centers when a hardware retainment is needed (less hardware retainment pause), snapshots and backups of the states of virtualized SDN controllers can be shared from one data center to another in a cloud for quick reconfiguration after a failure (faster recovery).

NFV related network functions (VNFs) includes IP network functions (load balancing, routing and forwarding, security, firewall or Authentication, Authorization and Accounting (AAA), EPC/LTE network control functions,

Serving Gateway (SGW), Mobility Management Entity (MME) and PDN Gateway (PGW) and virtualization of Path Computation Element (PCE) [13], [14]. In general, VNFs are deployed as software instances in dedicated specialized hardware in data centers or distributed computing platforms. NFV is appropriate virtualization technology for any control plane function or data plane packet processing in static and dynamic network infrastructures. Despite of all, this work focuses on virtualization of IP functions particularly traffic load management through which load balancing would be achieved to distribute the workload on several resources to avoid overload on any resource. Some load balancing goals include taking full advantage of throughput and bandwidth, minimizing the transmission delay and response time with optimized traffic flows [15], [16]. When it comes to saving of resources, load balancing can be the centralized decision based or the distributed decision based [17]. Centralized decision and distributed decision are not so efficient methods because of their processing delays and extended completion times. Centralized decision collects all load information of local controllers and sends load balancing requests to the local overloaded controller. Distributed decision [18] allows every controller to do load balancing locally without sending commands. The processing delays of centralized decision and extended completion time of load balancing in distributed decision reduces the availability and scalability of both the strategies.

However, due to today's industry concerns [15], [19], the existing methods need to be revised and load balancing functionality would be virtualized to make it dynamic, resource saving and independent of vendor-specific. In this paper, we utilize the abilities of NFV paradigm and propose traffic load balancing using SDN controller as virtualized network function (creation of vSDN). When using vSDN we have this opportunity that by the increase of load we can further add secondary vSDNs to share this load. Since all the resources (switches, routers and connections etc.) get virtualized, so we can assign/add hardware resources as per requirement. So, firstly it should be determined that when there is a need to create a copy of vSDN controller and then secondly, all nodes should learn about the existence of secondary controllers. A copy of vSDN with exactly same configurations as original vSDN operates correctly and shares traffic load balancing tasks with original vSDN controller. Both vSDN controllers independently placed in cloud with transparency assuring that there is no master controller and every host in network is familiar with the existence of the newly created secondary vSDN controller.

The remaining sections of paper are planned in a way that section II gives the literature review of formerly proposed related work and describes the intention for this research. NFV architecture and scope is discussed in section III to understand the operations and importance of NFV. Section IV is the main part of this paper, constitute the proposed system design for load balancing using vSDN controller as VNF. This section step by step describes the followed strategy. Section V and VI shows the experimental setup and obtained results respectively. Finally, section VII concludes the complete work.

II. RELATED WORK

There are some related works on load balancing of SDN controller, some of these are mentioned here. In OpenFlow descriptions, the switch configuration including flow table entries can be altered only via master c-node proposed in [20]. This master c-node is responsible for equalize the flow of incoming and outgoing messages at varying number of switches to increase the scalability. For load balancing in SDN-enabled networks, a technique called BalanceFlow was proposed in [21], in which a super controller is deployed among distributed controllers to handle uneven traffic load problem. A decision-maker controller node gathers the information about all other controller nodes and then resolves a load balancing issue by considering the load variations of all controllers. Limitations of this approach includes (i) performance compromises due to exchange of frequent control messages and limited resources like memory, bandwidth and CPU power (ii) load information is obtained with delays which do not portray the real load conditions, due to two network transmissions (sending commands and collecting loads) and (iii) Entire load balancing operation can be down if central controller collapses.

Dynamic and adaptive algorithm (DALB) proposed in [22], enabled all slave SDN controllers for local decisions just like master controller. This algorithm allows scalability and availability of distributed SDN controllers and need one network transmission for gathering load. Consequently, decision delay reduced because all controllers do not collect the load information too frequently. While considering the network resources, integration of SDN and NFV introduced in [11] to enhance the network protocol and functions programmability. NFV paradigm supports the dynamic adjustment of network resources and gives the concept of virtualized network control functions for tenant networks. This way, control function software instances can be dynamically deployed and migrated if need for efficient utilization of available resources.

Previous work on load balancing rely on physical SDN resources whether consider SDN controller in central or distributed mode. Through NFV, all the resources can be virtualized and further vSDN controllers can be added for load balancing in case of increased uneven traffic load in vSDN-enabled networks. A copy vSDN can be configured dynamically to share the load and to perform same tasks as of original vSDN. So, first issue here exist is when we need to create a copy of vSDN controller and the other issue is how nodes will know about the existence of secondary controller? Our work novels in a sense that we enhance the functionality of SDN/NFV integration and introduce IP load balancing functionality in virtual SDN controller-enabled networks by utilizing NFV paradigm so that network resources would be save with improved performance.

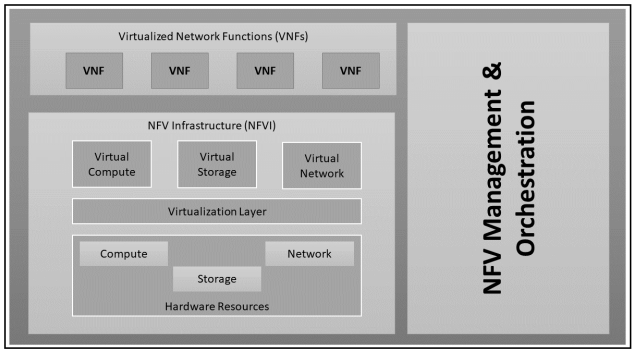

FIGURE 2. NFV architectural framework [26].

III. NFV FRAMEWORK AND SCOPE

European Telecommunications Standards Institute (ETSI) defines a three-layer NFV framework consisting of Network Function Virtualization Infrastructure (NFVI), NFV Management and Orchestration (NFV-MANO) [23] and Virtual Network Functions (VNFs). These high-level architectural functional blocks are illustrated in Figure 2. This section describes these three elements [23], [24].

A. NFV INFRASTRUCTURE

The NFVI make the environment where VNFs are employed, responsible for holding both software and hardware resources. These physical resources comprise of commercial-off-the-shelf (COTS) computation components, network and storage resources which offers processing, storing and connecting links to the VNFs. Here abstraction of physical computing, network and storage resources is known as virtual pool of resources. A hypervisor-based virtualization layer decouples the underlying hardware resources from virtual resources to achieve abstraction. Virtual networks are deployed from virtual links and nodes like VTNs while compute and storage can most likely be categorized as multiple Virtual Machines (VMs) in cloud environment. Virtual node is created by employing either hosting or routing as software component enclosed in a VM [10] while virtual link provides a logical connectivity between two or more virtual nodes but gives the impression of a direct physical interconnection having dynamically varying properties [25]. NFVI includes diverse amount of physical resources which can be virtualized along with the support for execution of VNFs.

B. VIRTUAL NETWORK FUNCTIONS (VNFS)

NFs are functional wedges in a network framework consisting of definite interfaces and functionalities [23]. They can be IP network based, EPC/LTE network control or Path Computation Element (PCE). Consequently, a VNF is implementation of NFs as software instances which is obtained by deploying a NF on virtual resources namely a VM and capable of operating over a NFVI. A single VNF may be implemented over several VMs because it can contain multiple components inside and hence each VM would host a solitary component of that VNF [26]. One or more VNFs make up services that TSP offers [10], virtualized and placed on multiple VMs but act like one service. NFV gives opportunity of same service provisioning regardless the functions running on dedicated hosts or on VM resources.

C. NFV MANAGEMENT AND ORCHESTRATION (NFV-MANO)

MANO framework proposed by ETSI enables NFV-MANO [27] to provide the required serviceability of VNFs and associated operations including deployment and configuration of the VNFs. NFV-MANO looks for life cycle management and orchestration of hardware and/or software resources with support of infrastructure virtualization. Moreover, it deals with the databases that stores the deployment and life cycle data models and information about functions, their services, and available resources. All necessary virtualization and management related tasks in NFV framework are the concerns of NFV-MANO. Interfaces for communication between different NFV-MANOs and coordination with legacy network management systems such as Business Support Systems (BSS) or Operations Support Systems (OSS) allow the management of VNFs together with the functions running on traditional equipment [10].

D. SCOPE

NFV offers realization of service provisioning to the stakeholders independent of vendor-specific hardware and software and so familiarizes in several differences with non-virtualized networks [9], [10], [26]. Major differences include:

1) DECOUPLING OF RESOURCES

As evolution of hardware and software resources is self-determining from each other. NFV enables both hardware and software to work autonomously and restrain the need of integration of hardware and software entities.

2) DYNAMIC FUNCTIONALITY OF VNFS

Performance of VNFs can be scaled in more flexible and diverse way with finer granularity due to presence of instantiable software components when functionality of network functions is decoupled. Based on current network settings, network operators can scale NFV efficiency on grow-as-you-need basis.

3) FLEXIBLE EMPLACEMENT OF NETWORK FUNCTIONS

Presence of pool of infrastructure resources makes network function instantiation automated. These instances may deliver dissimilar functions and services at different time in distinct data centers. This encourages the quick and intelligent deployment of new services over the corresponding physical framework.

IV. SYSTEM DESIGN AND IMPLEMENTATION

NFV offers effective dealing of VNFs and associated services in dynamic network infrastructures. When using vSDN as

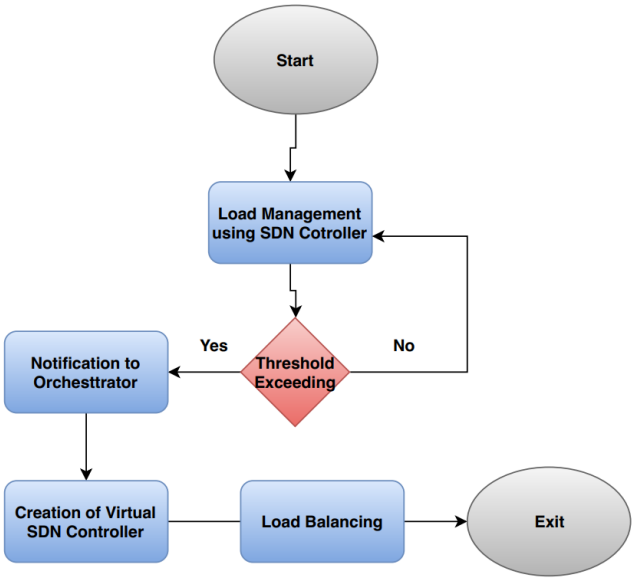

FIGURE 3. Flowchart for the proposed system.

a VNF, we have this opening that we can further add more identical VNFs for the same task to share the traffic passing through underlying network. In case of increased uneven traffic load, a secondary vSDN can be created for load balancing in vSDN-enabled networks. Since all the resources (switches, routers and connection etc.) are utilized virtually under NFV, so we need to assign/add hardware resources as per requirement. Need for a secondary vSDN controller is determined and a copy of vSDN controller created with exactly same configurations as original vSDN which work accurately and shares traffic load. Both vSDN controllers independently placed in cloud with transparency assuring that every client in network is familiar with the existence of the newly created secondary vSDN controller. In this section we present the proposed model for traffic load balancing in tenant networks using SDN controller as VNF. The strategy we follow is represented in Figure 3 in the form of flow diagram.

A. PROVISIONING OF VSDN CONTROLLER

A network hypervisor aggregates or/and partitions the physical transport network resources in virtual resources and then provide connectivity to form multiple end-to-end VTNs. Each VTN may possess a different VNT topology and may co-exist with the same physical infrastructure [11]. This hypervisor discovers the network by representing the abstracted topology of each VTN and provisions an independent tenant SDN controller for remote control of that VTN. It creates, modifies and deletes connections for VTNs and allocated resources dynamically. On application demands, a network hypervisor can create, modify and delete VTNs dynamically in response concluded from a matrix relating resource requirement and connections [28]. Usually, the SDN controller of each tenant network (physical or virtual) deployed at physical server, but through SDN/NFV orchestration and management, network

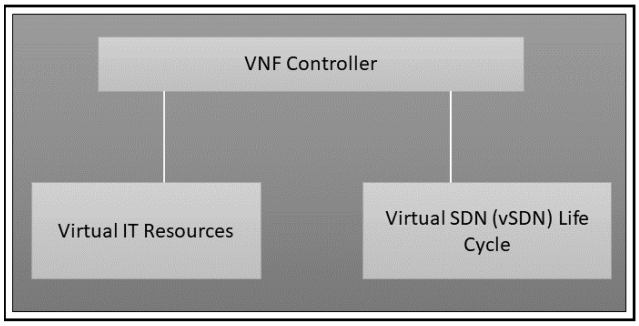

FIGURE 4. vSDN manager architecture.

control functions namely SDN controller can also be virtualized (create vSDN) and moved into the data centers of cloud [29] to implement independent controller prototypes dynamically within minutes. This way, vSDN controllers operate as Virtual Network Functions (VNFs) in cloud.

NFV Infrastructure (NFVI) is comprise of transport network hardware resources (compute, storage and network) interconnecting distributed servers in data centers. A NFVI virtualization layer is there on top of the physical resources which is based upon a NFVI manager, namely, Virtualized Infrastructure Manager (VIM), sometimes referred as cloud controller in NFVI-MANO. VIM is in charge for managing and provisioning of Virtual Machines (VMs). Next layer consists of some VNF managers [30] that oversee the VNF's life cycle supervision (i.e., create, configure, and remove). When using SDN controller as a VNF, particularly the virtualized SDN controller managers - vSDN managers are deployed which supervise the creation of SDN controller-enabled VMs in cloud Figure 4.

Finally, the orchestrator for SDN-enabled tenant networks provides a generic network abstraction mechanism and oversee the entire process from creation of new vSDN controllers (placed into the cloud), deployment of VTN, and connections between that VTN and the vSDN controllers. SDN/NFV orchestration architecture by deploying vSDN controller as VNF is displayed in Figure 5.

For provisioning of the new vSDN controller, orchestrator appeals to the vSDN manager and specify the required SDN controller distribution (e.g., OpenDaylight, ONOS, POX or Floodlight etc.). Then the vSDN manager forwards this request towards the VIM which forms a new VM containing pre-installed desired SDN controller. This vSDN contained VM is deployed in a host server near to the corresponding tenant network so that latency would be minimized. vSDN manager informs orchestrator and replies with IP address of up and running vSDN controller. Then, the second appeals that an orchestrator makes is of connectivity. It calls for the provisioning of flow between the vSDN controller and the corresponding tenant network. After creation of connection, orchestrator requests the network hypervisor to form VTN with desired topology graph and given IP address of vSDN controller. This topology graph is a combination of virtual nodes and links which represents VNT as a single virtual node

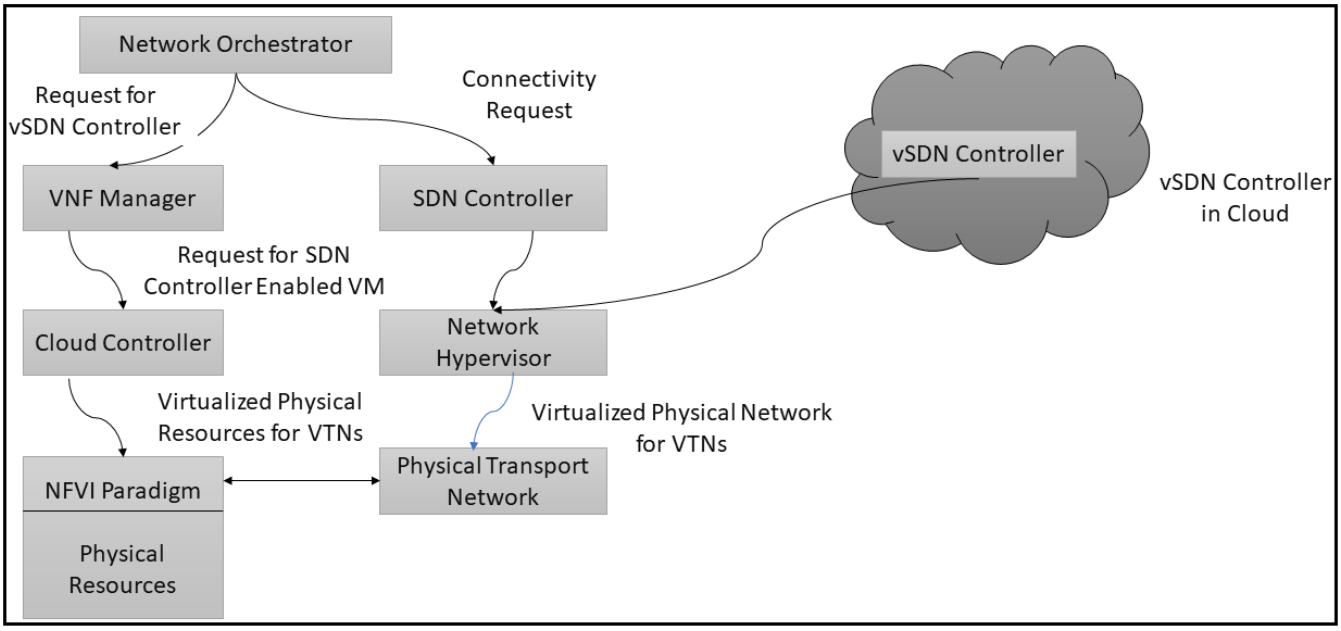

FIGURE 5. SDN/NFV orchestration using vSDN controller as VNF.

or VNT as a set of virtual links as a connection through the physical nodes. At time when the entire process is successfully completed, vSDN starts its functionality and a tenant network is controlled and managed SDN controller located in the cloud [31]. The entire process is demonstrated in Figure 6.

B. CONGESTION DETECTION

A vSDN controller act as a strategic control point and manages flow control of the switches and routers through southbound APIs in deployed transport networks. The forwarding functionality of controller concerns with the decision making for incoming flows i.e. what to do with each incoming packet, where a flow defines a group of packets transmitted from one network endpoint or multiple endpoints to other endpoint or multiple endpoints. Whenever flow reaches to a certain limit and controller utilization reaches to a threshold limit, congestion detection component of controller notifies about congestion. The threshold decision is determined by using the parameters like CPU, RAM and network congestion/throughput. Here three components, topology creation component, host management component and congestion detection component of the controller work together. Topology creation component discovers and stores link status to form current network topology. This component sends Link Layer Discovery Protocol (LLDP) packet on all ports for identification of links and then switches replies with required information. The current network topology is stored, and this information is accessible and helpful for further use of the controller [32].

Host management component manages all discovered hosts in network by storing the necessary information together with MAC and IP addresses of source host and destination host, OpenFlow switches IDs, connected nodes and number of available ports of OpenFlow switches. This information is reserved for next step so that the proper route and shifting on secondary vSDN controller for large flow would be done if congestion occurs in the network. The main component in this entire method is congestion detection component, which sets periodic queries and stores statistics from all OpenFlow switches. Obtained statistics are utilized to identify large flows and then compute load on various links so that whenever a flow reaches the threshold limit, it would be detected immediately. For congestion detection, vSDN controller gathers statistics per table, per port and per flow by polling request of STATS_REQUEST message given to PORT, TABLE and FLOW in network after fixed intervals. As a response, switches in topology replies the controller with STATS_REPLY message [32].

$$L_{Trans} = \frac{L_c - L_{thr}}{L_{thr}} \tag{1}$$

The vSDN controller observes the transmitted data bytes at the ports of every switch periodically. At time when the transferred data bytes get 70% greater than that of the link capacity, it is supposed to reach the threshold and congestion conditions come to occur in the controller. From Equation 1, overload transferred bytes can be determined, where L_c denotes the current load bytes, L_{thr} is the threshold load value of controller and L_{trans} is increased portion of transferred load. Upon identification, the large flow which induces congestion are reserved and control of that flow is inferred to shift on secondary vSDN controller for load balancing.

Equation 1 gives overloaded data bytes that pass over the 70% threshold of the link capacity. This identification is supposed as the fulfillment of congestion conditions. Need for creation of a secondary vSDN is verified here. As vSDN

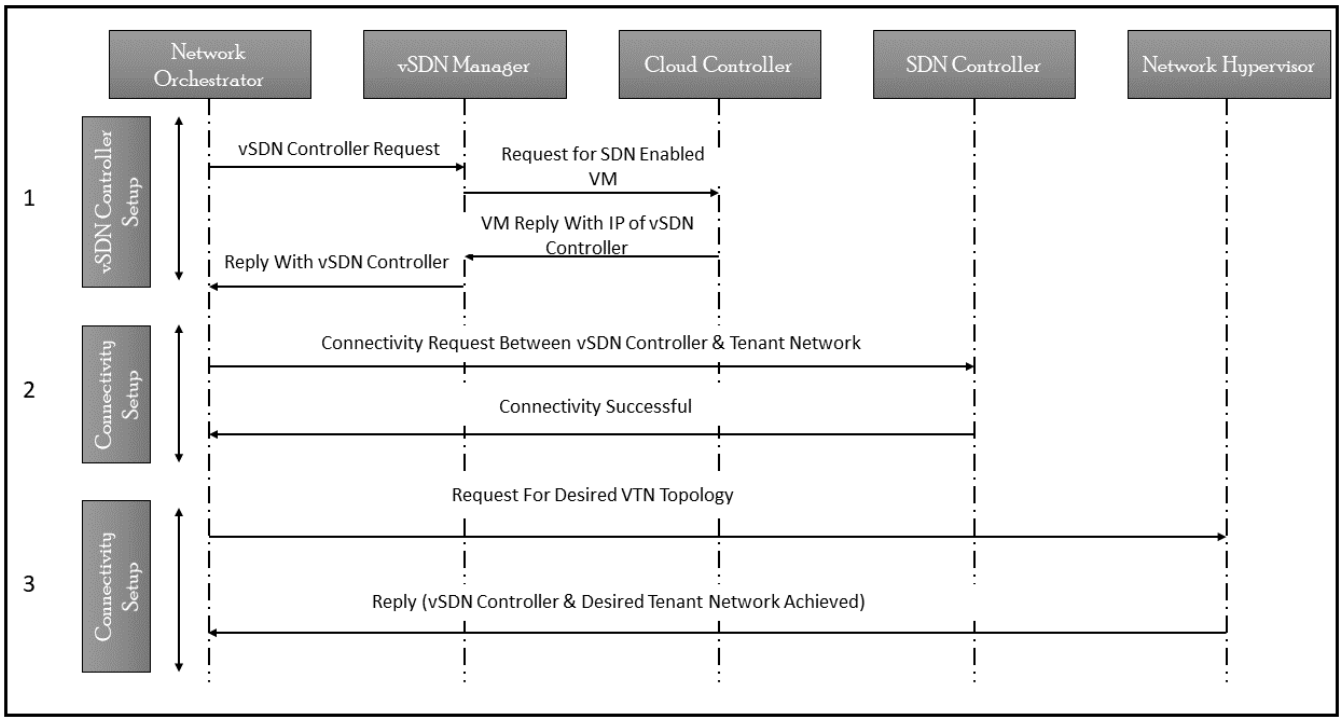

FIGURE 6. vSDN provisioning workflow.

controller is aware of the congestion and works as a VNF, it notifies orchestrator about this congestion. At this time, orchestrator and vSDN manager come into play. Orchestrator requests the vSDN manager for creation of a new vSDN manager. Primary vSDN doesn't select the subsequent controllers. This selection is responsibility of vSDN manger that creates another SDN controller-enabled VM from available network resources with exactly same configurations as original one and that involves the same operating system, configurations and flow table entries.

While performing simulations 2 controllers are used, since vSDN is taken as VNF and its creation and termination are dynamic which makes our proposed method scalable, so there may be third one or up to N if needed. Even so, we believe that one vSDN controller and a supporting secondary vSDN controllers are enough for handling of increasing load and corresponding network functions until an unexpected load is observed which may approach the threshold of both the controllers. Load is distributed among all the other controllers which is greater than the capacity of each previously created controller. For instance, the load more than threshold of first controller will be shifting to second controller, if there is need of third one, then extra load of second controller will be shifted to third and so on. On the other hand, decrease in load will lead to the removal of each newly created next in line controller, to prevent the underutilization of network resources.

C. VSDN CONTROLLER DUPLICATION AND MIGRATION

vSDN controller duplication becomes unavoidable on validation of congestion detection. In this regard, vSDN controller informs orchestrator about need of a secondary vSDN controller so that congestion would be eliminated, and network performance would not be compromised. As vSDN controller works as a VNF, so on this notification, orchestrator requests the vSDN manager for dynamic creation of another SDN controller-enabled VM with exactly same configurations as original one, namely secondary vSDN controller (duplicate or create copy of primary vSDN controller with same operating system, configurations and flow table entries). Consequently, whole process of vSDN provisioning is repeated which is described earlier, takes a short time duration for getting up and running. This VNF instance is also moved into the cloud to ensure transparency to the users. In view of this, two identical virtual appliances control the same tenant network without any break in ongoing services.

D. TRAFFIC LOAD BALANCING USING VSDN CONTROLLER

As newly created secondary vSDN controller gets the list of all clients connected to primary controller and knows about the topology and network connections, so it broadcasts its existence by sending a FEATURE_REQUEST message to all the hosts and wait for reply so that all hosts register secondary vSDN as their controller. As a reply, hosts update their flow tables and register with secondary vSDN controller and provide feature information for instance, the data-path ID (DPID) and list of ports etc. So previously unaware hosts of vSDN controller simultaneously connect to multiple controllers in network.

$$w_k = \Sigma_{i=1}^{n} \frac{L_o}{c_o} + \Sigma_{i=1}^{n} \frac{L_i}{c_i} \tag{2}$$

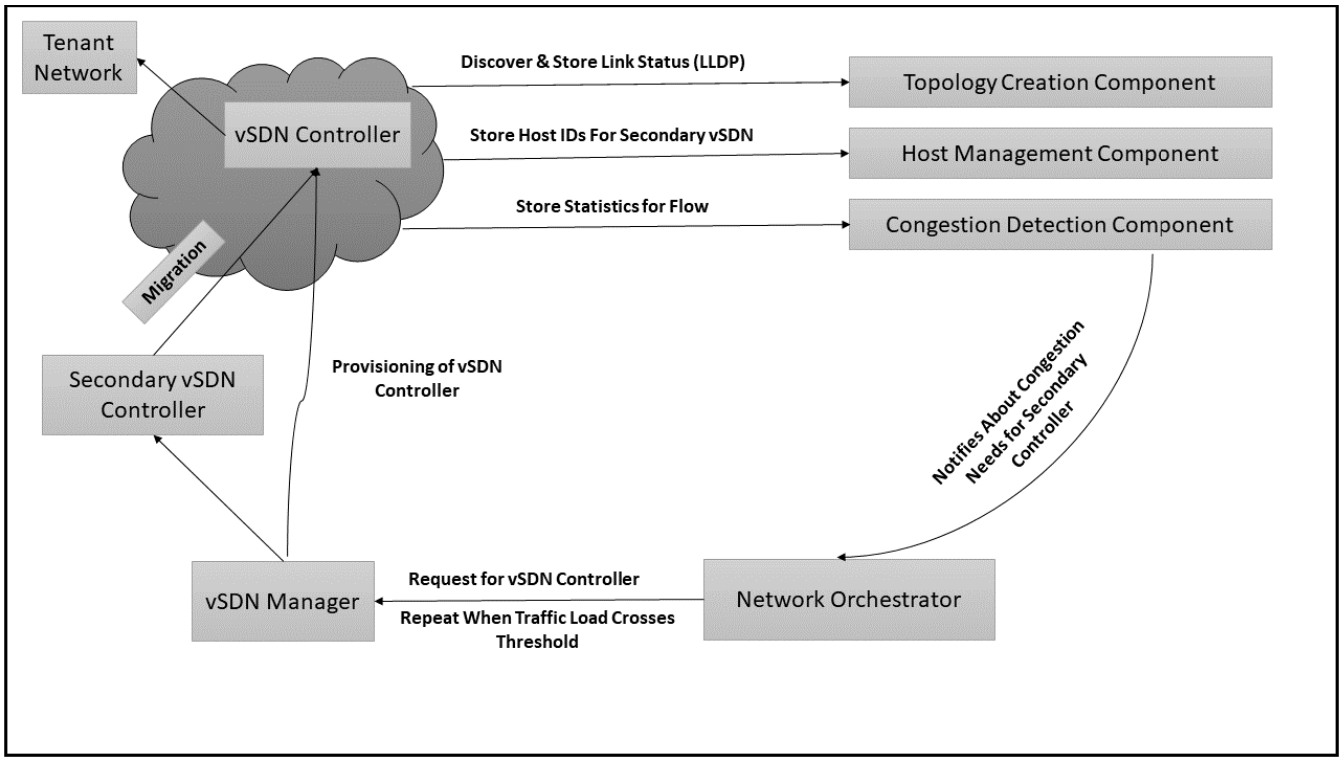

FIGURE 7. Proposed system design for traffic load balancing by utilizing vSDN controller.

For load balancing, excessive load is shifted to the secondary vSDN controller and then on the bases of gathered statistics by congestion detection component, it determines the minimum burdened shortest paths among available set of shortest paths. Equation 2 gives the total cost of each path, here one path is defined as $w_k \epsilon S$, w_k represents path and S represents the set of available paths. L_i and c_i denotes the link load and link capacity respectively. Initially, all the paths have predefined fixed load L_0 and capacity c_0. The L_i and c_i are estimated from statistics after threshold reaches and a gradual change occurs in both the parameters. The path with minimum w_k is selected from S and current flow table is updated by a OFP_FLOW_MOD message. Finally, the re-routing process re-routes the traffic on alternative paths.

$$\beta = \frac{\frac{1}{k}\Sigma_{i=1}^{k} L_i, \ldots, L_k}{L_{max}} \tag{3}$$

The load balancing rate is defined in Equation 3, where $L_i, .., L_k$ represents the load of entire system including controller. The value of β varies between 0 and 1. When β is close to 0, it means that there is no need of load balancing operation. While when β crosses the 0.7, load balancing works and load is distributed based on (2). One more important thing is realized here, that is, whenever traffic load decreases from the specified value (value of β goes less than 0.7, $\beta <$ 0.7) and it seems like there is no need for the secondary vSDN controller, vSDN manger can request VIM for the deletion of secondary vSDN-enabled VM and restoration of resources accordingly. Figure 7 shows the functional blocks for proposed system including secondary vSDN controller for load balancing. The algorithms used during research work are provided below. Algorithm 1 is used for *Creation* of SDN Controller. Algorithm 2 is used for establishing *Connection* between the newly created SDN Controller and the hosts. For detecting and minimizing *Congestion* Algorithm 3 is used.

Algorithm 1 is used for creation of secondary SDN controller. The process is initiated by the existing SDN Controller, when it detects that the traffic load exceeds threshold

Algorithm 1 SDN Controller Creation

```
1: CreateSDN()
2: {
3: i=1
4: ZQ:
5: GetNewMessage Message (i)=SDNManager(OD,ONOS,POX);
6: CloudCtrl ctrlMsg =message(i);
7: Create VM(i);
8: VM(i)=New SDNCtrl(PredefinedParameters);
9: vSDN= VM(i);
10: if β < ω then
11:     hostServerS == vSDN(i)
12: else
13:     GOTO ZQ;
14: end if}
```

Algorithm 2 Connection Creation

```
CreateConnection( ) {
newConnection = vSDN(i);
getIP IP = vSDN(i);
getTGraph GP = vSDN(i)
new VTN = VTN (IP,GP);
startFlow newFlow(VTN) }
```

Algorithm 3 Congestion Control

```
CongestionControl() {
SDN newMsg;
newMsg=STATS_REQUEST(PORT,TABLE,FLOW);
Y=newMsg;
if Y >= 70% then
    SecvSDN = newvSDN(i + 1);
    vSDN(i + 1) = vSDN(i);
    vSDN(i + 1) = messagemsg(FEATURE_REQUEST);
    HostList = vSDN(i + 1);
    i= i + 1;
end if}
```

it notifies the SDN Manager. SDN Manager then sends a *ctrlMsg* to the Cloud Controller for the creation of another SDN Controller. The Cloud Controller creates a VM and assigns a copy of existing SDN Controller with exact same parameters to the newly created VM. This creates a Virtual SDN Controller (vSDN) identical to the existing Controller as a Secondary SDN Controller.

After creation of Secondary SDN Controller the next step is to establish connection between the newly created controller and the hosts in the network. Algorithm 2 starts its working and introduces the controller to the hosts. This is done by creating a new connection for the newly created vSDN Controller. The vSDN Controller then gets IP address and Graph of the network while combining these two to create its own Virtual Tenant Network (VTN). Finally, the vSDN Controller disseminates its newly established Flows to the network hence making introducing itself to the hosts.

Algorithm 3 lets the SDN Controller work as a congestion detector in the system. This is done by creating a *newMsg* by SDN Controller. This specific message is used to read statistics of the data being transmitted between the hosts. When this message starts consuming more the 70% resources of the network, as discussed in this section previously, the SDN Controller sends request for creation of another SDN Controller to the SDN Manager. The SDN Manager then creates secondary vSDN Controller and divides the traffic load on both the controllers for the sake of load balancing. This whole process allows the SDN Manager to manage traffic load throughout the network efficiently.

V. EXPERIMENTAL VALIDATION

Mininet has been used to perform experimental validation of the proposed methodology, as Mininet can create realistic virtual network topology with application code with SDN support on a single machine in seconds. We used Fat-Tree topology as representative data center network infrastructure because Fat-Tree has identical bandwidth at any bisections, depicted in Figure 8. In our topology, switch IDs are in decimal and hexadecimal to avoid conversion complications. For traffic generation, we have considered iPerf since it provides active measurement of link utilization. iPerf is open source and useful for the assessment of the traffics which is generated over TCP and UDP with the support of several types of measurement scales including throughput, link utilization and data rate. When using iPerf, the data packets with definite size and rate are conveyed in a specific number of hosts. This method generates 56-byte TCP data packets at a rate of 120Kbps at 8-pairs of VMs with a straightforward Python script and executing in the proper network namespace created in Mininet.

This emulated setting works on a solitary Intel i7 2.4GHz CPU, 16GB RAM, running Ubuntu 16.04. The generated traffic rate has kept relatively tolerant, but it doesn't affect the validity of our experiments. VMs are created in VirtualBox hypervisor containing Ubuntu 16.04 installed with allocation of 8GB memory to the virtual system and left the CPU allocation default. In large infrastructures, like in real data center environments, the communication between hypervisors and one SDN controller can slow down the performance of the controller and the network, so to avoid these kind of scenarios, we prefer the use the remote-control plane. Along with this setup, Wireshark is used for capturing packets & graphs related to packets size, bandwidth utilization and load-balancing. We used OpenDaylight controller as SDN controller [33] which acted as the main controller throughout the experiments. OpenDaylight is java based open source SDN controller. The aim of OpenDaylight controller is to provide a functional SDN platform which allows users to directly deploy SDN controller virtually. Figure 8 shows the topology used during experiments, consist of eight terminal hosts and ten switched. Switch IDs are shown here in figure next to each switch while port numbers are shown near the links. The port numbers may vary when code is executed in mininet.

VI. RESULTS AND DISCUSSIONS

In our experiment we first deployed a single remote vSDN controller namely OpenDaylight controller Beryllium distribution and connected it with an abstract Fat-Tree topology in Mininet emulator. Initially, connectivity information is achieved, such as information about all connected hosts, their connected switches, their IP addresses, MAC addresses and port mapping etc. Then statics about links are gathered periodically so that it would be notified whenever traffic load reaches to threshold limit. At time when the transmitted traffic is 70% higher than that of the link capacity, secondary vSDN is deployed for the same topology. At this stage network topology is controlled by two identical controllers. Route/path availability information between two hosts is obtained using Dijkstra in a way limiting the search of

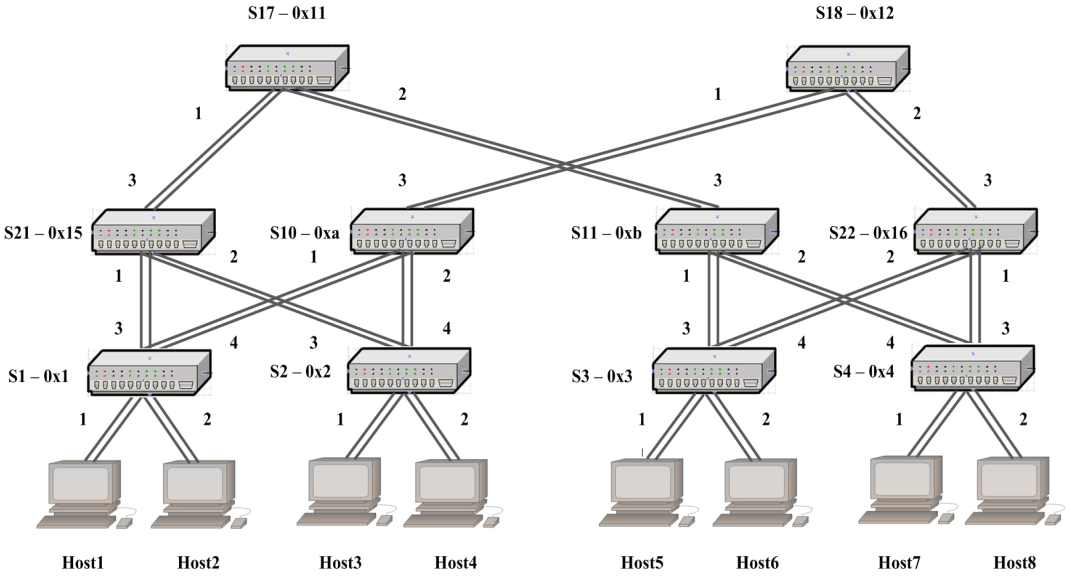

FIGURE 8. Network topology used during experiments.

shortest paths to the only one section of Fat-Tree topology where load balancing has to be performed. Requests are made to calculate total cost of links for all the paths between two hosts in terms of transmitted data. The packet flows are shaped by considering the minimum transmission cost of links at the current time and the best path is determined, and static flows are moved to the other controller and to every switch which lies in the given best path. Substantial information such as source IP, source MAC, destination IP, destination MAC, in-port, and out-port is provided to all flows. The program periodically updates this information after minute in so doing make it dynamic. Wireshark was used to capture and analyze the connectivity between hosts when controller is running and connected to the topology created in Mininet.

Figure 9 to Figure 13 present the results achieved prior to and after load balancing. We present the results in Figure 9 to Figure 11 including load rate, pinging and link capacity in Gbps for Host1 to Host4 & Host2 to Host6 as a sample in our topology, but these results can be acquired for any host in the network.

Figure 9 illustrates Load Variation on a Link from Host 1 to Switch S1 between Host1 to Host4 & Host2 with Switch S1 between Host2 to Host6 with variation in time on x-axis. Without load balancing, the load increases with the passage of time. However, in case of proposed method that load on a single link decreases after load balancing because load get distributed on alternative paths. At start the load of the proposed system is high, it is because of the number of hosts and the amount of data they are communicating with each other. However, this high load at start does not affect the performance badly, because enough resources are available at the start of simulation for each VM. In case of no load balancing, the load increases with time and the scarce resources also start

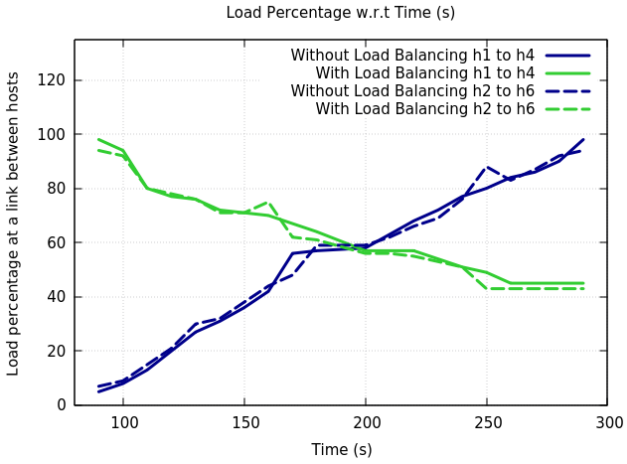

FIGURE 9. Load variation on a link between Host1 to Host4.

to decrease which may result in availability or decrease in performance of VM.

Figure 10 shows improvement in iPerf pinging prior to and after the load balancing for Host1 to Host4 & Host2 to Host6. This figure clearly depicts that the Round Trip Time (RTT) has decreased significantly due to the proposed load balancing scheme. When there were 40 packets, the average ping time for scheme which does not uses any load balancing system i.e. existing SDN was 0.35 seconds. However, for the same number of packets (load), the proposed scheme reduced the average ping time to 0.15 seconds which is more than 50% improvement. This decreased ping time is due to the fact that the proposed load balancing distributed the load.

Figure 11 gives the idea of bandwidth enhancement after load balancing for the same hosts. It can be seen from figure that average link capacity of the links has improved

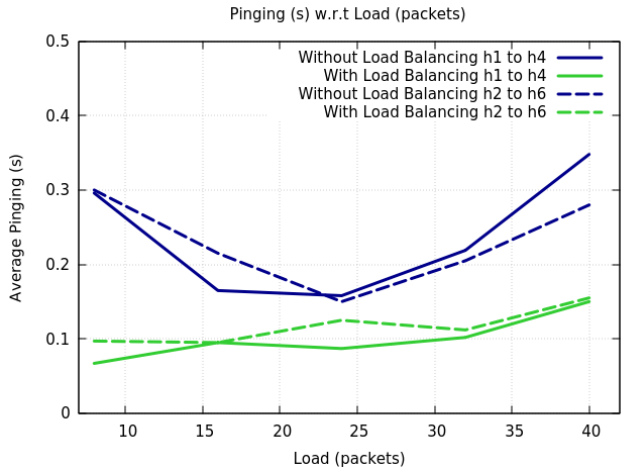

FIGURE 10. Pinging between Host1 to Host4.

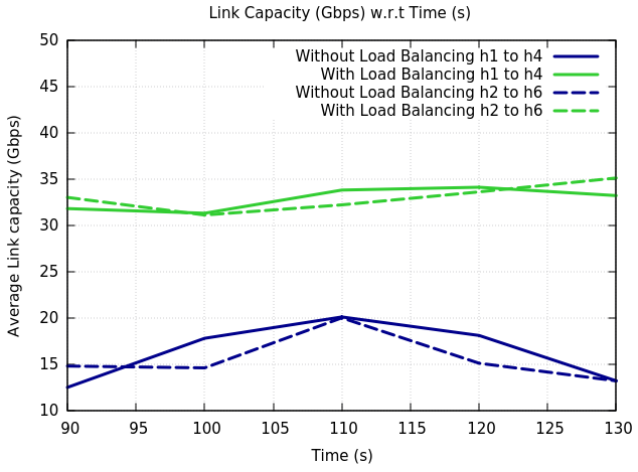

FIGURE 11. Average available link capacity between Host1 to Host4.

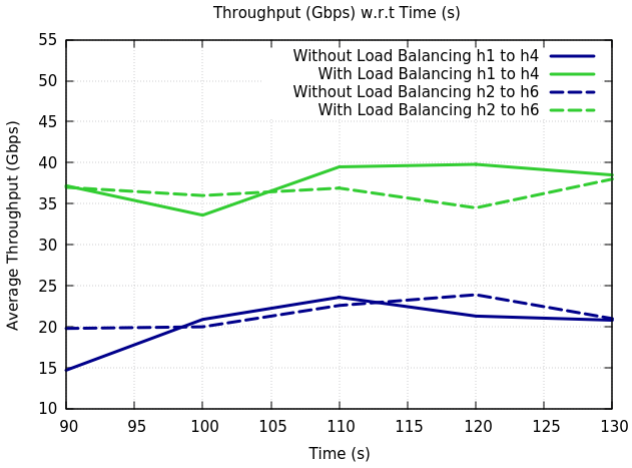

FIGURE 12. Throughput considering different time intervals.

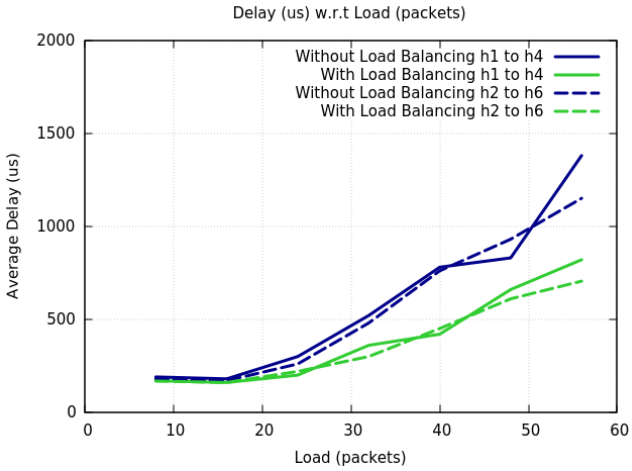

FIGURE 13. Average delay with increase in load.

as compared to the existing system. Also, the average link capacity is not decreasing at the same rate with the passage of time as it is decreasing for the existing solution. This decrease rate is very slow, which tells that the proposed load balancing is a solution that will stabilize the network and will not decrease its performance over the passage of time.

Figure 12 indicates the throughput at different time intervals while keeping the load percentages stable. Again, the throughput is improved with the proposed load balancing scheme.

Considering the Figure 13, average delay can be seen if packet size varies in the range of 8 to 56 bytes. At start, when the load was 8 the delay for both schemes is same. However, as the load (packet size) increases the average delay in micro seconds increases with a high pace for existing scheme as compared to the proposed load balancing scheme. At load of 56, the average delay of existing scheme was 1400 micro seconds. When the proposed load balancing was applied, the average delay is reduced to 825 micro seconds for the same load. This shows 41% improvement in average delay.

Statistical and graphical comparison shows a significant improvement in average load rate, pinging, bandwidth, throughput and delay. So, it is realized that proposed approach enhances the network performance in terms of above mentioned parameters. Virtualization of control functions and use them as an VNF comes with saving of resources and better performance of network to the user satisfaction.

Most of the time, the proposed methodology in any research work turns out to be as much relatable to the problem statement as the researchers wanted. But, there are also some limitations in every research work. Our proposed work may also have some limitations, which may provide future research direction to the researchers. Following are the few limitations of the proposed methodology, discussed below:
The complexity of proposed technique is somehow high due to the consuming of multiple controllers. It may increase with the number of controllers. Another limitation of our work is, it does not mention the challenges of energy consumption and carbon emission. These issues can be independently analyzed and discussed. So, when it comes to determining the effectiveness of load balancing mechanism in terms of energy consumption and carbon emission.

VII. CONCLUSION

In this paper we presented the traffic load balancing mechanism using SDN controller as VNF in SDN-enabled networks. The proposed system allows the provisioning of a vSDN controller which is acting as a VNF service. Whenever traffic load reaches to a certain threshold, a secondary vSDN controller with exact same configuration as original can be added in the same network to share the load and tasks of original vSDN controller ultimately balancing load on both controllers. Since, all the hosts know the existence of both the controllers so exceeded load would be shifted to the secondary vSDN controller which switches the load and balances the flows among connected hosts. We performed the experiment using Fat-Tree topology as representative data center network infrastructure with OpenDaylight as SDN controller on Mininet emulator for load balancing. We found accurate working of two controllers and a rise in average pinging of hosts, transfer rate and link capacity after load balancing was witnessed. This refers to the improvement in network performance. In future, we aimed to deploy more IP network functionalities as VNF services and direct our research towards virtualization of EPC/LTE network control functions.

REFERENCES

[1] E. Haleplidis, K. Pentikousis, S. Denazis, J. H. Salim, D. Meyer, and O. Koufopavlou, *Software-Defined Networking (SDN): Layers and Architecture Terminology*, E. Haleplidis and K. Pentikousis, Eds. Internet Research Task Force, 2015.

[2] Open Networking Foundation (ONF). (2014). *SDN Architecture 1.0*. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR_SDN_ARCH_1.0_06062014.pdf

[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] Open Networking Foundation (ONF). *Sdn Architecture for Transport Networks*. [Online]. Available: https://www.opennetworking.org/wp-content/uploads/2014/10/SDN_Architecture_for_Transport_Networks_TR522.pdf

[5] (2014). *Understanding the SDN Architecture: SDN Control Plane SDN Data Plane*. [Online]. Available: https://www.sdxcentral.com/sdn/definitions/inside-sdn-architecture

[6] J. Matias, J. Garay, N. Toledo, J. Unzilla, and E. Jacob, "Toward an SDN-enabled NFV architecture," *IEEE Commun. Mag.*, vol. 53, no. 4, pp. 187–193, Apr. 2015.

[7] O. S. Brief, "OpenFlow-enabled SDN and network functions virtualization," *Open Netw. Found*, vol. 17, pp. 1–12, Feb. 2014.

[8] N. Operators, "Network functions virtualization, an introduction, benefits, enablers, challenges and call for action," in *Proc. SDN OpenFlow SDN OpenFlow World Congr.*, Oct. 2012, p. 48.

[9] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Commun. Mag.*, vol. 53, no. 2, pp. 90–97, Feb. 2015.

[10] R. Mijumbi *et al.*, "Network function virtualization: State-of-the-art and research challenges," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 236–262, 1st Quart., 2016.

[11] R. Muñoz, *et al.*, "Integrated SDN/NFV management and orchestration architecture for dynamic deployment of virtual SDN control instances for virtual tenant networks [Invited]," *J. Opt. Commun. Netw.*, vol. 7, no. 11, pp. B62–B70, Nov. 2015.

[12] R. Vilalta, A. Mayoral, R. Muñoz, R. Casellas, and R. Martínez, "Multitenant transport networks with SDN/NFV," *J. Lightw. Technol.*, vol. 34, no. 6, pp. 1509–1515, Mar. 15, 2016.

[13] R. Vilalta, *et al.*, "Transport network function virtualization," *J. Lightw. Technol.*, vol. 33, no. 8, pp. 1557–1564, Apr. 15, 2015.

[14] R. Vilalta, R. Muñoz, R. Casellas, R. Martínez, V. López, and D. López, "Transport PCE network function virtualization," in *Proc. Eur. Conf. Opt. Commun. (ECOC)*, Sep. 2014, pp. 1–3.

[15] A. A. Neghabi, N. J. 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.

[16] S. K. Askar, "Adaptive load balancing scheme for data center networks using software defined network," *Sci. J. Univ. Zakho*, vol. 4, no. 2, pp. 275–286, 2016.

[17] J. Yu, Y. Wang, K. Pei, S. Zhang, and J. Li, "A load balancing mechanism for multiple SDN controllers based on load informing strategy," in *Proc. 18th Asia–Pacific Netw. Oper. Manage. Symp. (APNOMS)*, Oct. 2016, pp. 1–4.

[18] Y. Zhou *et al.*, "A load balancing strategy of SDN controller based on distributed decision," in *Proc. 13th Int. Conf. Trust, Secur. Privacy Comput. Commun.*, Sep. 2014, pp. 851–856.

[19] T.-L. Lin, C.-H. Kuo, H.-Y. Chang, W.-K. Chang, and Y.-Y. Lin, "A parameterized wildcard method based on SDN for server load balancing," in *Proc. Int. Conf. Netw. Appl. (NaNA)*, Jul. 2016, pp. 383–386.

[20] A. Dixit, F. Hao, S. Mukherjee, T. V. Lakshman, and R. Kompella, "Towards an elastic distributed SDN controller," *Comput. Commun. Rev.*, vol. 43, no. 4, pp. 7–12, 2013.

[21] Y. Hu, W. Wang, X. Gong, X. Que, and S. Cheng, "BalanceFlow: Controller load balancing for OpenFlow networks," in *Proc. 2nd Int. Conf. Comput. Intell. Syst.*, Nov. 2012, pp. 780–785.

[22] K. Hikichi, T. Soumiya, and A. Yamada, "Dynamic application load balancing in distributed SDN controller," in *Proc. 18th Asia–Pacific Netw. Oper. Manage. Symp. (APNOMS)*, Oct. 2016, pp. 1–6.

[23] *Gs NFV 003-v1. 2.1-Network Function Virtualisation (NFV): Terminology for Main Concepts in NFV*, ETSI, Sophia Antipolis, France, Dec. 2014.

[24] P. Quinn and T. Nadeau, Eds., "Service function chaining problem statement," Internet-Draft, Network Working Group, Feb. 2014.

[25] R. Mijumbi, J. Serrat, and J.-L. Gorricho, "Self-managed resources in network virtualisation environments," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage.*, May 2015, pp. 1099–1106.

[26] S. Ejaz and Z. Iqbal, "Network function virtualization: Challenges and prospects for modernization," in *Proc. Int. Conf. Eng. Emerg. Technol. (ICEET)*, Feb. 2018, pp. 1–5.

[27] *Gs NFV-Man 001 V1. 1.1 Network Function Virtualisation (NFV); Management and Orchestration*, ETSI, Sophia Antipolis, France, 2014.

[28] R. Vilalta *et al.*, "Network virtualization controller for abstraction and control of OpenFlow-enabled multi-tenant multi-technology transport networks," in *Proc. Opt. Fiber Commun. Conf. Exhib. (OFC)*, Los Angeles, CA, USA, Mar. 2015, pp. 1–3.

[29] R. Cziva, S. Jouët, D. Stapleton, F. P. Tso, and D. P. Pezaros, "SDN-based virtual machine management for cloud data centers," *IEEE Trans. Netw. Service Manag.*, vol. 13, no. 2, pp. 212–225, Jun. 2016.

[30] R. Vilalta, A. Mayoral, R. Muñoz, R. Casellas, and R. Martínez, "The SDN/NFV cloud computing platform and transport network of the ADRENALINE testbed," in *Proc. IEEE 1st Conf. Netw. Softwarization (NetSoft)*, Apr. 2015, pp. 1–5.

[31] R. Muñozm *et al.*, "SDN/NFV orchestration for dynamic deployment of virtual SDN controllers as VNF for multi-tenant optical networks," in *Proc. Opt. Fiber Commun. Conf. Exhib. (OFC)*, Mar. 2015, pp. 1–3.

[32] M. Gholami and B. Akbari, "Congestion control using OpenFlow in software defined data center networks," in *Proc. 19th Int. ICIN Conf.-Innov. Clouds, Internet Netw.*, Mar. 2016, pp. 1–3.

[33] (2018). *OpenDaylight Platform Overview*. [Online]. Available: https://www.opendaylight.org/what-we-do/odl-platform-overview

SIKANDAR EJAZ received the B.Sc. degree in telecommunication and networking from COMSATS University Islamabad, Pakistan. He is currently a Post-Graduate Researcher with the Department of Computer Science, University of Engineering and Technology at Taxila, Taxila, Pakistan. His research interests include software-defined networking and network function virtualization.

ZESHAN IQBAL received the M.S. degree in computer engineering from the Center for Advance Studies in Engineering, Islamabad, Pakistan, in 2006, and the Ph.D. degree in computer engineering from the University of Engineering and Technology at Taxila, in 2013, where he is currently an Assistant Professor with the Department of Computer Science. His research interests include software-defined networks, network function virtualization, information centric networks, routing protocols optimization, and wireless body area networks.

PEER AZMAT SHAH received the Ph.D. degree from Universiti Teknologi PETRONAS, Malaysia, in 2014. He is currently Postdoctoral Researcher with the Erik Jonsson School of Engineering and Computer Science, The University of Texas at Dallas, Richardson, TX, USA. He is also an Assistant Professor with the Department of Computer Science, COMSATS University Islamabad, Attock Campus, Pakistan. His research interests include mobility management in wireless networks, the Future Internet, and modeling and optimization of network protocols and algorithms.

BILAL HAIDER BUKHARI received the M.Sc. degree in computer science from Griffth College, Dublin. He is currently pursuing the Ph.D. degree in computer science with Bahria University at Islamabad. He has been an Assistant Professor with COMSATS University Islamabad at Attock Campus (CIIT Attock Campus), Pakistan, since 2013. He also served as Manager in information system with CIIT Attock Campus, until 2017. His research interests include vehicular ad hoc networks, intelligent transportation systems, and software-defined networks.

ARMUGHAN ALI is currently an Assistant Professor with the Computer Science Department, COMSATS University Islamabad at Attock Campus, Attock, Pakistan. He is serving in this entrenched institute for the last ten years. Along with extraordinary pedagogical skills, he also marked his name as one of the leading researchers in the university. His research interests include wireless networks, and optimization of networks using machine learning and artificial intelligence.

FARHAN AADIL received the B.S. degree in computer science from Allama Iqbal Open University, Pakistan, in 2005, and the M.S. and Ph.D. degrees in software engineering and computer engineering from the University of Engineering and Technology at Taxila, Taxila, Pakistan, in 2011 and 2016, respectively. He pursued a career in computer science for four years, from 2005 to 2009. He is currently an Assistant Professor with the Department of Computer Science, COMSATS University Islamabad at Attock Campus. His research interests include vehicular ad hoc networks, machine learning, and evolutionary algorithms.