Intent-Based Slicing between Containers in SDN Overlay Network

Adeel Rafiq, Asif Mehmood, and Wang-Cheol Song
Computer Engineering, Jeju, South Korea
Email: {adeel.rafiq asif, philo}@jejunu.ac.kr

Abstract**—**Container-based service orchestration is getting famous increasingly because of its ability to be deployed quickly and it is cheaper as well as more reliable when compared to virtual machines. Leveraging the container characteristic, deployment of 5G modules as a group of containers on different nodes is an excellent solution to construct a 5g system. In a 5G system, network slicing plays a vital role to isolate the on-demand services and the SDN based overlay network consisting of containers provides the platform which manages to slice the network using intent-based networking. As the number of containers in a deployment increases on hosts across the overlay network with the passage of time in the 5G network, therefore it is necessary to manage them with the tool in a straight way with high-level abstraction. In this paper, we present the deployment of an overlay network which consists of Kubernetes nodes and Open vSwitch. In this paper, we also present the intent-based slicing system using Software Defined Network controller and an Intent-Based Networking (IBN) Manager among the containers across the Kubernetes pods in an overlay network.

Index Terms**—**Containers, kubernetes, software-defined network, slicing, intent-based networking, overlay networking

I. Introduction

Fifth-generation (5G) networks are the driving force behind the growth and enhancement of IoT applications and its evolution is increasing day by day due to its high utility in IoT applications [1]. A report of International Data Corporation (IDC) mentions that around $1.2 billion will be spent on activity management solutions which will be solely derived by global 5G services and it will cover 70% of global market companies [1]. Owing to the need for implementing new business models in future IoT and cater to the demanding requirements of a new application, the latest performance standards are yet to be met. These standards include new criteria such as ultra-reliable, trustworthy, massive connectivity, wireless communication’s coverage, security, drastic low latency, throughput for multiple types and a massive number of IoT devices [2]. To fulfill all these needs, new connectivity interfaces are required. For upcoming IoT applications, the required connectivity demands can be fulfilled by Long-Term Evolution (LTE) and 5G technologies.

It is very important and favorable to design the latest 5G systems that are more scalable, flexible and elastic, exploiting advanced virtualization techniques and development in Network function virtualization (NFV) in order to cope with the boost and development in cloud computing and their support of virtualized services. NFV has got an upper hand in future 5G architecture development as it provides an immense number of leads for network flexibility, scalability, and elasticity. Owing to these advantages NFV has apparently become a prime element in the development of future 5G systems. Federated networked cloud provides fragmented physical infrastructures that have been used by researchers in both industry and academia to develop new architectures that are able to operate an end to end platform of a virtual network and also capable of elasticity composition [3]. The carrier cloud is expected to play a vital part in the design of 5G wireless networks. Software-Defined Networking (SDN) techniques seem like one of the prime initiators for this visual perception of carrier cloud. SDN has introduced a considerable amount of flexibility and programmability into network programming and hence it has greatly impacted the field of computer networks [4]. Intent Framework provided by SDN controller allows programmers to propose high-level policies without the consideration of any low-level device details.

In virtualization, trend containerization is the new initiative [2], [5], [6]. A container contains all the required elements that are mandatory to run the procedure with system tools and runnable libraries, which implies that it provides a namespace cell and comprehensive filesystem. Moreover, a container is eminently portable as it has no external dependencies. To get rid of naming and version conflict issues, the container’s namespace is separated from other containers. Such as, two web servers can run in two containers on a bare-metal machine without introducing conflicts of port 8080 or 8181. These highly useful features of portability and independence make the testing and maintenance of containerized application very easy and simple which remove the complexities from the entire cycle of the application.

For the management of containers in a cluster, Google has released an open-source project named Kubernetes. It is based on over 10 years of running container clusters internally. The purpose of Kubernetes is to deploy, manage and make a schedule of containers to clusters of machines. Basically, Kubernetes is designed to provide all the primal elements for microservice architectures and to have these the network support is mandatory since almost

Manuscript received June 7, 2019; revised February 2, 2020.
doi:10.12720/jcm.15.3.237-244

all the functions of applications such as Discovery, communication, and synchronization are performed through the network. Therefore, for any Kubernetes cluster, network setup is one of the most significant and prime factors.

As the technology is boosting exponentially so the total number of connected devices in 2020 will be expected to reach around 25 billion as recently stated in a survey of Gartner [7]. Due to the rapid increase in associated data streams and objects, problems like data confidentiality and network performance have become significantly prominent which also has put a lot of pressure on the network and edge layer. In addition to this, managing an IT infrastructure that contains numerous running machines each having multiple containers, is itself a tedious task. Keeping this in view, automated devices are getting into fashion rapidly and becoming popular day by day.

In this complicated situation, this paper proposes a system using a Kubernetes cluster to deal with the connectivity problem and slicing in a dispersed containers-based scheme. Specifically, the Kubernetes cluster implements a system in three different machines. Open vSwitch (OVS) is used as one of the mediator plug-ins to provide connectivity in Kubernetes' cluster components. SDN controller communicates with Open vSwitch (OVS) through OpenFlow protocol and provides an ideal platform to control slicing among containers across the overlay network using intent-based networking. Keeping in view the importance of slicing in a 5G network, we implemented the system with the SDN controller and IBN Manager to control the slicing using intent-based networking.

The proposed system consists of a web application i.e. IBN Manager to provide a graphical user interface for slicing management by providing intent, SDN controller to control the Open vSwitch (OVS) by modifying flow table according to the provided intent and Kubernetes cluster.

The remaining paper is composed as follows. Section II explains the related work. The proposed system is presented in section III. Experimental results and conclusion, as well as future work, are discussed in Sections IV and V respectively.

II. RELATED WORK

In the past few years, many experimental and methodical works have been proposed dealing with service composition in Cloud computing. In [8] researchers proposed a technique for managing the entire lifespan of an application for automatically providing the Cloud business applications. Cloud management tools are employed in this technique which allows them to regulate and maintain the necessary software components disposal and their related dependencies. Specifically, a Linked Unified Service Description Language (LUSD) is proposed which describes different services that are analogous to user's needs and demands. Designing of an edge Cloud network requires consideration of several tasks in steps. The foremost step is to decide where the facilities should be installed among the feasible slots followed by the assignment of sets of access points. In parallel, it should be capable to satisfy the service-level agreements and to support the management of VM, moreover, it should take into account the information about partial user mobility. In [9] a link-path formulation supported by heuristics to estimate and calculate the answers in a rational amount of time is discussed which specifically mentions the advantages of mobility management for both users and VMs. In [10] a Cloud4IoT platform is proposed which provides self-regulating deployment, management and non-static configuration of IoT support software elements and comprehensive data applications that are employed for analyzing and processing the data. It allows new sensor objects' plug-and-play integration and provides dynamic workload scalability.

Container technology can provide ease in the distribution and deployment of applications and services. Moreover, container technology is very easy and fast to package [11]-[13]. Because of these factors, it has gained tremendous popularity.

Primarily containers are processes but to provide namespace isolation they use mechanisms as chroot [14]. Containers are notably portable due to the interdependence of container-based applications. This makes them able to distribute and deploy easily [2]. Moreover, containers are much lightweight as that of VMs and their initialization process is way faster than VMs [15, 16]. Containers can be operated by the Linux and Windows operating systems as per need or desire. Containers can be exploited both within a VM or on bare metal. There are many container management systems [5], [6] but Docker [2] is the most famous and noted one among all.

Mostly the containerized applications comprise of several containers. Such as in Hadoop [17], reduces and mapper is a separate container. Nowadays advance web services are consisting of multiple small services e.g. database, restful APIs, load balancer and webs server are deployed with multiple containers for each layer on multi-host machine cluster. Containers-based cluster orchestration software Mesos [18] or Kubernetes [5] provides controlled deployment in this situation. Scaling, upgrading or replacement of container is safe in this architecture because the replacement of defected containers is very quick on the same host or another host machine. Network performance has a globally impact on the performance of an application in case of working with a single application container [19]-[22].

A container networking solution must be able to tackle the plausible four cases that happen to own the possibility that if the containers run on a physical machine or VM hosts. Today most of the containers use overlay networks frequently in order to have maximum portability. There are numerous available possible ways at the software end which provide overlay networking like Weave [23] and Calico [24]. These solutions use software bridges to connect the virtual interface of containers with the network interface card of a host machine and provide communication medium between them. The router also executes the right tunneling (encapsulation and de-encapsulation) for the movement of the traffic between the overlay fabric and the tangible network. To connect with software routers on added hosts, standard networking protocols like BGP are exploited at the routers' end. Containers always remain in the state of doubt related to further container's location because they terminate the IP packet through this overlay.

Deployment management of microservices for Edge computing is proposed in [25] which also considers geographical constraints. It assumes that a shared service is possessed many microservices and users can make a selection of the definitive regions in which these services will be deployed using geolocation deployment constraints. It also includes the instruction for the deployment of each node involving Fog computing.

Considering the management of networking services, a protocol for management control is proposed in [26]. It supplies a way for managing the joint network of cloud by providing a solution to transport application programming interface. Subsequently, it allows communication of divergent control planes in order to provide restoration and provisioning of end-to-end quality of service services. In [27] authors specifically discussed the upcoming 5G network applications in the future. In this paper, the authors discussed the Mobile Network Operator which is constructed over a common physical network by virtualizing backhaul tenant using SDN/NFV orchestrator tool. Different infrastructure providers own divergent technologies such as packet-based technology or optical technology etc. that form the basis of a physical aggregation network over which backhaul tenants are built.

In [28] authors proposed the latest method for the management of energy-familiar services using energy-familiar resource management framework for scattered Cloud setup. In this research, authors have mainly focused on proposing a system's architecture in the context of managing IT resources and networks. To drill and examine the communication problems which are associated with SO deployment in IoT systems, a compound methodology is presented in [29] and [30]. It basically models and simulates IoT systems with regard to network simulator and multi-agent systems.

The goal of authors in the research work [31] is the development of a platform in order to aid the non-static resource-provisioning that rest on Kubernetes. The detailed monitoring system, provisioning of resources vigorously and the restraint loop are major features of this developed platform.

To the best of our knowledge, the proposed paper presents a unique system for service orchestration in containers. This system is unique due to providing on-demand service in the Kubernetes cluster using intent-based slicing in the SDN overlay network.

III. System Architecture

This section presents the complete architecture of our system and discusses each module in detail. The uniqueness of our system is given in the last paragraph of "Related Work". We have designed the proposed system with the context of Software Defined Network which consists of three major parts and each representing an SDN layer as shown in Fig. 1. IBN Manager represents the management layer, Open Networking Operating System (ONOS) represents the controller layer and Kubernetes nodes represent the data layer.

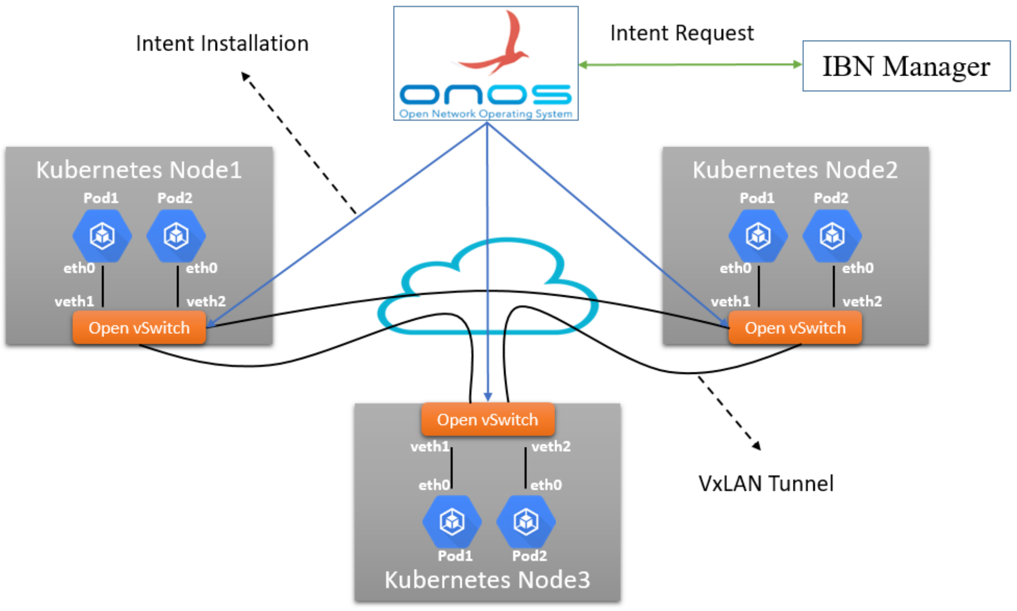


Fig. 1. System architecture.

Our objectives in this section are to provide the detail for creating SDN based Kubernetes cluster in which pods should be routable across the nodes as well as all containers inside the pods should communicate will others and provide slicing mechanism among the containers using Intent-Based Networking (IBN).

The Intent-Based Networking (IBN) Manager is developed as a Java-based web application for the end-user for defining their policy. Fig. 2 shows the IBN Manager consisting of a back-end and a front-end module.

According to our research, a graphical user interface (GUI) is mandatory for the end-user to define their policies. The GUI's main purpose is to reduce human error while providing a friendly environment for defining network policies.

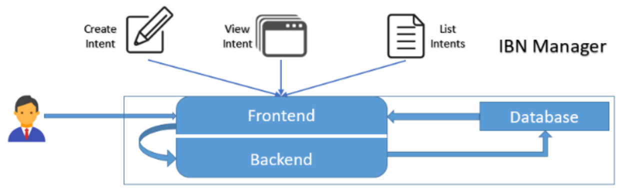
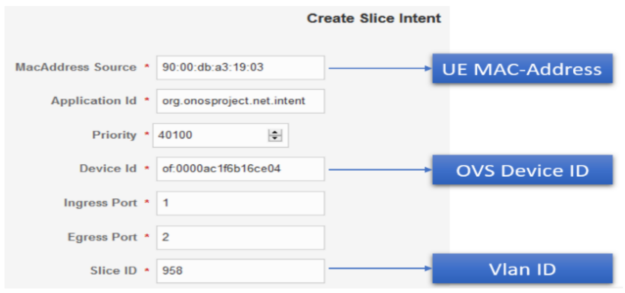


Fig. 2. Frontend and backend moduel representation of IBN manager.



Create Slice Intent

MacAddress Source *	90:00:db:a3:19:03	UE MAC-Address
Application Id *	org.onosproject.net.intent	
Priority *	40100	
Device Id *	of:0000ac1f6b16ce04	OVS Device ID
Ingress Port *	1	
Egress Port *	2	
Slice ID *	958	Vlan ID

Fig. 3. Creating slice using point to point Intents API of ONOS controller.

The Front-end module is shown in Fig. 3 provides an interface where the user can create the intent. It gives a detailed view of each intent and a list of all installed intents. The Front-end module can query the information from a database and then submit a creation request to the back-end module of IBN Manager.

One of the features of the IBN Manager is the creation of Point to Point (P2P) intents, which gives an interactive way to define a detailed policy between two switches and the rest of the work is done by the backend module. By using P2P intents with ONOS, we can achieve slicing. In order to realize this, specific information need to be configured for each intent that is created as shown in Fig. 3. P2P Intents use the source MAC Address of the pod for which a specific VLAN ID needs to be assigned for slicing, it also requires the Identification of the switch (this can be obtained from the ONOS controller list of devices) as well as the Ingress and Egress ports. The most important field that is used for denoting a specific slice is the VLAN ID.

The Backend module shown in Fig. 4 contains a request handler that is responsible for communication between the front-end, the database, and the ONOS. It also contains a policy-making module that takes input from the front-end and maps all values to the variables in order to make a proper policy that is intent-based. The Backend also contains a CLASS to the JSON module for converting objects (created with the policy-making module) to JSON format in order to achieve Restful communication with ONOS.

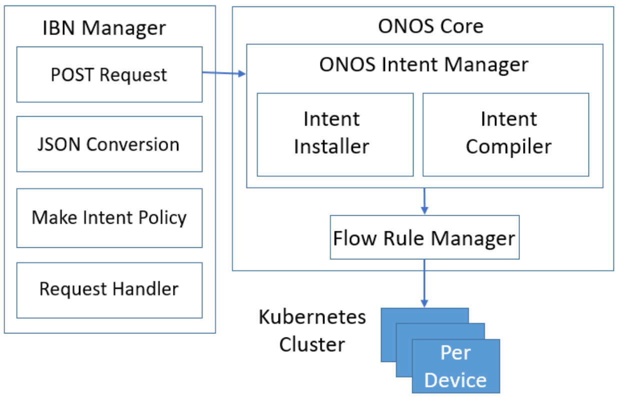


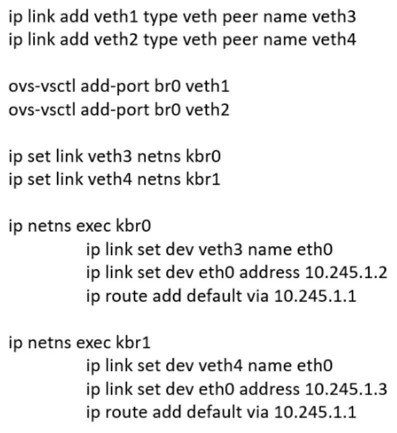
Fig. 4. Internal architecture of IBN manager and ONOS module involve in intent based networking.

ONOS controller at the control layer communicates with both IBN Manager via the northbound API and with the Kubernetes cluster via southbound API. The intent manager module of the ONOS controller receives the request from IBN Manager in the form of JSON and converts it into the low-level command after compiling it which is executable on switches. Then posts the intent (in the form of JSON) installer module of ONOS core for installation. After the successful installation, our intent installer communicates with the flow rule manager module to keep a record of the created intent for the relevant Kubernetes node.

IV. Experiment Setup and Results

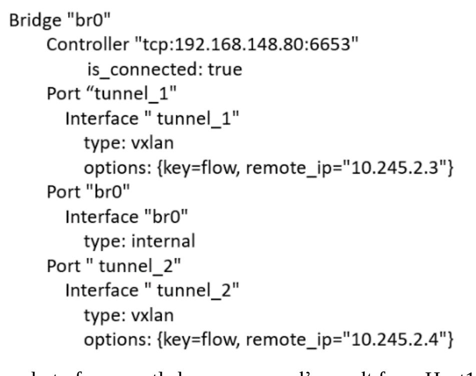
In this section, we present complete detail involved in the construction of an overlay network for the Kubernetes cluster and in the configuration of the ONOS controller. We have deployed IBN Manager in windows desktop machine and ONOS controller deployed in a virtual machine running in the same windows host which is configured with host machine using bridge networking adaptor of Virtual Box. All three Kubernetes nodes are deployed in three physical machines with Ubuntu 16.04 LTS operating system.

Let's take an example of a Kubernetes node 1 with the help of Fig. 1 to explain all configurations required to create a setup in one hypervisor because this configuration is enough to understand the configuration of the rest two Kubernetes nodes. At first, we installed Kubernetes in the Ubuntu machine and created two pods with two different IP addresses. After creating two pods, we created an OVS bridge and then configured the SDN controller IP in order to provide connectivity between OVS and the controller. Every switch device contains a unique Identification and it appears in ONOS controller in the list of devices when it is connected to the ONOS controller. Unt il now OVS can communicate with the SDN controller but the pods are running alone. To establish communication between pods and OVS, we need to create two pairs of virtual Ethernet interfaces veth1, veth2, veth2, veth4 and the links between them as shown in the first two lines of Fig. 5 virtual interfaces are the only solution to create a connection between Linux and OVS bridges. In Fig. 5, third and fourth lines are used to assign veth1 and veth2 interfaces to OVS and sixth and seventh lines are assigned veth3 and veth4 to Kubernetes pods. The last two groups of command rename the virtual interface, assign an IP to the virtual interface and default gateway. At this point, we have successfully completed this setup in one machine with two pods of Kubernetes, connected to OVS which is further connected to the SDN controller. By replicating the procedure of the first machine on the other two machines, the basic experimental setup is completed which is monitored by the ONOS controller. VxLAN tunnel is used to support multi-host overlay networking between Kubernetes pods across the nodes in this experiment. Virtual LAN (VLAN) instances or VxLAN (encapsulated L2) can be used to achieve slicing at layer 2. In the meantime, VLANs have viewable limitations of scalability and service location stickiness in the growing cloud environment, VxLAN solves all of these problems by introducing the larger tag space as compared to VLAN (VLAN: 2^12 = 4096, VxLAN: 2^24 = 16777216).



```
ip link add veth1 type veth peer name veth3
ip link add veth2 type veth peer name veth4

ovs-vsctl add-port br0 veth1
ovs-vsctl add-port br0 veth2

ip set link veth3 netns kbr0
ip set link veth4 netns kbr1

ip netns exec kbr0
        ip link set dev veth3 name eth0
        ip link set dev eth0 address 10.245.1.2
        ip route add default via 10.245.1.1

ip netns exec kbr1
        ip link set dev veth4 name eth0
        ip link set dev eth0 address 10.245.1.3
        ip route add default via 10.245.1.1
```

Fig. 5. Configuration detail between pods and OVS on Host 1.



```
Bridge "br0"
    Controller "tcp:192.168.148.80:6653"
        is_connected: true
    Port “tunnel_1"
      Interface " tunnel_1"
        type: vxlan
        options: {key=flow, remote_ip="10.245.2.3"}
    Port "br0"
      Interface "br0"
        type: internal
    Port " tunnel_2"
      Interface " tunnel_2"
        type: vxlan
        options: {key=flow, remote_ip="10.245.2.4"}
```

Fig. 6. Snapshot of ovs-vsctl show command’s result from Host1

According to system architecture shown in Fig. 1, we have created two VxLAN tunnels in each OVS bridge which are pointed to the other two host machines for the purpose of connection establishment between three Kubernetes nodes. Fig. 6 shows the configuration of two VxLAN tunnels in the OVS of host1 pointing towards the other two hosts. SDN overlay network completion after replicating the procedure of VxLAN configuration and resultant topology is shown in Table I.

TABLE I. TABULAR REPRESENTATION OF NETWORK TOPOLOGY

Switch	Port 1	Port 2	Port 3	Port 4
S1	Pod 1	S2	Pod 2	S3
S2	Pod 1	S1	Pod 2	S3
S3	Pod 1	S1	Pod 2	S2

TABLE II. OPENFLOW TABLE ENTRY OF OPEN VSWTICH

Switch	Source	Destination	VLAN_ID	Action
S1	1	2	958	PUSH
S1	2	1	958	POP
S1	3	4	957	PUSH
S1	4	3	957	POP
S2	1	2	958	PUSH
S2	2	1	958	POP
S2	3	4	957	PUSH
S2	4	3	957	POP
S3	1	2	958	PUSH
S3	2	1	958	POP
S3	3	4	957	PUSH
S3	4	3	957	POP

Until now, the whole system has been deployed successfully as mentioned in Section III and all the three layers including Management, Control and Data Layer has been started communication with each other successfully. For the testing of slicing scenario in the deployed system, we utilized the Kubernetes cluster as follows:

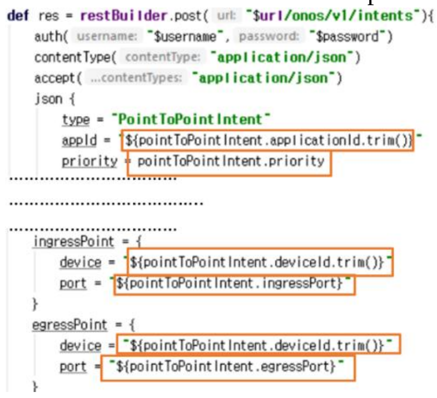
- We have deployed a video server in the container of pod1 and web browsing sever in the container of pod2 of K-Node3.
- Firewall software is deployed in containers of both pods of K-Node2.
- Clients for video server is deployed in the container of pod1 and web browser server is deployed in the container of pod2 of K-Node1.

VxLAN tunnels in each Kubernetes node isolate the traffic of each node to separate the traffic of each pod, a VLAN_ID is needed to attach with each port of OVS. As explained earlier in Section III, network administrators can configure each port of OVS using IBN Manager’s GUI by providing their intents.

The testing procedure for creating slicing and traffic flow is precisely explained below. Prior to start testing, an overlay network should be created first. The network administrator can use IBN Manager to install their intents in Open vSwitch (OVS). IBN Manager provides multiple functionalities for intent management and Fig. 3 shows the image of intent creation page in which network administrators can provide their intents by filling all fields in the perspective of creating slicing between containers and submit it to IBN Manager.

Intents are created using IBN Manager with specific directives on all Open vSwitch for pushing and popping the VLAN ID of the network that serves the slice of the connection, as shown in Table II.

Before sending the intent to ONOS, backend module of IBN Manager receives the submitted intent from frontend and after compiling it into the intent policy as shown in Fig. 7, then transforms it to JSON object which is a contractual object, transferred between management application and the intent framework of the ONOS controller. Rectangular boxes in Fig. 7 shows the variables that hold the intent information while making intent policy in the backend module. Now ONOS receives the intent request from IBN Manager and installs those rules in the flow table of targeted Open vSwitch located in the Kubernetes node using southbound as explained in the previous section. Fig. 8 shows that we have achieved two slices in the overlay network after installing the flow rule patterns presented in Table II for all of the Open vSwitch.



```
def res = restBuilder.post( url: "$url/onos/v1/intents"){
    auth( username: "$username", password: "$password")
    contentType( contentType: "application/json")
    accept( ...contentTypes: "application/json")
    json {
        type = "PointToPointIntent"
        appId = "${pointToPointIntent.applicationId.trim()}"
        priority = pointToPointIntent.priority
.................................

.....................................

.................................
    ingressPoint = {
        device = "${pointToPointIntent.deviceId.trim()}"
        port = "${pointToPointIntent.ingressPort}"
    }
    egressPoint = {
        device = "${pointToPointIntent.deviceId.trim()}"
        port = "${pointToPointIntent.egressPort}"
    }
```

Fig. 7. IBN Manager's code for making intent policy from input intent

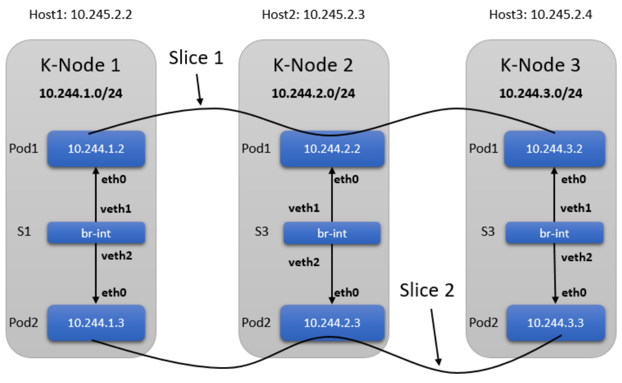


Fig. 8. Complete end to end slicing for service orchestration

We have executed two tests to verify the two created slices. We added the valid IP address in a successful test case and we also added the wrong IP address in failure test for each slice in firewall software. For a successful test, K-Node2 authenticated the request of both clients of K-Node1 and received data from servers of K-Node3 successfully. In Failure test, K-Node2 blocked the request of both clients of K-Node1.

Fig. 9 shows the result of the iperf test which is executed on both clients of K -No de1 to provide the evidence of traffic isolation in each slice. The result shows that the slice did not disturb the performance of another slice.

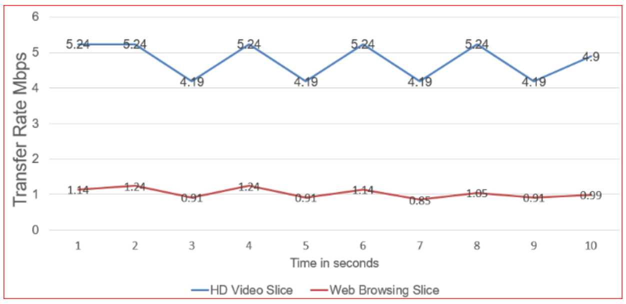


Fig. 9. Result of iPerf test for each slice

V. Conclusion

In this paper, we presented different aspects for the management of Kubernetes cluster by introducing an SDN controller. In this study, we presented the creation of an overlay network of Kubernetes cluster using Open vSwitch and management of traffic flow by creating slicing among the containers, deployed on different separate hypervisors using the SDN controller and IBN Manager. Experimental results have shown that the proposed system has successfully achieved two slices using an intent-based networking approach, one slicing created for video streaming and second slicing created web browsing.

Open Virtual Networking (OVN) is emerged as the best solution to manage the Kubernetes cluster in the mode of overlay network which is configured using Open vSwitch (OVS). OVN monitored all Kubernetes nodes centrally by using the northbound and southbound databases. OVN northbound database is effectively like a public API, where Kubernetes defines its network configurations. OVN southbound database is used for the internal state of OVN and it contains more detailed information that how a network should behave which is intended to be consumed by the hypervisor. OVN creates the logical switches and ports among the containers running on different hosts in an overly-network and produces less overhead due to native OVS support for L2/L3 overlays.

Acknowledgment

This research was supported by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2019-2017-0-01633) supervised by the IITP (Institute for Information & communications Technology Planning & Evaluation).

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2016R1D1A1B01016322).

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

Adeel Rafiq conducted the research, developed the intent-based networking software and wrote the paper while Asif Mehmood participated in the establishment of kubernetes test-bed. All of the work was done under the supervision of Professor Wang-Cheol Song. All authors had approved the final version of paper.

References

[1] I-Scoop, 5g and iot in 2018 and beyond: The mobile broadband future of iot, [Online]. Available: https://www.i-scoop.eu/internet-of-things-guide/5g-iot/.

[2] Docker. [Online]. Available: http://www.docker.com/

[3] T. Taleb, "Towards carrier cloud: Potential challenges & solutions," *IEEE Wireless Commun.*, vol. 21, no. 3, pp. 80-91, June 2014.

[4] T. Subramanya, R. Riggio, and T. Rasheed, "Intent-based mobile backhauling for 5G networks," in *Proc. 12th International Conference on Network and Service Management* (CNSM), Montreal, QC, 2016, pp. 348-352

[5] Kubernetes. [Online]. Available: http://kubernetes.io/

[6] CoreOS. [Online]. Available: https://coreos.com/

[7] Gartner. [Online]. Available: https://www.gartner.com/newsroom/id/2970017

[8] H. Benfenatki, C. F. D. Silva, G. Kemp, A. N. Benharkat, P. Ghodous, and Z. Maamar, "Madona: A method for automated provisioning of cloud-based component-oriented business applications," *Service Oteamriented Computing and Applications*, vol. 11, no. 1, pp. 87–100, 2017.

[9] A. Ceselli, M. Premoli, and S. Secci, "Mobile edge cloud network design optimization," *IEEE/ACM Transactions on Networking*, 2017.

[10] D. Pizzolli, G. Cossu, D. Santoro, L. Capra, C. Dupont, D. Charalampos, F. De Pellegrini, F. Antonelli, and S. Cretti, "Cloud4iot: A heterogeneous, distributed and autonomic cloud platform for the iot," in *Proc. International Conference on Cloud Computing Technology and Science, CloudCom*, 2017, pp. 476–479.

[11] Docker. Docker community passes two billion pulls. [Online]. Available: https://blog.docker.com/2016/02/docker-hub-two-billion-pulls/

[12] A. Dragojevic, D. Narayanan, M. Castro, and O. Hodson, "Farm: Fast remote ´memory," in *USENIX NSDI*, 2014.

[13] Iron.io. Docker in production – what we've learned launching over 300 million containers. [Online]. Available: https://www.iron.io/docker-in-production-what-weve-learned/

[14] FreeBDS. chroot – FreeBDS Man Pages. [Online]. Available: http://www.freebsd.org/cgi/man.cgi

[15] A. Madhavapeddy, T. Leonard, M. Skjegstad, T. Gazagnaire, *et al.*, "Jitsu: Just-in-time summoning of unikernels," in *USENIX NSDI*, 2015.

[16] D. Merkel, "Docker: Lightweight linux containers for consistent development and deployment," *Linux J.*, 2014.

[17] Apache Hadoop. [Online]. Available: http://hadoop.apache.org/

[18] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center," in *USENIX NSDI*, 2011.

[19] G. Ananthanarayanan, S. Kandula, A. G. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris, "Reining in the outliers in map-reduce clusters using mantri," in *USENIX OSDI*, 2010.

[20] M. Chowdhury and I. Stoica, "Efficient coflow scheduling without prior knowledge," in *ACM SIGCOMM*, 2015.

[21] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, "Managing data transfers in computer clusters with orchestra," in *ACM SIGCOMM*, 2011.

[22] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient coflow scheduling with varys," in *ACM SIGCOMM*, 2014.

[23] Weave Net. [Online]. Available: https://www.weave.works/

[24] Project calico. [Online]. Available: https://www.projectcalico.org/

[25] M. Villari, A. Celesti, G. Tricomi, A. Galletta, and M. Fazio, "Deployment orchestration of microservices with geographical constraints for edge computing," in *Proc. IEEE Symposium on Computers andCommunications*, July 2017, pp. 633–638.

[26] A. Mayoral, R. Vilalta, R. Munoz, R. Casellas, R. Martinez, M. Moreolo, J. Fabrega, *et al.*, "Control orchestration protocol: Unified transport api for distributed cloud and network orchestration," *Journal of Optical Communications and Networking*, vol. 9, no. 2, pp. A216–A222, 2017.

[27] R. Martinez, A. Mayoral, R. Vilalta, R. Casellas, R. Munoz, S. Pachnicke, T. Szyrkowiec, and A. Autenrieth, "Integrated sdn/nfv orchestration for the dynamic deployment of mobile virtual backhaul networks over a multilayer (packet/optical) aggregation infrastructure," *Journal of Optical Communications and Networking*, vol. 9, no. 2, pp. A135–A142, 2017.

[28] G. Fioccola, P. Donadio, R. Canonico, and G. Ventre, "Dynamic routing and virtual machine consolidation in green clouds," in *Proc. International Conference on Cloud Computing Technology and Science, CloudCom*, 2017, pp. 590–595.

[29] G. Fortino, R. Gravina, W. Russo, and C. Savaglio, "Modeling and simulating internet-of-things systems: A hybrid agent-oriented approach," *Computing in Science Engineering*, vol. 19, no. 5, pp. 68–76, 2017.

[30] G. Fortino, W. Russo, C. Savaglio, W. Shen, and M. Zhou, "Agentoriented cooperative smart objects: From iot system design to implementation," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, no. 99, pp. 1–18, 2017.

[31] C. C. Chang, S. R. Yang, E. H. Yeh, P. Lin, and J. Y. Jeng, "A kubernetes based monitoring platform for dynamic cloud resource provisioning," in *Proc. GLOBECOM 2017*

- *2017 IEEE Global Communications Conference*, Dec. 2017, pp. 1–6

Adeel Rafiq is a Ph.D. student at the Computer Engineering department, Jeju National University, South Korea. He received his Master's degree in Software Engineering from University of Engineering and Technology, Taxila, Pakistan in 2014 and Bachelor degree in Computer Engineering from COMSATS Institute of Information Technology, Wah, Pakistan in 2011.
He worked as a research assistant and visiting lecturer in UET Taxila and CIIT Wah repectively. He also has professional work experience of 5 years as a software developer in renowned US-based software houses in Pakistan.
Currently, his fields of research comprise of SDN, NFV, E2E Orchestration, Open Source MANO (OSM), ONOS, Intent-Based Networking (IBN) and Load Balancing applications.

Asif Mehmood is a Master's student at Computer Engineering department, Jeju National University, South Korea. He received his Bachelor's degree in Computer Science from COMSATS Islamabad, Pakistan in 2015.
He worked as a Software Engineer in several organizations such as U Microfinance Bank, TypeUX, Horizon, BroadPeak Technologies. The total years of industrial experience are 3 years in very dynamic and fast-paced environments. Currently he is researching on IBN, Orchestration, SFC, SDN, NFV using different techniques such as Kubernetes, M-CORD, COMAC, etc

Wang-Cheol Song work at the Computer Engineering department, Jeju National University, South Korea. Member of Advanced Network Forum (correspondent), Korea Contents Association (correspondent Distinguished Service medal 2005), Korea Information Science Society (correspondent), Korean Institute Communications and Sciences (correspondent).
He received B.S. degree in Food Engineering and Electronics from Yonsei University, Seoul, Korea in 1986 and 1989, respectively. And M.S. and Ph.D. in Electronics studies from Yonsei University, Seoul, Korea, in 1991 and 1995, respectively. Since 1996 he has been working at Jeju National University. His research interests include VANETs and MANETs, Software Defined Networks, network security, and network management.