Journal of King Saud University – Computer and Information Sciences xxx (xxxx) xxx

Contents lists available at ScienceDirect

Journal of King Saud University – Computer and Information Sciences

journal homepage: www.sciencedirect.com

A virtual execution platform for OpenFlow controller using NFV

Bata Krishna Tripathy [a], Kshira Sagar Sahoo [b], Ashish Kr. Luhach [c], N.Z. Jhanjhi [d,*], Swagat Kumar Jena [e]

[a] School of Electrical Sciences, Indian Institute of Technology, Bhubaneswar, India
[b] Department of Information Technology, VNR VJIET, Hyderabad, India
[c] The PNG University of Technology, Papua New Guinea
[d] School of Computer Science and Engineering SCE, Taylor's University, Malaysia
[e] Trident Academy of Technology, Bhubaneswar, India

ARTICLE INFO

Article history:
Received 14 November 2019
Revised 20 February 2020
Accepted 1 March 2020
Available online xxxx

Keywords:
Software defined networking
Network functions
Virtualization
Security

ABSTRACT

The Software Defined Networking (SDN) paradigm decouples the network control functions from the data plane and offers a set of software components for flexible and controlled management of networks. SDN has promised to provide numerous benefits in terms of on-demand provisioning, automated load balancing, streamlining physical infrastructure, and flexibility in scaling network resources. In order to realize these network service offerings, there is an important need for developing an efficient, robust, and secure execution platform. As a primary contribution, we present a novel virtual execution platform for the OpenFlow controller using Network Function Virtualization (NFV). Theoretically, NFV can apply to any network function, which can simplify the managing of the heterogeneous data plane. The characteristics of our proposed architecture include pipe-lined processing of network traffic, virtualized and replicated execution of network functions, isolation between task nodes, and random mapping of traffic to task nodes. The proposed architecture has two major components: a Network Packet Schedulers (NPS) and a Task Engine (TE). The TE consists of Task Nodes (TNs) which are responsible for executing different network functions on various traffic flows and each TN is realized as a virtual machine. Upon receiving traffic from the data plane, NPS analyses the functional requirements of the traffic and different controller performance parameters. Then it allocates the traffic to appropriate TNs for executing necessary network functions. In this respect, it provides performance benefits, robustness, fine-grained modularity, and strong isolation security in the processing of traffic flows on the SDN platform. Efficacy of our proposed architecture has been demonstrated with a case study.

1. Introduction

The growth of the Internet has transformed the world to a digital society where the goal is to connect everything and access from anywhere (Wood, 2015; Blenk, 2016; Sahoo et al., 2019). In a traditional IP network, controlling and managing large network systems with a vertically integrated set of network control functions becoming increasingly complex. In this regard, Software Defined Networking (SDN) and OpenFlow are the emerging paradigms that promise to change the state of the art of networking, where the network control logic is isolated from its underlying hardware (Sahoo et al., 2016; Nishtha et al., 2014; Sahoo et al., 2019). It offers a set of software components for flexible and controlled management of networks where network functions may span from traffic forwarding and flow control monitoring to security enforcement (Li and Yong, 2015; Karakus and Murat, 2017). The significant advantages of SDN include the ability to introduce network innovations faster and to radically simplify and automate the management of large networks with on-demand provisioning (Ghodsi et al., 2011; Ferrús et al., 2016).

Despite several advantages, there are various challenges involved in SDN, such as managing network scalability and security, etc. Different challenges arise due to the lack of standardized execution platform, control-data plane interfaces, and security enforcement functions (Jammal, 2014). Metzler (2012) reported that, when the network scales up, the SDN controller can become

* Corresponding author.
E-mail addresses: bt10@iitbbs.ac.in (B.K. Tripathy), ashishluhach@acm.org (A.Kr. Luhach), noorzaman.jhanjhi@taylors.edu.my (N.Z. Jhanjhi), swagatkumar.jena@tat.ac.in (S.K. Jena).

Peer review under responsibility of King Saud University.

Production and hosting by Elsevier

https://doi.org/10.1016/j.jksuci.2020.03.001
1319-1578/

Please cite this article as: B. K. Tripathy, K. S. Sahoo, A. K. Luhach et al., A virtual execution platform for OpenFlow controller using NFV, Journal of King Saud University – Computer and Information Sciences, https://doi.org/10.1016/j.jksuci.2020.03.001

a key bottleneck. When the number of switches and traffic flows increases, more requests are queued up at the controller. In turn, it increases the response time of the requested events. Besides, OpenFlow based APIs and the centralized management of these functions can introduce various security threats to the underlying network. These challenges demand a seamless, and efficient execution platform for the SDN control plane to serve different network functions and preventing possibilities of various security threats (Martins et al., 2014; Koponen et al., 2014; Sahoo, 2019).

In this paper, we present a novel virtual execution platform for the SDN control plane that uses the concept of NFV (Zahoor and Naaz Mir, 2018). NFV is about wide range of network functions, which is driven primarily by new network requirements. It put back network services issued by dedicated hardware devices with virtualized software. The network services which were required specialized hardware, with the NFV technology, they can run over standard commodity servers,in turn it reduces the overall cost (Almusaylim and Zaman, 2019).

The proposed has two major components,*i.e.* Task Engine (TE) and Network Packet Schedulers (NPS). The TEs are responsible for executing different network functions on various traffic flows. These TEs maintain a data structure to record several instances of different functions that are being served by it. The NPS analyses the functional requirements of the traffic and different controller performance parameters. Further NPS allocates the traffic to appropriate Task Nodes (TN) for executing necessary network functions. The TE consists of Task Nodes. The important characteristics of this architecture include pipe-lined processing of network traffic, virtualized and replicated execution of network functions, and random mapping of traffic to TNs (Tripathy et al., 2016).

The rest of the paper is organized as follows. Section 2 presents the proposed SDN controller platform. We described the control flow of traffic on the proposed platform in Section 3. Section 4 presents the performance enhancement of the proposed architecture. The evaluation of the proposed architecture has demonstrated with a case study in Section 5 followed by the concluding remarks in Section 6.

2. Proposed SDN controller execution platform architecture

This section presents the architecture of the proposed SDN controller platform. An abstract view of the overall architecture has illustrated in Fig. 1. Before illustrating the proposed execution platform, let us discuss briefly on the entities involved in the SDN controller. There are two major entities: (i) user network applications and (ii) network functions. A user can request a specific network service with various requirements. Such service requests are named as user network applications. For example, an employee of an enterprise network may want to use a VPN service to connect to the R&D subnet of the organization with high bandwidth, minimum delay, and appropriate security enforcement. A network function, on the contrary, is a network control operation or task to be executed corresponding to a user application. A sequence or a list of network functions are often required to execute for serving a network function. We consider six possible network functions namely (i) routing control, (ii) flow monitoring, (iii) policy checking, (iv) bandwidth guarantee, (v) security enforcement, and (vi) load balancing. The architecture of the proposed execution platform for SDN controller is shown in Fig. 2. It consists of two components, namely Network Packet Scheduler and Task Engine. The Network Packet Scheduler is called NPS and Packet Schedulers (PS) are its component. The Task Engine is also known as TE and Task Nodes (TN) are the components of TE. These nodes are responsible for executing different network functions on various traffic flows whereas the packet scheduler intelligently allocates the traffic to appropriate nodes. The detailed functionalities of these components are described below, while the abbreviations/notations used for different parameters are presented in Table 1. (See Fig. 3).

2.1. Task nodes and their implementation

A TN serves as an execution unit for running a set of network functions. Each TN is responsible for executing a finite number (>1) of network functions limited by a threshold (d). The parameter d indicates the maximum degree of parallelism which may vary with the type of network functions running on it and the application context. For the purpose of simplicity, we consider a predefined d associated to each task node. In a node, a network function (NF) is associated with a unique tag that is used to reference that function in the corresponding node. A network function can be served by multiple task nodes.

Each TN is realized as a virtual machine with a predefined and fixed set of computing and storage resources. A TN has an input queue (first-in, first out) that keeps track of the packet processing request received from NPS. Packet processing request consists of a packet header and a list containing information about Task Nodes (TNs). Details about the list can be found in a later section. In order

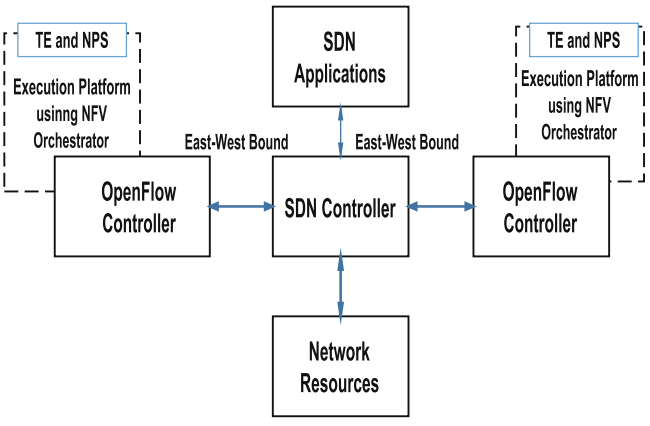

Fig. 1. An abstract view of the overall Architecture.

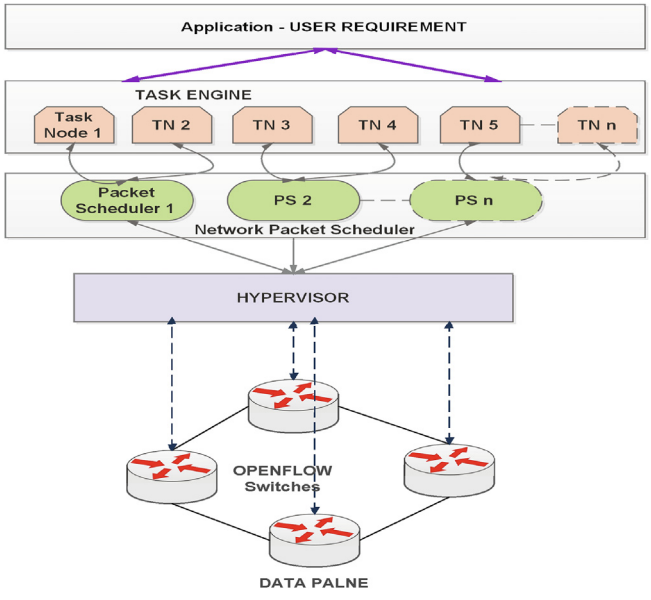

Fig. 2. Proposed virtual Execution platform for SDN controller.

Please cite this article as: B. K. Tripathy, K. S. Sahoo, A. K. Luhach et al., A virtual execution platform for OpenFlow controller using NFV, Journal of King Saud University – Computer and Information Sciences, https://doi.org/10.1016/j.jksuci.2020.03.001

Table 1
Abbreviations and Symbols used in the paper.

Abbreviation/Notation	Meaning
TN	Task Node
NF	Network Function
PS	Packet Scheduler
TN_i	i^{th} Task Node
PS_i	i^{th} Packet Scheduler
NF_i	i^{th} Network Function
$NF_{i,k}$	Tag of NF_i in TNk
$SEQ(NF_i)$	Sequence OF NF
NFAT	NF Allocation Table
TNAT	TN Status Table
TN_i^p	Current Queue Size In TN_i
$TN_{i,st}$	Current State Of TN_i
NF_i^t	Avg. Completion Time For NF_i

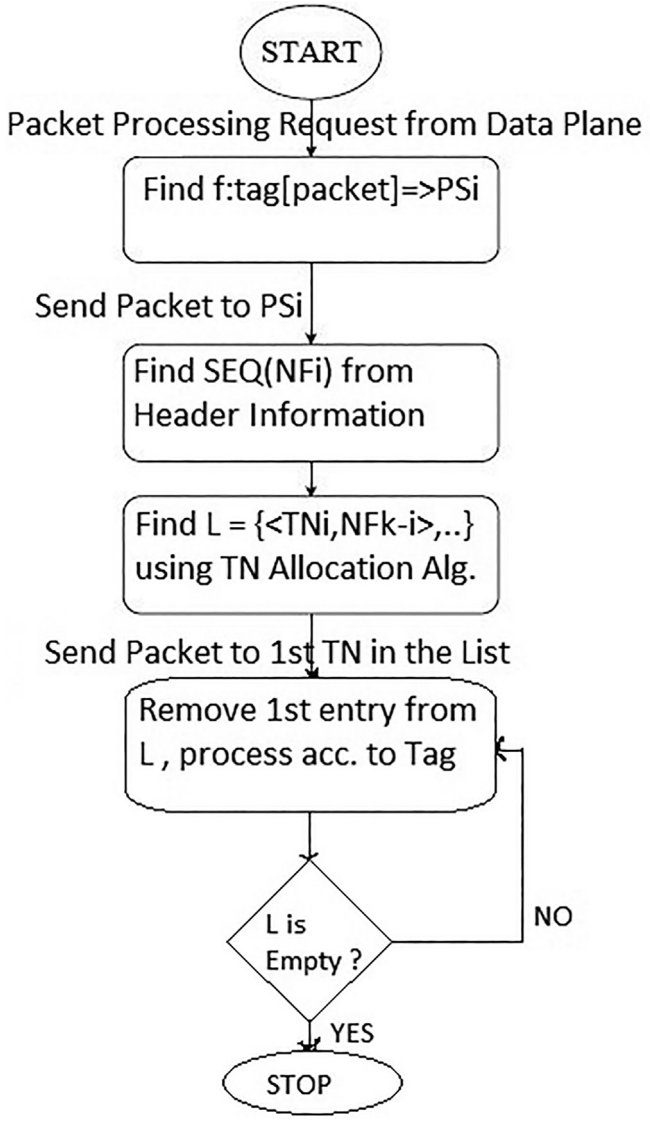

Fig. 3. Flow of a packet through this Controller.

to serve a packet, the TN takes a packet from the queue, reads the tag of the packet, matches this tag with its NF tags, and executes the function corresponding to the matched tag on the packet. Upon successful completion of the network function, the TN updates the status of the packet. Further, it sends an acknowledgment to NPS (i.e., the corresponding PS) and sends the packet to the next TN on the list. Inter-TN communication is accomplished through the message passing.

A TN can be loaded with new NFs or updated on existing NFs depending on the requirements. This configuration is governed and synchronized by the PS. During this configuration period, the environment of the corresponding TN is saved and the state is set to the BLOCKED state. The saving state consists of the NFs it is assigned with, it's packet processing status, queue content etc. Once the configuring process is over the TN becomes active (the ACTIVE state). It then resumes execution from the saved environment. If a TN crashes due to some error then the TN is suspended with an error message sent to the associated PS.

2.2. Packet schedulers

A packet scheduler keeps track of the information about the TNs. This information includes the state of each TN, network functions running on these TNs along with their tags, and processing statistics (average packet processing time, failure rate, current queue size, etc.). A network packet scheduler has two main tasks, namely NF-sequence determination and TN allocation:

- **NF-Sequence Determination:** The PS extracts information from the packet header that includes source and destination IP addresses, source and destination ports, etc. Then, it determines a list of NFs and their sequence that are required to be executed to process the packet request. This NF list is created using filtering criteria based on the extracted fields and criteria are provided by different applications. A packet, specific to an application must be filtered through IP subnet or port number required by the corresponding application. A packet should be provided with all the NFs in the sequence for successful execution of the application.
- **TN Allocation:**PS initializes a list L to empty.This list will contain entries of 2-tuple $< TN_k, NF_{i,k} >$. It iterates through each entry of the NF-sequence(generated in the previous step).For each *i*'th entry say NF_i , it adds a tuple $< TN_k, NF_{i,k} >$ to the list L.Here, TN_k is the *k*'th active TN and $NF_{i,k}$ is the tag of NF_i associated with TN_k. The tuple creation process is discussed in detail in Algorithm 1. It requires two data structures:
  - Network Function Allocation Table (NFAT)
  - Task Node Status Table (TNST)

A detailed description of these structures are described in the following section.

After updating the list by iterating through the NF sequence, the packet scheduler sends the packet to the first TN in the sequence. The TN sends an acknowledgment to the corresponding PS after successfully execution of corresponding network function in the TN. The PS updates the statistics of the TN. There are often more than one PS, each of which is responsible for scheduling packet requests of data plane switches connected to it. When a packet arrives at the controller, it is forwarded to corresponding PS based on the tag associated with the packet. As multiple schedulers work simultaneously, it allows serving multiple requests simultaneously and, thus, providing fault-tolerance as explained later in this paper.

In addition, a PS controls the load of new network functions and updates existing network functions to a TN depending on the change in the context or user requirements. In such cases, the packet scheduler decodes the requirements, finds the list of network functions to serve the requirements, and accordingly updates the filtering criteria and network functions (if necessary). All the packet schedulers communicate with each other and maintain a consistent state of the system. The changes made to any TN or

Please cite this article as: B. K. Tripathy, K. S. Sahoo, A. K. Luhach et al., A virtual execution platform for OpenFlow controller using NFV, Journal of King Saud University – Computer and Information Sciences, https://doi.org/10.1016/j.jksuci.2020.03.001

any update sent by a TN, have reflected to all schedulers. The next section presents the control flow of a packet through the proposed controller execution architecture.

3. Control flow of a packet in the proposed platform

This section presents a detailed description of how a packet flows through our proposed controller execution platform. Fig. 2 shows the flow chart depicting the control flow. The control flow consists of a sequence of four steps as described below.

Step 1. A new packet is mapped to appropriate packet scheduler

When a packet arrives at a data plane switch, if there is no match found with the flow-rule of the corresponding switch, the packet is sent to the controller. The tag-bit associated to the packet determines the packet scheduler to which the processing is to be routed. This is realized through a simple modular hash-based mapping function on some fields of the packet header.

f: tag[packet] $\Rightarrow PS_i$.

Step 2. Packet Scheduler de-queues a packet from its queue and allocates appropriate Task node.

This step consists of following tasks. **[subtask-a]:** Extract $< PORT_{src}, PORT_{dstn}, IP_{src}, IP_{dstn} >$ from packet header. Let, this tuple be H. Create a sequence of network functions, $SEQ(NF_i)$ based on H and available information about application requirements.

[subtask-b]: Initialize List $L = \varnothing$. Iterate through $SEQ(NF_i)$. For each NF_i, find an active Task node TN_k and corresponding tag $NF_{i,k}$. Add $< TN_k, NF_{i,k} >$ to L.

$NF_{i,k}$ signifies NF_i is running on TN_k with tag $NF_{i,k}$.

[subtask-c]: Send the packet along with the sequence to the first task node (TN) in the list.

Step 3. Execution of Network Function in Task Node. Upon receiving a packet, the corresponding task node removes an entry from the list, reads tag and runs corresponding network function. The TN sends an acknowledgment to PS after successful completion of network function, and forwards the packet to next TN in the list.

Step 4. Iterate step 3 The Step 3 is iterated until all the network functions are executed on the packet.

3.1. Data structure for implementing control flow

For realizing the step **2(b)** of the above control flow operations, following two data-structures are required.

1. **Network Function Allocation Table(NFAT):** This table records the availability of executing network functions to the Task Nodes. An NF might be available at more than one Task Node. This table is indexed by NF number and one entry of NFAT is of the following form:
   $< NF_i, \ll TN_j, TN_j^p >, .., < TN_k, TN_k^p \gg$ The queue size of a TN is increased by one after allocation of a packet to the task node (TN) for execution of a network function. It is decreased by one after receiving an acknowledgement about completion of the related network functions from any TN.
2. **Task Node Status Table(TNST):** This table stores the current state of a TN, list of Network Function available in it along with their tag number. The table is indexed by TN number. An entry in TNST is of the form:$< TN_i, TN_{i,st}, < 1, NF_i, NF_i^t >, ., < k, NF_p, NF_p^t \gg$

The detailed algorithm of Task Node Allocation to Packet requests is presented in Algorithm 1.

Algorithm 1 Task Node Allocation to Packet Requests

```
1: procedure FINDTASKNODE SEQ(NF_i)
2:   Initialize List L to empty
3:   foreach NF_i in SEQ(NF_i) do
4:     TN^p_min = INT-MAX
5:     Initialize TN_min to empty
6:     foreach < TN_j, TN^p_j > in NFAT[NF_i] do
7:       if TNST[TN_j].state = ACTIVE then
8:         if TN^p_j < TN^p_min then
9:       TN^p_min = TN^p_j
10:          Set TN_min to empty
11:          Append TN_j to TN_min
12:      else if TN^p_j = TN^p_min then
13:        Append TN_j to TN_min
14:      end if
15:    end if
16:  end for
17:  [TN_opt, NF_tag]= FOTN(TN_min, NF_i)
     //FOTN: FindOptTaskNode
18:    Append < TN_opt, NF_tag > to L
19:  end for
20:  Update queue entry for TN_opt
21:  Return L
22: end procedur
23: procedure FINDOPTTASKNODE TN_min, NF_i
24:  TN^t_min=INT-MAX
25:  Initialize TN_opt to empty
26:  NF_tag = 0
27:  foreach < TN_j > in TN_min do
28:    if TNST[TN_j, NF_i].time < TH^t_min then
29:        TN^t_min=TNST[TN_j, NF_i].time
30:      TN_tag = TNST[TN_j, NF_i].tag
31:      TN_opt = TH_j
32:    end if
33:  end for
34:  Return [TN_opt, NF_tag]
35: end procedure
```

The following assumption has been made on the execution of network functions in the proposed Execution Platform.

- At any time instant, for each network function(NF), at least one TN is active for executing the function.
- Degree of Parallelism (DoP) is the maximum number of packets waiting in the queue of a TN. The DoP of a task node (TN) is enough to serve all the packet allocated to a TN at any instance of time.

The next section presents the performance enhancement characteristics of our proposed architecture.

4. Performance enhancement characteristics

Our proposed controller execution platform architecture has the following characteristics that improve the performance and strengthens the security of SDN controller and the underlying network.

4.1. Pipelined processing

A large no of packets can be processed in pipelined fashion as the TNs provide different and independent services. A task node

Please cite this article as: B. K. Tripathy, K. S. Sahoo, A. K. Luhach et al., A virtual execution platform for OpenFlow controller using NFV, Journal of King Saud University – Computer and Information Sciences, https://doi.org/10.1016/j.jksuci.2020.03.001

writes an update after processing a packet and schedules it to next task node in the list. This improves the throughput of the system.

4.2. Network Function replication at task nodes

The Network Functions provided in the controller are replicated at different TNs, i.e., the same function is implemented in more than one TN. All the TNs running one service are in sync with each other and the packet scheduler is to maintain a consistent state for that Network Functions. This provides more than one path to process a flow, thus it can handle multiple similar flows simultaneously. The packet scheduler maintains a flag to keep track of any modification to the network functions in both network function allocation table and task node status table. Hence, if any change in the flag is detected, it triggers the modification process in all the replicated copies of the network functions to maintain synchronization among them.

4.3. Intelligent scheduling of Network Functions

This is realized based on the service replication at different TNs. While allocating a TN for an NF, we have considered the following assumptions.

- Select the TN which has minimum queue length. This ensures less queuing delay.
- If there are multiple TN with same queue length, select one which has less average execution time for that NF. Thus, it ensures a fast response time.

Combining the above rules we have realized automatic load balancing among the TNs. As task nodes are allocated with least queue length, there is no possibility that a TN gets to its maximum queue length while others at less queue length. This ensures the finiteness of DoP. Task Node Allocation algorithm has two procedures. The FindTaskNode procedure finds TN with minimum queue length. If there is more than one TNs with same queue size, FindOptTaskNode procedure finds the TN which takes minimum average time to complete the corresponding NF.

4.4. Random allocation of packets to task nodes

For a similar type of traffic, our proposed algorithm allocates the packets to different TNs to serve the same Network Section. This introduces randomness in the allocation of TNs to packets which makes it difficult for the attacker to crack the execution signature of a packet. This ensures stronger security.

5. Evaluation of proposed controller execution platform

This section presents the performance evaluation of our proposed controller execution platform with a case study. Fig. 4 shows an instance of an enterprise network with four subnets corresponding to different departments.

The list of requirements on various network traffic is stated as follows:

Req_1**:** All outbound traffic from CR should go through the proxy server.

Req_2**:** All traffic from employees working outside should be served through VPN to RD and Sales, with high Bandwidth.

Req_3**:** Any incoming traffic to RD and Finance should be forwarded through policy check and security compliance.

Req_4**:** Any outbound traffic from RD and Finance should guarantee high bandwidth and availability.

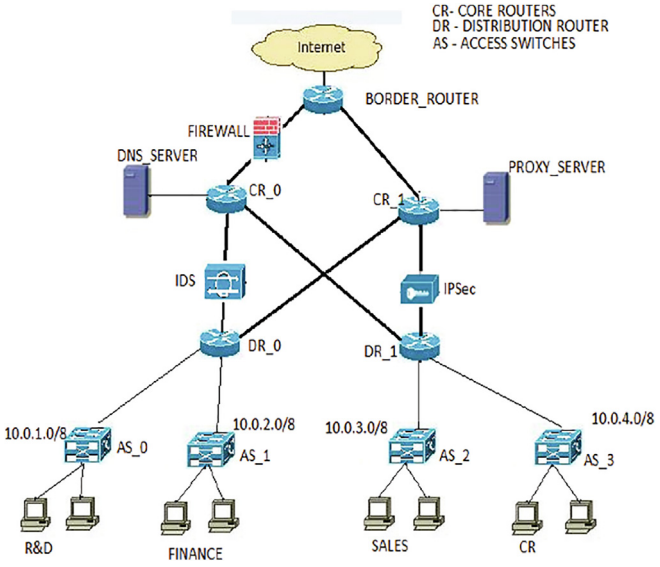

Fig. 4. A segment of an Enterprise network.

Req_5**:** Customers should be able to communicate to CR through a secure channel. Let, they connect through port number 8091.

Req_6**:** Internal flooding should be stopped as soon as possible.

We consider the following Network Function (NF) implementations to serve different traffic with above-mentioned requirements.

- Routing(R)
- Flow Monitoring(FM)
- Policy Checking(PC)
- Bandwidth Guarantee(BW)
- Security Enforcement(SE)
- Load Balancing(LB)

Considering these requirements, we have generated different traffic with varying traffic rates and evaluated the performance of our architecture in Mininet simulation platform (Mininet Documentation, 2020). The following are the NF implementations corresponding to the traffic with various requirements.

NF implementation (for traffic with Req_1)

- DR_1:- If IP_{src} is from 10.0.4.0/8 and IP_{dstn} is out of the autonomous system, send packet to CR_1. (R, PC)
- CR_1:- If IP_{src} is from 10.0.4.0/8 and IP_{dstn} is out of the autonomous system, send packet to Proxy server. (R, PC)

NF implementation (for traffic with Req_2)

- *BorderRouter*:- If IP_{src} is from outside and IP_{dstn} belongs to 10.0.1.0/8 or 10.0.3.0/8 and VPN is connected(Can be checked through port number) send packet to CR_0(in case 10.0.1.0/8) or CR_1(in case 10.0.3.0/8), then bypass firewall and reserve required bandwidth (R, PC, BW).
- CR_0:- If IP_{src} is from outside and IP_{dstn} is from 10.0.1.0/8, then send packet to DR_0, bypass IDS and reserve required bandwidth(R, PC, BW).

NF implementation (for traffic with Req_3)

- *BorderRouter*:- If IP_{dstn} belongs to 10.0.1.0/8 or 10.0.2.0/8, then send packet to CR_0 through firewall(R, PC, SE).
- CR_0:- If IP_{dstn} belongs to 10.0.1.0/8 or 10.0.2.0/8, then send packet to DR_0 through IDS (R, PC, SE).

Please cite this article as: B. K. Tripathy, K. S. Sahoo, A. K. Luhach et al., A virtual execution platform for OpenFlow controller using NFV, Journal of King Saud University – Computer and Information Sciences, https://doi.org/10.1016/j.jksuci.2020.03.001

NF implementation (for traffic with Req_4)

- DR_1:- If IP_{src} is from 10.0.1.0/8 or 10.0.2.0/8 and IP_{dstn} is out of the autonomous system, then send packet to CR_1 or CR_0 bypassing IDS. Reserve required bandwidth for this flow (R, PC, LB, BW).
- CR_0:- If IP_{src} is from 10.0.1.0/8 or 10.0.2.0/8 and IP_{dstn} is out of the autonomous system, then send packet to *BorderRouter*, bypass firewall and reserve required bandwidth (R, PC, BW).

NF implementation (for traffic with Req_5)

- CR_1:- If IP_{src} is from out of the autonomous system and $PORT_{dstn}$=8091, then send packet to DR_1 through IPSec (R, PC, SE).

NF implementation (for traffic with Req_6)

- *ANYROUTER*:- If IP_{src} is from the autonomous system and flow statistics shows anomaly, drop the flow and check origin of flooding (R, FM, SE).

For accessing the system throughput, we have executed and analyzed five different traffic at the same time. The controller execution platform used in the analysis consists of two Packet Schedulers (PS), four Task Nodes (TN). Each having room for four Network Functions(NF) which as shown in the Fig. 5. The used abbreviations, e.g., R, FM, and PC for different network functions have described earlier. An execution analysis of this traffic is presented below.

1. *Traffic 1*: 10.0.1.0/8 to 10.0.2.0/8: NF required :: (R)
2. *Traffic 2*: 10.0.4.0/8 to out of autonomous system NF required :: (R, PC)
3. *Traffic 3*: out-of-autonomous system to 10.0.4.0/8 over port 8091(secured channel for customer) NF required :: (R, PC, SE)
4. *Traffic 4*: from employee working outside to 10.0.1.0/8 through VPN NF required :: (R, PC, BW)
5. *Traffic 5*: 10.0.1.0/8 to out-of-autonomous system NF required :: (R, PC, LB, BW)

The Fig. 6 shows a schedule of network functions related to above traffic to different task nodes with respect to timeline. It is assumed that each NF takes the same amount of time for execution. The table shows that the total execution time of the above traffic is 6 unit of time. On the other hand, the sequential execution

Time	1	2	3	4	5	6	7	8	9
TN1	1 R	4 R	4 P	4 B	--	--	--	--	--
TN2	2 R	2 P	5 R	5 P	--	5 B	--	--	--
TN3	3 R	3 P	3 S	--	5 L	--	--	--	--
TN4	--	--	--	--	--	--	--	--	--

Fig. 6. Gantt Chart of the scheduling of the network functions.

of this traffic would have required 13 units of time. Therefore, the improvement in throughput of our proposed execution platform is 13/6 = 2.16 times. To process this traffic, a sequential execution would take 13 time units, whereas, our proposed network function execution platform takes no more than 6 time units, leading to 2.16 times performance improvement. In addition, our platform ensures fault tolerance during failure of any task node as the same function can be executed in other task nodes. It also allocates the same type of traffic (in sequence) to different task nodes based on availability, which introduces randomness in traffic allocation and thereby, strengthens the security of the SDN control plane and the underlying network.

According to the literature (White paper, 2012), the deployment of NFV approaches in an application domain may significantly affect the overall performance of the system. Usually, the overall performance degrades in terms of latency. It is due to the implementation of industry-standard hardware for supporting speedup

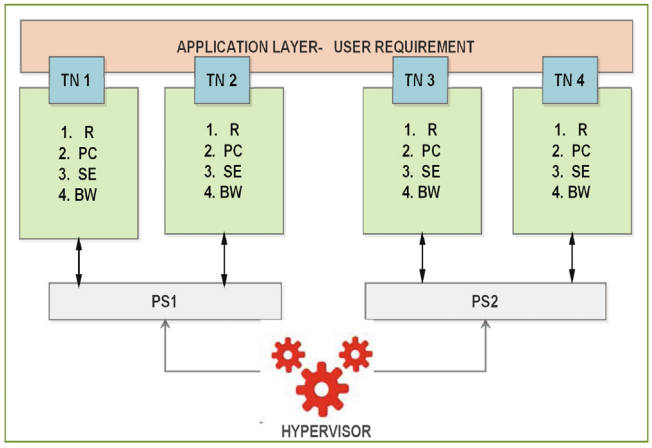

Fig. 5. Controller organization for the case study.

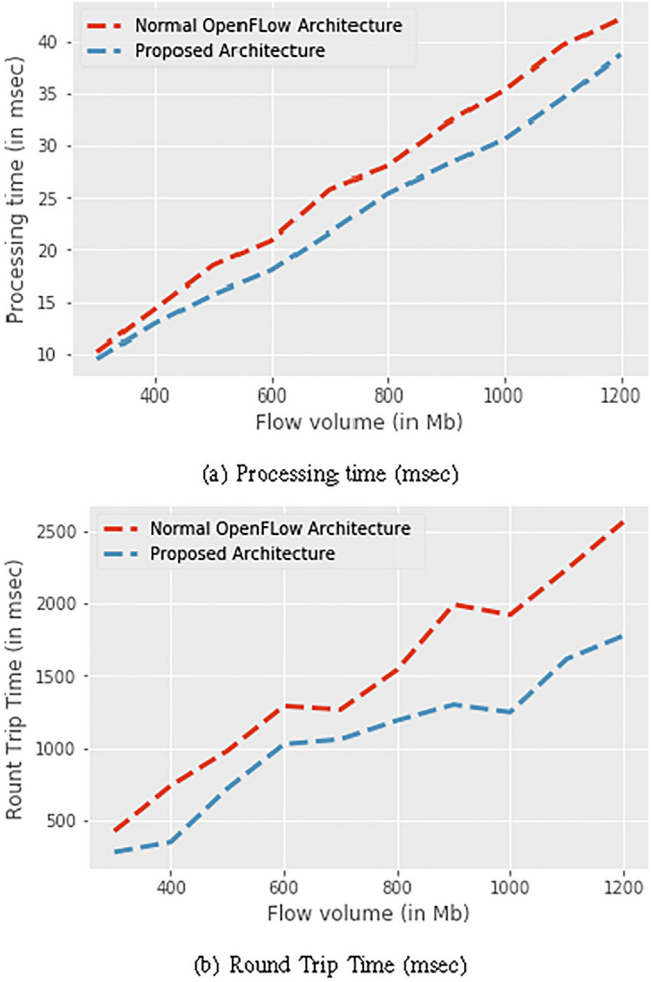

Fig. 7. Average processing time and RTT.

Please cite this article as: B. K. Tripathy, K. S. Sahoo, A. K. Luhach et al., A virtual execution platform for OpenFlow controller using NFV, Journal of King Saud University – Computer and Information Sciences, https://doi.org/10.1016/j.jksuci.2020.03.001

in processing. The latency of a network is measured as Round Trip Time (RTT). It is defined as the time taken by a packet to be transmitted from a source to destination and receiving a response from destination to source. For experimentation, Floodlight OpenFlow controller on mininet platform has been used (OpenFLow, 2020). The Floodlight controller is chosen due to its efficient management of the network control functions (Morales et al., 2015). Further, heterogeneous traffics has generated such as HTTP, FTP, imix, etc. with varying requirements using an OTCL script. The flow volume of different traffic is varied from 300 Mbps to 1200 Mbps to verify the efficacy of our proposed network functions. Hence, we compare the RTT of our proposed architecture in SDN with the standard OpenFlow Floodlight controller platform.

Fig. 7a shows the processing time of our architecture and typical OpenFlow architecture with respect to flow volume (in Mb). From the experiment, it is observed that the processing time of packets in our proposed architecture is less than that of typical OpenFlow architecture. It is due to the pipeline processing of packets instead of sequential processing. Fig. 7b shows the RTT of the proposed architecture and standard OpenFlow architecture concerning flow volume. It describes that the RTT in the proposed architecture is stable as compared to standard OpenFlow architecture. It is because of the implementation of fine-grained modularity in packet allocation and pipeline processing in the proposed solution. In another experiment, various Quality of Service (QoS) parameters such as average packet loss and jitter have been estimated. The jitter is interpreted as the difference in the latency of a flow between two hosts. It happens due to a variety of reasons, such as network congestion and changing routes. Fig. 8a and Fig. 8b depicts the packet loss and jitter during the experiment, respectively. The average packet loss and jitter value of the proposed scheme are lower than the normal OpenFlow architecture.

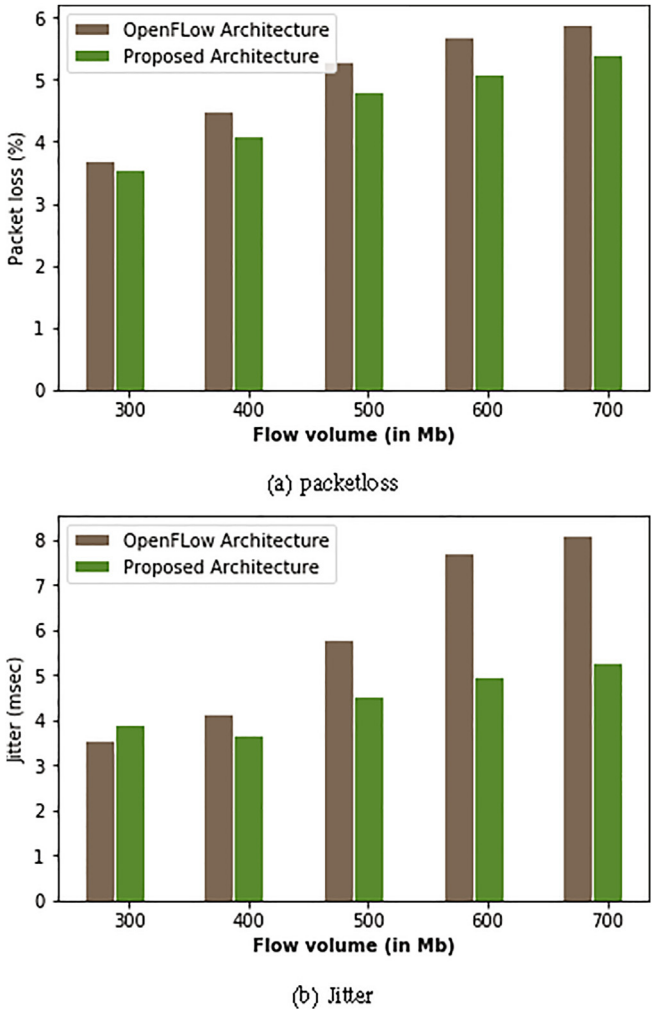

Fig. 8. Impact of various QoS parameters.

6. Conclusion

The Software Defined Networking decouples the network control functions from the data plane and offers a set of software components for flexible and controlled management of networks. In order to realize the network service offerings, we present a novel virtual execution platform for the SDN controller that provides performance benefits along with fine-grained modularity and strong isolation. Two major components, such as Network Packet Schedulers (NPS) and Task Engine (TE) are responsible for executing different network functions on various traffic flows. Additionally, NPS analyses the functional requirements of the traffic and the controller performance parameters; in turn, it allocates the traffic to appropriate task nodes for executing necessary network functions. Finally, the proposed architecture has been evaluated with a case study, and the pipelined processing of network traffic improves the throughput 2.16 times more compared to sequential processing. Besides, the architecture also improves the average processing time and RTT compared to the traditional OpenFlow architecture. In the future scope of the work, the framework will be tested with a large number of heterogeneous traffics in varying requirements and contexts.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Almusaylim, Zahrah A., Zaman, Noor, 2019. A review on smart home present state and challenges: linked to context-awareness internet of things (IoT). Wireless Netw. 25 (6), 3193–3204.

Blenk, Andreas et al., 2016. Survey on network virtualization hypervisors for software defined networking. IEEE Commun. Surveys Tutor. 18 (1), 655–685.

Ferrús, Ramon et al., 2016. SDN/NFV-enabled satellite communications networks: opportunities, scenarios and challenges. Phys. Commun. 18, 95–112.

Ghodsi, et al., 2011. Intelligent design enables architectural evolution. In: Proceedings of 10th ACM Workshop Hot Topics in Networks, Article No. 3. pp. 1–6..

Jammal, M. et al., 2014. Software defined networking: state of the art and research challenges. Comput. Networks Elsevier 72, 74–98.

Karakus, Murat, Durresi, Arjan, 2017. Quality of service (QoS) in software defined networking (SDN): a survey. J. Network Comput. Appl. 80, 200–218.

Koponen, Teemu, et al., 2014. Network virtualization in multi-tenant datacenters. In: 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)..

Li, Yong, Chen, Min, 2015. Software-defined network function virtualization: a survey. IEEE Access 3, 2542–2553.

Martins, Joao, et al., 2014. ClickOS and the art of network function virtualization. In: 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)..

Metzler, 2012. Research: Understanding Software-Defined Networks, Information Week Reports. pp. 1–25. [Online] Available:http://reports.informationweek.com/abstract/6/9044/Data-Center/research-understanding-software-defined-networks.html..

Mininet Documentation, GitHub. [Online]. Available:https://github.com/mininet/mininet/wiki/Documentation..

Morales, Laura Victoria, Murillo, Andres Felipe, Rueda, Sandra Julieta, 2015. Extending the floodlight controller. In: 2015 IEEE 14th International Symposium on Network Computing and Applications. IEEE.

Nishtha, M. Sood, 2014. Software defined network-Architectures. In: Proceedings of 2014 International Conference on Parallel, Distributed and Grid Computing (PDGC). pp. 451–456..

OpenFLow specification [Online]. Available:http://archive.OpenFlow.org/..

Please cite this article as: B. K. Tripathy, K. S. Sahoo, A. K. Luhach et al., A virtual execution platform for OpenFlow controller using NFV, Journal of King Saud University – Computer and Information Sciences, https://doi.org/10.1016/j.jksuci.2020.03.001

Sahoo, Kshira Sagar et al., 2016. A comprehensive tutorial on software defined network: The driving force for the future internet technology. In: Proceedings of the International Conference on Advances in Information Communication Technology & Computing. ACM.

Sahoo, Kshira Sagar et al., 2019. ESMLB efficient switch migration-based load balancing for multi-controller SDN in IoT. IEEE Internet Things J.

Sahoo, Kshira Sagar, et al., 2019. Toward secure software-defined networks against distributed denial of service attack. J. Supercomput. 1–46..

Sahoo, Kshira Sagar et al., 2019. Improving end-users utility in software-defined wide area network systems. IEEE Trans. Netw. Serv. Manage.

Tripathy, Bata Krishna, et al., 2016. A novel secure and efficient policy management framework for software defined network. In: 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC). vol. 2. IEEE..

White paper, 2012 [Online]. Available:http://portal.etsi.org/NFV/NFVWhitePaper.pdf..

Wood, Timothy et al., 2015. Toward a software-based network: integrating software defined networking and network function virtualization. IEEE Network 29 (3), 36–41.

Zahoor, Saniya, Naaz Mir, Roohie, 2018. Resource management in pervasive Internet of Things: a survey. J. King Saud Univ.-Comput. Inf. Sci.

Please cite this article as: B. K. Tripathy, K. S. Sahoo, A. K. Luhach et al., A virtual execution platform for OpenFlow controller using NFV, Journal of King Saud University – Computer and Information Sciences, https://doi.org/10.1016/j.jksuci.2020.03.001