IEIE Transactions on Smart Processing and Computing, vol. 6, no. 1, February 2017
https://doi.org/10.5573/IEIESPC.2017.6.1.060

A Prototype Virtual Network Embedding System using OpenStack

Yukinobu Fukushima[1], Kohei Sato[1], Itsuho Goda[1], Heung-Gyoon Ryu[2], and Tokumi Yokohira[1]

[1] Graduate School of Natural Science and Technology, Okayama University / Okayama 700-8530, Japan
{fukusima, yokohira}@okayama-u.ac.jp
[2] Department of Electronics Engineering, Chungbuk National University / Cheongju, Korea 361-763, Korea
ecomm@chungbuk.ac.kr

***** Corresponding Author: Yukinobu Fukushima

Received November 5, 2016; Revised January 5, 2017; Accepted February 2, 2017; Published February 28, 2017

** Regular Paper*

** Extended from a Conference: Preliminary results of this paper were presented at the ITC-CSCC 2016. This paper has been accepted by the editorial board through the regular review process that confirms the original contribution.*

***Abstract*:** Network virtualization enables us to make efficient use of resources in a physical network by embedding multiple virtual networks in the physical network. In this paper, we develop a prototype of a virtual network embedding system. Our system consists of OpenStack, which is an open source cloud service platform, and shell scripts. Because OpenStack does not provide a quality of service control function, we realize bandwidth reservation for virtual links by making use of the ingress policing function of Open vSwitch, which is a virtual switch used in OpenStack. The shell scripts in our system automatically construct the required virtual network on the physical network using the OpenStack command-line interface, and they reserve bandwidth for virtual links using the Open vSwitch command. Experimental evaluation confirms that our system constructs the requested virtual network and appropriately allocates node and link resources to it.

***Keywords*:** Network virtualization, Virtual network embedding, OpenStack

1. Introduction

Network virtualization has attracted increasing attention in both industry and academia [1, 2]. In network virtualization, multiple virtual networks consisting of virtual nodes and virtual links are embedded in a single physical network consisting of physical nodes and physical links. As a result, we can share and effectively use the resources in the physical network among multiple virtual networks.

When embedding a virtual network in a physical network, we have to determine to which physical node/path in the physical network each virtual node/link in the virtual network is mapped so that a predetermined objective function is optimized under some constraints (e.g., a resource constraint). This problem is called a virtual network embedding (VNE) problem [3].

A large number of algorithms for the VNE problem (VNE algorithms) have been proposed in the literature [3-8]. However, they were evaluated only in computer simulations. In order to evaluate various VNE algorithms in a real situation with various performance measures (e.g., virtual network provisioning time), we need a VNE system that actually constructs virtual networks on a physical network.

In this paper, we develop a prototype VNE system. Our system uses OpenStack [9] as the base platform. OpenStack is an open source cloud service platform for infrastructure-as-a-service (IaaS), and enables us to generate a tenant network, which consists of virtual machines (VMs) and virtual links, in data centers. We can use the tenant networks generated by OpenStack as virtual networks in network virtualization by implementing network functions (e.g., a router) on the VMs in the tenant networks, regarding them as virtual nodes. However, OpenStack does not provide a quality of service (QoS) control function, and consequently, we cannot reserve bandwidth for virtual links. In order to achieve this, we make use of the ingress policing function of Open vSwitch, which is a virtual switch used in OpenStack. Our system

consists of OpenStack and shell scripts. The shell scripts automatically construct the required virtual network on the physical network using the OpenStack command-line interface, and they reserve bandwidth for virtual links using the Open vSwitch command. In addition, we experimentally evaluate our system.

2. Virtual Network Embedding and OpenStack

2.1 Virtual Network Embedding

Fig. 1 depicts an example of virtual network embedding. In virtual network embedding, the physical network is denoted by an undirected graph, $\boldsymbol{G}^P = (\boldsymbol{N}^P, \boldsymbol{V}^P)$, where $\boldsymbol{N}^P$ is the set of physical nodes, and $\boldsymbol{V}^P$ is the set of physical links. Each physical node is equipped with a limited amount of node resources (i.e., CPU, memory, and storage). In Fig. 1, the number attached to each physical node shows the remaining amount of node resources on the physical node. For example, physical node A has 50 remaining node resources. Similarly, each physical link is equipped with a limited amount of link resources (i.e., bandwidth). In Fig. 1, the number attached to each physical link shows the remaining amount of link resources on the physical link. For example, physical link (A, D) has eight remaining link resources.

The virtual network, which is mapped to the physical network in virtual network embedding, is denoted by an undirected graph, $\boldsymbol{G}^V = (\boldsymbol{N}^V, \boldsymbol{V}^V)$, where $\boldsymbol{N}^V$ is the set of virtual nodes, and $\boldsymbol{V}^V$ is the set of virtual links. Each virtual node requires a predetermined amount of node resources on the physical node to which the virtual node is mapped. In Fig. 1, the number attached to each virtual node shows the amount of node resources required by the virtual node. For example, virtual node *a* requires 10 node resources. Similarly, each virtual link requires a predetermined amount of link resources on the physical path to which the virtual link is mapped. In Fig. 1, the number attached to each virtual link shows the amount of link resources required by the virtual link. For example, virtual link (*a*, *b*) requires one link resource.

In virtual network embedding, we have to determine how the requested virtual network is mapped to the physical network, so that some objective function (e.g., to maximize the financial profit of the physical network provider) is optimized. A virtual network embedding problem is divided into two subproblems: a *node mapping problem* and a *link mapping problem*. In the node mapping problem, we have to determine to which physical node each virtual node is mapped. In the link mapping problem, we have to determine to which physical path each virtual link is mapped.

In node/link mapping problems, resource constraint has to be satisfied. Each virtual node can only be mapped to physical nodes that have enough remaining node resources. Similarly, each virtual link can only be mapped to physical paths that have enough remaining link resources on all the physical links along the physical path.

A large number of VNE algorithms have been

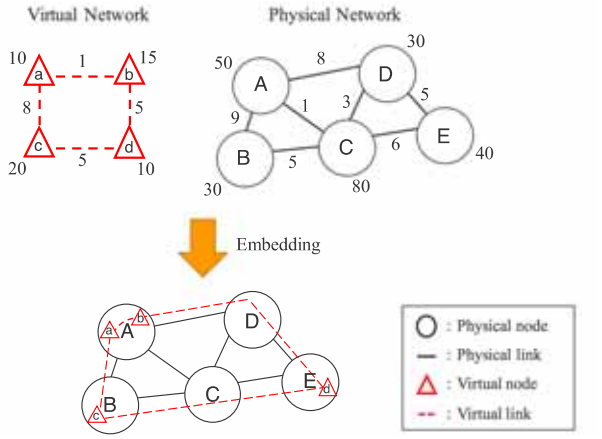

Fig. 1. Example of virtual network embedding.

proposed [4, 5, 7]. Yu et al. [4] proposed a VNE algorithm where the goal is to maximize the financial profit of the physical network provider by accommodating as many virtual networks as possible. The VNE algorithm first solves the node mapping problem in a greedy manner: it assigns the virtual nodes with a larger node resource request to the virtual node with the larger remaining node resources. Then, the VNE algorithm solves the link mapping problem: it assigns each virtual link to multiple physical routes by solving the multi-commodity flow problem. Chowdhury et al. [5] proposed a VNE algorithm where the goal is to minimize the cost associated with virtual network embedding. The VNE algorithm solves both the node mapping problem and the link mapping problem in a coordinated manner. It first creates an augmented graph over the physical network by adding meta-nodes (each of which corresponds to a virtual node) to the physical network. Then, it obtains the set of routes between the meta-nodes whose corresponding virtual nodes are also connected by a virtual link on the virtual network, so that the embedding cost incurred by the set of routes is minimized. The links between a meta-node and a physical node on the augmented graph, which are passed by the set of routes, correspond to the solution to the node mapping problem, and the physical routes between physical nodes, which are passed by the set of routes, correspond to the solution to the link mapping problem. Fischer et al. [7] proposed a VNE algorithm where the goal is to minimize the energy consumption associated with virtual network embedding. It formulates the virtual network embedding problem as an integer linear programming model, solves it, and then obtains the optimal mapping and the optimal energy consumption.

2.2 OpenStack

OpenStack [9] is an open source cloud computing platform for IaaS. With OpenStack, we can construct a tenant network consisting of VMs and virtual links in data centers.

OpenStack consists of multiple components. Among them, *Nova* and *Neutron* play an important role in our VNE system. Nova maintains node resources of physical computers and is responsible for generating and removing

VMs that correspond to virtual nodes in VNE. Neutron is responsible for generating and removing virtual local area networks (VLANs), which correspond to virtual links in VNE. The operations of Nova and Neutron can be requested via WebGUI and command-line interface.

Neutron is not equipped with a QoS control function. Therefore, we have to reserve link resources (i.e., bandwidth) for virtual links without using Neutron. Instead, we directly configure Open vSwitch, which is a virtual switch used in OpenStack, so that we can reserve bandwidth for virtual links. In tenant networks generated by OpenStack, Open vSwitch works as a layer 2 switch, and VMs are connected to virtual links via Open vSwitch within the physical computer. Thus, we expect that we can realize bandwidth reservation by performing ingress policing at the Open vSwitch ports to which VMs are connected.

3. Virtual Network Embedding System

Fig. 2 shows the procedure for VNE. In Step 1, we solve the node mapping problem and the link mapping problem using a VNE algorithm. In Step 2, given the result of virtual network embedding (VNE information), we actually construct the requested virtual network on the physical network. The VNE system developed in this paper processes only Step 2, because the simulation programs for evaluating the existing VNE algorithms are available for processing Step 1.

In our VNE system, we assume that VNE information is described with the Boston University Representative Internet Topology Generator (BRITE) format [10] which is a well-known topology generator.

Fig. 3 depicts an example of VNE information. The first line expresses the numbers of virtual nodes and links in the requested virtual network. The fifth to eighth lines show the result of node mapping. Each of the lines corresponds to mapping of a virtual node to a physical node. Each line consists of three pieces of information: 1) the ID of the virtual node, 2) flavor number (i.e., how much of the CPU/memory/storage resources are requested by the virtual node), and 3) the ID of the physical node to which the virtual node is mapped. For example, the seventh line means that virtual node 2 is mapped to physical node 3 and the amount of node resources requested by virtual node 2 is specified by flavor number 1. (We can define the relationship between flavor number and how much of the node resources are requested by the virtual node in OpenStack in advance.) The eleventh to fourteenth lines show the result of link mapping. Each of the lines corresponds to mapping a virtual link to a physical path. Each line consists of four pieces of information: 1) the ID of the virtual link, 2) the ID of one virtual node at both ends of the virtual link, 3) the ID of the other virtual node at both ends of the virtual link, and 4) the link resources (bandwidth [in Kbps]) allocated to the virtual link. For example, the twelfth line means that virtual link 1, which connects virtual node 0 and virtual node 3, is allocated bandwidth of 10,000 [Kbps] (10 [Mbps]). In this paper, we assume that virtual links are

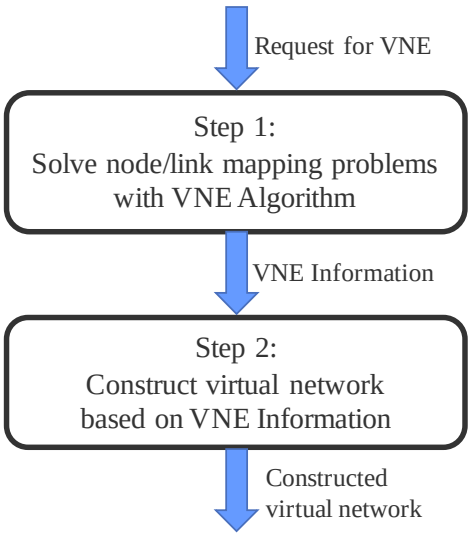

Fig. 2. Procedure for VNE.

```
1   Topology: (4, 4)
2   Model
3
4   Nodes: (4)
5   0 1 1
6   1 1 2
7   2 1 3
8   3 1 1
9
10  Links: (4)
11  0 0 1 100000
12  1 0 3 10000
13  2 1 2 1000
14  3 2 3 100
```

Fig. 3. VNE information,

mapped to physical paths that OpenStack automatically determines. Specifying the physical path to which a virtual link is mapped has been left for future work.

Fig. 4 depicts the constitution of our system. Our system consists of OpenStack and three programs or shell scripts: Preprocess.o, MakeVN.sh, and RsvBW.sh. Preprocess.o is a program written in C, whereas MakeVN.sh and RsvBW.sh are bash shell scripts. A detailed explanation of them follows.

Preprocess.o is responsible for preprocessing. Given the VNE information, it generates and executes MakeVN.sh.

MakeVN.sh is responsible for 1) generating virtual links, 2) generating virtual nodes, and 3) allocating node resources to virtual nodes. Algorithm 1 shows the procedure in MakeVN.sh. In the first to third lines, for each virtual link VL_i in the set ($\boldsymbol{V}^V$) of virtual links in the requested virtual network, we generate a VLAN that corresponds to VL_i by executing OpenStack command line interfaces “neutron net-create” and “neutron subnet-

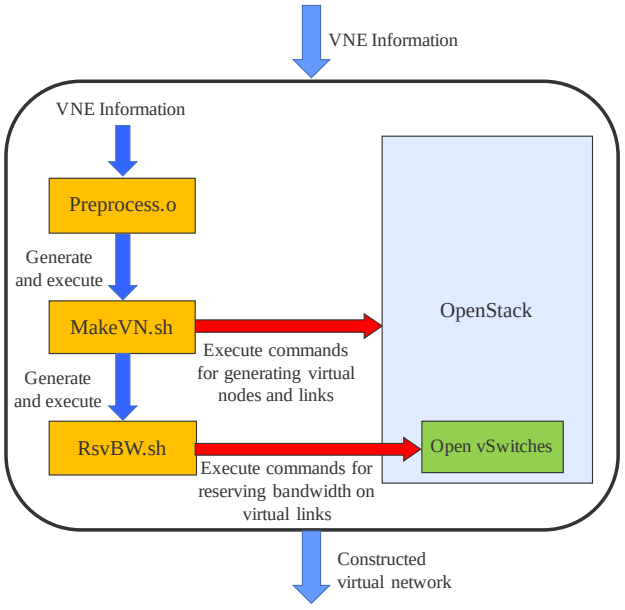

Fig. 4. System constitution.

create." In the fourth to sixth lines, for each virtual node VN_j in the set ($\boldsymbol{N}^V$) of virtual nodes in the requested virtual network, we generate a VM that corresponds to VN_j by executing OpenStack command-line interface "nova boot." In addition, in executing "nova boot," we add the following options: 1) the "--flavor" option to specify how much CPU/memory/storage resources are to be allocated to VN_j, 2) the "--availability-zone" option to specify to which physical node VN_j is mapped, and 3) the "--nic" option, specifying to which virtual link VN_j is connected. In the seventh line, we finally generate and execute RsvBW.sh to reserve link resources on virtual links.

RsvBW.sh is responsible for allocating link resources to virtual links, which is not supported by OpenStack. We realize this by making use of the ingress policing function of Open vSwitch. Algorithm 2 shows the procedure in RsvBW.sh. For each virtual link VL_i in the set ($\boldsymbol{V}^V$) of virtual links in the requested virtual network, we reserve the bandwidth requested by VL_i in the following manner. For each virtual node VN_j at both ends of VL_i, we log in to the physical node to which VN_j is mapped using the ssh command. Then, we limit the inbound traffic rate of an ingress port of the Open vSwitch to which VN_j is connected so that the traffic transmission rate on the virtual link is kept below the allocated bandwidth. We limit the inbound traffic rate of an ingress port by executing the following Open vSwitch command: ovs-vsctl set Interface (Interface name) ingress_policing_rate=(allocated bandwidth)

4. Experimental Evaluation

4.1 Experimental Environment

Fig. 5 depicts our experimental physical network and requested virtual network. The physical network is a star network consisting of four physical nodes and one physical

Algorithm 1. MakeVN.sh

1: **for** each virtual link VL_i in $\boldsymbol{V}^V$ do
2: Execute OpenStack command-line interfaces "neutron net-create" and "neutron subnet-create"
3: **end for**
4: **for** each virtual node VN_j in $\boldsymbol{N}^V$ do
5: Execute OpenStack command-line interface "nova boot"
6: **end for**
7: Generate and execute RsvBW.sh

Algorithm 2. RsvBW.sh

1: **for** each virtual link VL_i in $\boldsymbol{V}^V$ do
2: **for** each virtual node VN_j at both ends of VL_i do
3: Log in to the physical node to which VN_j is mapped using ssh command
4: Execute Open vSwitch command "ovs-vsctl set Interface"
5: **end for**
6: **end for**

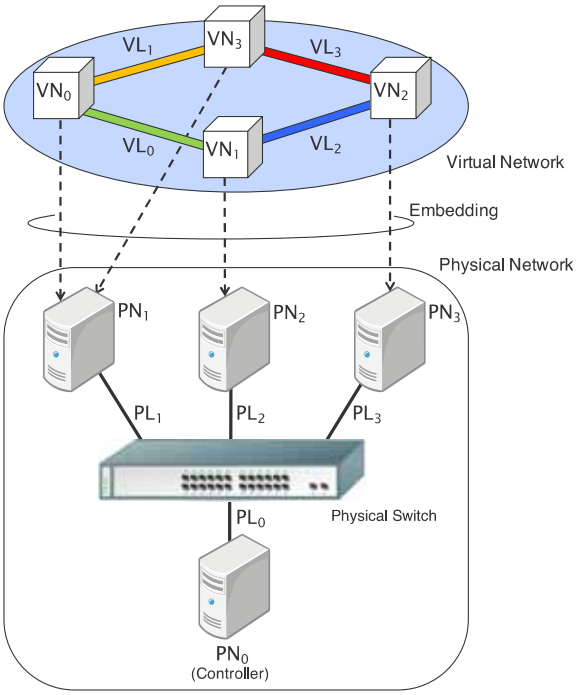

Fig. 5. Experimental physical network and requested virtual network.

switch. Physical node PN_0 is a controller for OpenStack and does not accommodate any virtual node or link, whereas physical nodes PN_1, PN_2, and PN_3 are used to accommodate virtual nodes and links. The remaining resources of the physical nodes and links are summarized in Table 1. The virtual network is a ring network consisting of four virtual nodes and four virtual links. The resource requests of the virtual nodes and links are summarized in Table 2. We assume that a VNE algorithm decided that virtual nodes VN_0 and VN_3 are mapped to physical node 1, virtual node VN_1 is mapped to physical node 2, and virtual node VN_2 is mapped to physical node 3.

Table 1. Remaining resources of physical nodes and links.

Physical nodes or links		Remaining resources
Physical nodes	PN_0	1 CPU (4 cores, 3.30 GHz), 16GB memory, 500 GB storage
	PN_1	1 CPU (4 cores, 3.30 GHz), 16GB memory, 500 GB storage
	PN_2	1 CPU (4 cores, 3.30 GHz), 16GB memory, 500 GB storage
	PN_3	1 CPU (4 cores, 3.30 GHz), 16GB memory, 500 GB storage
Phyical links	PL_0	Bandwidth of 1 Gbps
	PL_1	Bandwidth of 1 Gbps
	PL_2	Bandwidth of 1 Gbps
	PL_3	Bandwidth of 1 Gbps

Table 2. Resources requested by virtual nodes and links.

Virtual nodes or links		Requested resources
Virtual nodes	VN_0	1 CPU, 1GB memory, 3 GB storage
	VN_1	1 CPU, 1GB memory, 3 GB storage
	VN_2	1 CPU, 1GB memory, 3 GB storage
	VN_3	1 CPU, 1GB memory, 3 GB storage
Virtual links	VL_0	Bandwidth of 100 Mbps
	VL_1	Bandwidth of 10 Mbps
	VL_2	Bandwidth of 1 Mbps
	VL_3	Bandwidth of 100 Kbps

In our system, we use CentOS 7.1 for the operating systems of both physical and virtual nodes, and OpenStack version Liverty.

4.2 Results

Fig. 6 shows the virtual network constructed by our system that is the output of OpenStack. In the figure, display icons correspond to virtual nodes, and cloud icons correspond to virtual links. We can see that the requested four-node ring virtual network is appropriately constructed in our system.

We next check whether node resources are appropriately allocated to virtual nodes. We logged in to each of the virtual nodes and confirmed that all four virtual nodes are equipped with the requested CPU/memory/storage resources shown in Table 2.

We finally checked the reachability between virtual nodes via virtual links and whether link resources are appropriately allocated to virtual links. Using the ping command, we confirmed reachability between any two virtual nodes that are directly connected via virtual link. In order to check whether the requested bandwidth is

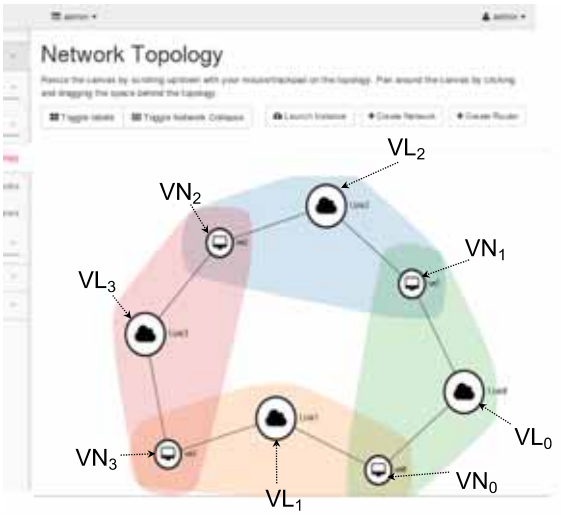

Fig. 6. Virtual network constructed by our system.

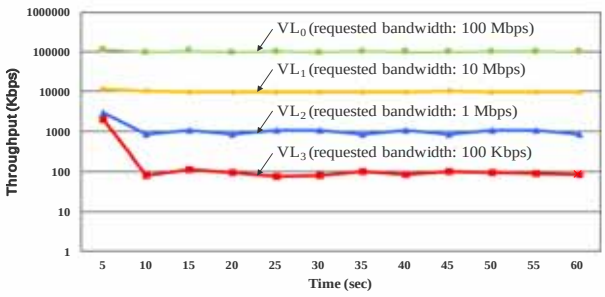

Fig. 7. Throughput of each virtual link.

allocated to each virtual link, we measured their throughput using iPerf. Fig. 7 shows the results. In the figure, we can see that the throughput of each virtual link is bounded by the requested bandwidth shown in Table 2.

5. Conclusions

In this paper, we have developed a prototype of a virtual network embedding system using OpenStack. Because OpenStack does not provide a QoS control function, we have to realize bandwidth reservation for virtual links by making use of the ingress policing function of Open vSwitch, which is a virtual switch used in OpenStack. An experimental evaluation confirms that our system correctly constructs the required virtual network on a physical network and appropriately allocates node and link resources to the virtual network.

One of our future works is to extend our system so it can specify the physical path to which a virtual link is mapped.

Acknowledgement

This work was supported by JSPS KAKENHI Grant Number 15K00129.

References

[1] K. Tutschku, T. Zinner, A. Nakao and P. Tran-Gia, "Network Virtualization: Implementation Steps towards the Future Internet," in Proc. of Workshop on Overlay and Network Virtualization, May 2009. Article (CrossRef Link)

[2] N. M. M. K. Chowdhury, "Network Virtualization: State of the Art and Research Challenges," IEEE Communications Magazine, vol. 47, no. 7, pp. 20–26, Jul. 2009. Article (CrossRef Link)

[3] A. Fischer, J. F. Botero, M. T. Beck, H. Meer, and X. Hesselback, "Virtual Network Embedding: A Survey," IEEE Communications surveys & Tutorial, Vol. 15, Iss., 4, pp. 1888-1906, Feb. 2013. Article (CrossRef Link)

[4] M. Yu, Y. Yi, J. Rexford and M. Chiang, "Rethinking virtual network embedding: substrate support for path splitting and migration," ACM SIGCOMM Computer Communication Review, vol. 38, Iss. 2, Apr. 2008. Article (CrossRef Link)

[5] N. M. M. K. Chowdhury, M. R. Rahman and R. Boutaba, "Virtual Network Embedding with Coordinated Node and Link Mapping," in Proc. of INFOCOM, pp. 783-791, Apr. 2009. Article (CrossRef Link)

[6] X. Cheng, S. Su, Z. Zhang, H. Wang, F. Yang, Y. Luo, and J. Wang, "Virtual Network Embedding Through Topology-Aware Node Ranking," ACM SIGCOMM Computer Communication Review, vol. 41, No. 2, Apr. 2011. Article (CrossRef Link)

[7] A. Fischer, M. T. Beck and H. D. Meer, "An approach to Energy-efficient Virtual Network Embeddings," in Proc. of IM, pp. 1142–1147, May 2013. Article (CrossRef Link)

[8] H. Sugiyama, Y. Fukushima and T. Yokohira, "Performance Evaluation of an Energy Efficient Virtual Network Mapping Method - In the case of load-depending power consumption model -," in Proc. of ICESS, 11 pages, July 2015. Article (CrossRef Link)

[9] OpenStack project website. Article (CrossRef Link)

[10] A. Medina, A. Lakhina, I. Matta, and J. Byers, "BRITE: Universal Topology Generation from a User's Perspective," Technical Report, 47 pages, Boston University, 2001. Article (CrossRef Link)

Yukinobu Fukushima received his B.E., M.E. and Ph.D. degrees from Osaka University, Japan, in 2001, 2003 and 2006. He is currently an assistant professor of the Graduate School of Natural Science and Technology, Okayama University. His research interest includes optical networking, P2P live streaming and cloud computing. He is a member of IEEE, ACM, OSA, and IEICE.

Kohei Sato received his B.E. degree from Okayama University, Japan, in 2016.

Itsuho Goda received his B.E. degree from Okayama University, Japan, in 2015.

Heung-Gyoon Ryu received the B.S. and M.S. and Ph.D. degrees in electronic engineering from Seoul National University in 1982, 1984 and 1989. Since 1988, he has been with Chungbuk National University, Korea, where he is currently Professor of Department of Electronic Engineering in Chungbuk National University. His main research interests are digital communication systems, communication circuit design, spread spectrum system and communication signal processing.

Tokumi Yokohira received the B.E., M.E. and Ph.D. degrees in information and computer sciences from Osaka University, Osaka, Japan, in 1984, 1986 and 1989, respectively. From April 1989 to May 1990, he was an assistant professor in the Department of Information Technology, Faculty of Engineering, Okayama University, Okayama, Japan. From May 1990 to December 1994 and from December 1994 to March 2000, he was a lecturer and an associate professor, respectively, in the same department. From April 2000 to June 2003, he was an associate professor in the Department of Communication Network Engineering of the same faculty. Since July 2003, he has been a professor in the same department. His present research interests include performance evaluation and improvement of computer networks and communication protocols, design algorithm of optical networks and network securities. He is a member of the IEEE Communication Society, the Institute of Electronics, Information and Communication Engineers and Information Processing Society of Japan.