Computer Networks 145 (2018) 118–127

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Noise-induced VNE method for software-defined infrastructure with uncertain delay behaviors

Koki Inoue[a,*], Shin'ichi Arakawa[a], Satoshi Imai[b], Toru Katagiri[b], Masayuki Murata[a]

[a] Graduate School of Information Science and Technology, Osaka University, 1–5 Yamadaoka, Suita, Osaka 565–0871, Japan
[b] Fujitsu Laboratories Ltd., Kamikodanaka 4–1–1, Kawasaki-shi, Kanagawa, 211–8588 Japan

ARTICLE INFO

Article history:
Received 6 January 2018
Revised 17 August 2018
Accepted 4 September 2018
Available online 6 September 2018

Keywords:
Virtual network embedding
Software-defined infrastructure
Yuragi-based VNT control
Software-defined Networks
Network-function Virtualization
Virtual node mapping
Delay profile

ABSTRACT

Software-defined infrastructure (SDI) provides virtualized infrastructures to customers by slicing computing resources and network resources. One of the important problems for deploying an SDI framework is to control the assignment of physical resources to a virtual network against changes of traffic demand and service demand. For this problem, the virtual network embedding (VNE) problem, which maps a virtual network to physical resources, has been addressed, but a centralized calculation was assumed. It is difficult to adopt the centralized approach as the size of infrastructure increases and the number of VN requests increases because the identification of current demand becomes more complicated. In this paper, we present a VNE method that works with limited information for large, complicated, and uncertain SDI frameworks. To achieve this, our method applies the biological "Yuragi" principle. Yuragi is a Japanese word whose English translation is "a small perturbation to the system." Yuragi is a mechanism that provides adaptability of organisms and is often expressed as an attractor selection model. This paper develops a Yuragi-based VNE method that deals with node attributes, has the generality to handle a performance objective, and runs in multi-slice environments. Simulation results show that the Yuragi-based method decreases VN migrations by about 29% relative to a heuristic method to adapt to fluctuations in resource requirements.

1. Introduction

Information networks are faced with new emerging services, such as mobile services, cloud computing services, and social services. Software-defined infrastructure (SDI) enables rapid deployment of new services on information networks and/or information systems by providing virtualized infrastructure to customers by slicing computing resources and network resources.

In an SDI framework, thanks to the advance of virtualization technologies combined with software technology, customers order resource by making requests to service providers and the sliced virtualized resource is immediately assigned to the requesting customer.

A key to leveraging an SDI framework is network virtualization technologies and their control. Network virtualization technologies are in the research and development phase. In recent years, software-defined networking (SDN) and network-function virtualization (NFV) technologies have been expected to replace the conventional network management systems, and standardization of SDN/NFV technologies is being promoted. SDN/NFV technologies enable programmable and automated network control, while conventional systems require the network operator to configure various kinds of network devices [1–5]. That is, SDI frameworks realized by SDN/NFV technologies have the potential to support rapid and flexible deployment of services, such as on-demand resource allocation, self-service provisioning, and secure cloud services [2].

Although SDN/NFV technologies and their standardization are important for deploying SDI, another important problem is to control the assignment of physical resources to a virtual network under changes in traffic demand and service demand. For this problem, the virtual network embedding (VNE) problem has been addressed [6–12]. The VNE problem is a placement problem in which virtual resources are to be allocated to the physical network with optimization of some performance objectives. In the VNE problem, service demands from customers are translated to virtual network (VN) requests. A VN consists of virtual nodes and virtual links. Each of the virtual nodes is hosted on a physical node as a form of virtual machine. Then, the virtual nodes are connected through a path of physical nodes, forming virtual links.

* Corresponding author.
E-mail addresses: k-inoue@ist.osaka-u.ac.jp (K. Inoue), arakawa@ist.osaka-u.ac (S. Arakawa), imai.satoshi@jp.fujitsu.com (S. Imai), torukata@jp.fujitsu.com (T. Katagiri), murata@ist.osaka-u.ac.jp (M. Murata).

https://doi.org/10.1016/j.comnet.2018.09.004
1389-1286/

The VNE problem is divided into two sub-problems: virtual node mapping and virtual link mapping. Virtual node mapping decides the location of the physical node for each virtual node. Note that each virtual node must be allocated to a physical node supporting its "node attribute." The node attribute allows classification of nodes in ways defined by the supported operating system (OS), storage type, or node use (e.g., computing, storage, or packet switching). Virtual link mapping decides the path on the physical network for virtual links between virtual nodes.

In [9–12], a centralized calculation was assumed to solve virtual node mapping and virtual link mapping. That is, a centralized component gathers traffic information and resource utilization for each VN and identifies the current situation (i.e., the current traffic demand and/or the current service demand) of the networks. Then, the component solves the optimization problem that optimizes some metric, such as maximizing revenue or minimizing resource utilization. However, when the network size gets larger and the number of multiplexed VNs increases, the identification of the current situation becomes complicated by the enormous amount of network information. As the network operators want to know the current situation more accurately and precisely, more information is necessary to collect. This will lead to increased used of link bandwidth, increased delay, and a bottleneck on network scalability [5]. Note that the calculation time to obtain a solution of the optimization problem also gets larger. However, the calculation time is not crucial because it may be relaxed by some heuristic algorithms with some sacrifice of the quality of the solution. Our concern in adopting the centralized approach is the overhead of collecting information, and this overhead gets larger as the size of the infrastructure and number of VN requests increase. Moreover, the environments surrounding the Internet today are continuously changing, thus, adaptive control of VNE is required to handle uncertain changes in the environments. Although precise modeling of the end-to-end delay in SDI environment is difficult, it would be required to suppress the maximum delay in order to guarantee a specific quality of experience (QoE) for applications on VNs. There are several models of network delay proposed, which are constructed generally and disregard the data contexts of packets [13]. However, the processing delay on servers depends on multiple factors, including server specification; CPU and memory utilization (on virtual machines); and details of processing, which depend on the context of the data.

In this paper, we present a VNE method that works with only a little information for large, complicated, and uncertain SDI frameworks. To achieve this, the proposed method applies the biological "Yuragi" principle. Yuragi is a Japanese word whose English translation is a small perturbation, both externally and internally generated, to the system. Yuragi is a mechanism that provides adaptability to organisms and is often expressed as an attractor selection model. Our research group has developed a virtual network control based on attractor selection for optical networks. Our results showed that our control mechanism has high adaptability to environmental fluctuations with restricted information. Unlike a virtual network on an optical network, a virtual network on an SDI framework has to consider various matters such as node attribute, computational performance of servers, and VN multiplexing. Therefore, this paper develops a Yuragi-based VNE method that deals with node attributes, has the generality to set a performance objective, and runs in multi-slice environments. One process of the method is executed for each VN slice, and each process needs information about only its own VN requests. Each of the processes behaves so as to improve its own performance function, considering other VNs as a part of an external perturbation (i.e., Yuragi). We have presented a preliminary version of this work in [14] and have demonstrated the basic behavior of the Yuragi-based VNE method with a simple queueing model for delay behavior. However, delay behavior is more complicated and difficult to identify in SDI. Thus, the system needs to operate under uncertain situations. In this paper, we examine a more complicated model of end-to-end delay and show that the proposed method can sustain its adaptability when several delay behaviors are present.

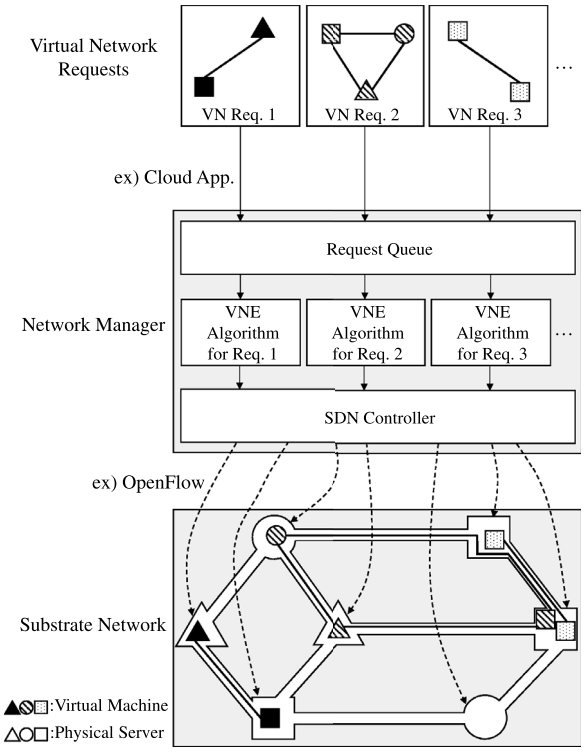


Fig. 1. Service model in software-defined infrastructure.

The rest of this paper is organized as follows. In Section 2, a service model for SDI frameworks is introduced and related works on VNE are referenced. The method based on the Yuragi principle is proposed in Section 3, and the results of performance evaluation are shown in Section 4. Finally, the conclusion of this paper and future work are presented in Section 5.

2. Virtual network services in SDI frameworks

In this section, we describe SDN frameworks and explain a service model for SDI frameworks. First, a whole system of the virtual network service is explained. Next, VNE, one of the important problems for SDI service, is described.

2.1. SDI

Fig. 1 shows a service model of SDI frameworks. In the model, customers request a VN from their service providers. The VN request includes topology information, which is a set of virtual nodes and virtual links. Then, the provider assigns computing resources for the virtual nodes by preparing virtual machines. Then, the provider configures the packet forwarding rules on the network switches via SDN controller to form virtual links.

The customers can specify the performance and capacity requirements, such as the CPU power of a virtual node and the bandwidth of a virtual link. They may also specify memory capacity (RAM), storage capacity (HDD), and in some cases, specify the detail of restrictions: the operating system (OS) of the virtual machine, the RAID type of storage, and the RAM type of a switching device. We call these specifications of virtual nodes the "node

attributes". Note that node attributes do not correspond one-to-one to server resources but do correspond one-to-one to a combination of server resources and some specific constraints. For example, node attribute *A* might express a requirement for a high-performance computing server (with a high number of CPUs and a large amount of memory), attribute *B* might express the need for a cloud file server (with big storage disks), and attribute *C* might express the need for several kind of servers with other specific constraints (e.g., required some geographical restriction).

The service provider has a network manager to handle VN requests. The network manager plays three roles. First, the network manager receives VN requests from customers and pushes them into a queue. Second, the network manager executes a certain VNE algorithm for each VN request in the queue in first-in-first-out (FIFO) order. The VNE algorithm decides a VN mapping (i.e., a virtual node mapping and virtual link mapping). Virtual node mapping decides the location of the physical node for each virtual node. Then, the virtual node is hosted on the physical node as a virtual machine. Virtual link mapping decides the path on the physical network for virtual links between virtual nodes. Then, the virtual nodes are connected through the path. When the VNE algorithm fails to find a VN mapping due to a shortage of physical resources, the VN request is rejected. Next, the network manager offers the mapping request to the SDN controller. Note that the SDN controller might be managed by other organizations, such as infrastructure providers, rather than the service provider. Then, the service provider installs virtual machines into physical servers and allocates the requested computing resources. Then, the SDN controller accesses the substrate nodes via some protocol (such as OpenFlow) and reconfigures the forwarding rules to establish the virtual links.

2.2. The virtual network embedding problem

VNE is one of the important problems in allocating physical resources in response to a VN request. The physical resources, including resources of the physical network and resources of physical servers, form a substrate network. OpenStack, which is one of the most general infrastructure-as-a-service (IaaS) frameworks, defines virtualized resource components [10]. The substrate node is classified into three types: computing servers, network switches, and storage. Each virtual node may have individual features, such as supported OS, protocols, and storage types. It is necessary to strictly check the consistency of the node features when embedding a virtual node to a substrate node. That is, the requested features of the virtual node must be supported by the substrate node. To simplify the service model, this paper abstracts the classifications of features of OpenStack into "node attributes."

The mapping of the virtual network has an effect on many aspects, such as resource utilization, blocking rate, revenue, QoE, energy efficiency, and migration cost. That is why the VNE problem deserves consideration. Fig. 2 shows an illustrative example of how the experienced delay of VNs differs depending on the mapping of the virtual network. Fig. 2a shows a substrate network including resource capacities. Fig. 2b shows VN requests including resource requirements. The numerical values $c(\cdot)$ and $d(\cdot)$ in the figure represent the number of CPUs on the node and the bandwidth of the link, respectively. Fig. 2c and d show two patterns of VN mapping, denoted as mapping *A* and *B*, respectively. In general, the delay of a server is longer when the CPU utilization is higher, and the delay of a link is longer when the link utilization is higher. In the case of mapping *A*, the CPU utilization on one of the substrate nodes reaches 80% and the calculation delay gets longer. However, in the case of mapping *B*, the CPU utilizations are at most 50%. As for the delay on a link, the maximum link utilization of the substrate link is 90% in the case of mapping *A*. In the case of mapping *B*, the link utilizations of the substrate links are low, and so no additional delay will be introduced. Therefore, the experienced delay under mapping *B* is expected to be shorter than that under mapping *A*. Thus, between the two mappings, mapping *B* is the preferred solution of the VNE problem. Generally, the VNE problem is divided into two phases: virtual node mapping (VNoM) and virtual link mapping (VLiM). The goal of VNoM is to obtain a matching between virtual nodes and substrate nodes under the constraint that the substrate node must support the node attribute of the matched virtual node. The goal of VLiM is to obtain a set of links in the substrate network that connects one virtual node to another virtual node.

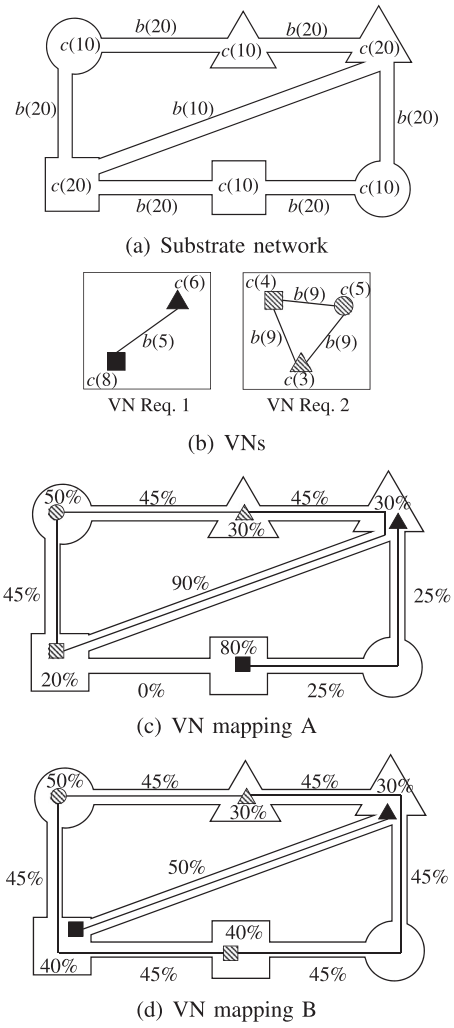


Fig. 2. Comprehension of VNE problem with a simple example.

2.3. Centralized approaches for VNE

A number of approaches to coping with VNE problem have been proposed. Most of them try to formulate and solve optimization problems and maximize/minimize some performance objectives. However, existing VNE formulations typically use integer linear programing (ILP), and the VNE problem is known to be an $\mathcal{NP}$-hard problem. Thus, some heuristic methods are also developed. Note that both the ILP methods and heuristic methods assume information of the network is collected in advance.

Chowdhury et al. deal with VNE problem of embedding multiple VN requests onto a substrate network [9]. They give a formulation as mixed integer linear programing (MILP) to minimize

embedding cost while achieving a balance of resource utilization. Guerzoni et al. formulate a MILP that considers node attributes to maximize the revenue while minimizing resource utilization [10]. Chen et al. present a virtual node mapping method to optimize energy efficiency, and also propose a heuristic algorithm for this [11]. Fajjari et al. minimize the running cost of the network infrastructure by releasing the unused bandwidth of a VN for other VNs [12].

To handle the VNE problem, it has been widely considered to take optimization approaches such as ILP and its heuristic methods. It is expected that those methods will give the solution with the best objective function value. However, to compute the best performance, these optimization methods examine the detailed situation of the whole infrastructure, and in the worst cases, the network will be congested with an increasing volume of traffic related to control messages for collecting the details of the situation [5]. The overhead of gathering such global information becomes a fundamental limitation to adopting the optimization approach in SDI because the orchestrater needs to manage a huge number of multiplexed VNs and highly dynamic requests. To avoid this problem, control methods driven by a small amount of knowledge of the situation are required.

3. Yuragi-based virtual network embedding method

This section proposes a Yuragi-based VNE method for SDI frameworks. The Yuragi principle, which is often called an attractor selection model, explains the biological adaptability. The key concept of attractor selection models is that systemic behavior is governed by a single value, called “activity,” and a small perturbation, which we call “Yuragi”. The activity is a kind of “comfortableness” for the system, and via feedback of the activity and small perturbations, the control state of the system falls into a comfortable state. When activity is high, the control state of the system is in a good condition and stays in that state. Such an equilibrium point is called an “attractor”. When activity becomes low or the condition becomes uncomfortable due to environmental changes, the system gets out of the attractor, i.e., escapes the basin of attraction (hereinafter, the attractor structure), and then looks for another attractor via feedback of the activity and small perturbations.

The proposed VNE method is expected to enjoy the adaptability of Yuragi to environmental changes. That is, VN migrations are driven according to experienced performance and the new VN mapping is obtained by means of attractor selection. A process of the Yuragi-based method is executed for each VN request. Thus, multiple processes are executed in parallel to deal with multiple VN slices. Different from optimizing problems and related heuristics, the Yuragi-based method can avoid the necessity of collecting detailed information about the entire network. The process for a VN request needs only enough information for comfortableness and does not need any information related to other VN requests.

3.1. Yuragi principle

The Yuragi principle is the principle that biological organisms use to adapt to environmental fluctuations. Attractor selection is a model that represents the Yuragi principle. The model describes the dynamics of state variables x_i $(i = 1, 2, \ldots, n)$ through environmental fluctuations as

$$\frac{d\mathbf{x}}{dt} = \alpha \times f(\mathbf{x}) + \eta, \tag{1}$$

where $\mathbf{x} = (x_1, \ldots, x_i, \ldots, x_n)$ represents the system state, activity α is the comfortableness of the present system state, $f(\mathbf{x})$ defines deterministic behavior governed by the attractor structure, and η represents stochastic behavior. When the system is in a comfortable state, and hence activity α is high, the deterministic term $f(\mathbf{x})$ controls the dynamics while the noise η is almost negligible. When the system condition gets worse and α gets close to zero, $f(\mathbf{x})$ is no longer influential and the stochastic term η becomes relatively dominant. Therefore, the system changes its state at random and searches for another attractor. Once the system reaches an attractor with a comfortable activity level (though not necessarily the best possible activity level), the system will stay in the new good state. When the system reaches a state with a high activity that has not been defined by the attractor structure $f(\mathbf{x})$, the system also stays in the state and $f(\mathbf{x})$ is reconstructed to register the state as a new attractor.

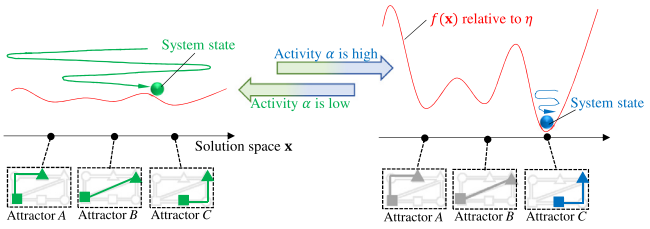


Fig. 3. An illustration of the Yuragi mechanism.

A system driven by the Yuragi principle achieves adaptability to environmental changes. The adaptability has two aspects. First, the system is robust to small fluctuations in the surrounding environment. As long as activity remains higher than a certain level, the system keeps staying at an equilibrium point even though the noise term is still present (see the right-hand side of Fig. 3). Second, the system has flexibility in responding to drastic changes in the environment. When the system falls into an uncomfortable state, the activity decreases immediately and the dynamics of the system behavior escapes from the attractor structure (see the left-hand side of Fig. 3).

3.2. Performance objectives

We can select various definitions of the activity when the Yuragi principle is applied to the VNE problem. The Yuragi-based VNE method tries to find a system state that maintains high activity. The high activity should be designed so that the performance objective does not violate a required threshold.

Conventional works usually consider link utilization [15] and/or energy consumption [16] as performance objectives because these can be described as a linear function of traffic load. Note that linearity in a mathematical sense is one of the key factors to solving the optimization problem. In this paper, we focus on experienced delay, which is the end-to-end delay on the VNs, consisting of the communication delays between VMs and processing delays on VMs. Experienced delay is thus not necessarily a linear function of the performance objective (comfortability) because experienced delay is one of the simplest and most fundamental performance objectives in networking. It is true that link utilization is often used as the performance objective of VN control. However, experienced delay is a more important measure in networking and is especially important for SDI frameworks. A longer delay can cause considerable degradation in QoE of an application running on the VN. Thus, customers of SDI services want to require a low delay to the infrastructure provider. Nevertheless, conventional approaches usually have to minimize utilization or workload instead of delay for the objective function because of the difficulty of modeling the experienced end-to-end delay, which is composed of complicated factors. Moreover, under virtualization environments, the delay is caused by not only utilization of the network bandwidth but also workloads on the VMs, and the delay becomes extremely long under heavy workloads [17,18]. The end-to-end delay in SDI frameworks comprises delays in networks and de-

lays on servers. The processing delays on servers depend on multiple factors. Thus, exact analyses and estimation of the software processes are indispensable for calculating optimization problems, but these are difficult in general [19]. Even when we have a good model, the network manager may deal with non-linear optimization problems that are tough to solve even by offline computation. Therefore, it is difficult to deal with delay requirements in conventional approaches. Instead of applying optimization with some sort of delay model, an online control approach is needed. The online approach measures the actual delay continuously. When the measured delay does not satisfy requirements, the network manager reconstructs the VN mapping immediately. In this way, the online approach avoids calculation of complicated optimization problems. Of course, the network manager must obtain a VN mapping solution quickly enough to control the VN. In this paper, we consider the end-to-end delay with applying the Yuragi principle, and confirm by simulation (with a topology of 50 nodes) that the calculation of the proposed Yuragi-based method terminates within a few seconds.

3.3. Yuragi-based VNE method

This section explains our Yuragi-based VNE method. Our proposed method consists of two phases: attribute-aware virtual node mapping and shortest-path virtual link mapping. The relation between state variables in the Yuragi principle and the VNE problem are explained first.

Our method decides where to allocate a virtual node with attribute k. In other words, the method finds a coupling between attribute k and physical node n. Let the number of attributes be K and the number of physical nodes be N. We prepare variables $\mathbf{x} = (x_1, \ldots, x_{kn}, \ldots, x_{KN})$. A variable x_{kn} is a decision variable that designates whether physical node n is a candidate for virtual node with attribute k. Then, the dynamics of each x_i $(i = 1, 2, \ldots, KN)$ is described as

$$\frac{dx_i}{dt} = \alpha \left\{ \varsigma\left(\sum_j W_{ij} x_j \right) - x_i \right\} + \eta, \tag{2}$$

where $\varsigma\left(\sum_j W_{ij}x_j\right) - x_i$ represents a deterministic term and η is a stochastic term. In the first term, the matrix $\mathbf{W}$ represents an attractor structure (discussed later). The function $\varsigma(z)$ is a sigmoid function defined as

$$\varsigma(z) = \tanh(\frac{\mu}{2} z), \tag{3}$$

where μ represents the gradient in the vicinity of the threshold. Here, the threshold is 0, and the output value of $\varsigma(z)$ gets close to 1 or -1. Note that the range of x_i is $[-1, 1]$. The second term η in Eq. (2) is a random value following a normal distribution. If $x_i > 0$ and i's corresponding node (resp., attribute) is n (resp., k), then physical node n is a candidate for a virtual node with attribute k. If $x_i\ (= x_{kn}) < 0$, the virtual node with attribute k is not embedded to physical node n. Each of the virtual nodes with attribute k is allocated onto one of the candidate nodes in descending order of x_{k^*} values. Note that, when physical node n is not compatible with attribute k due to the attribute restriction, x_{kn} is set to 0 without calculating the differential equation (2).

Finally, our method assigns the shortest path for each virtual link request. In this paper, we consider shortest-path routing to minimize hop length on the physical topology. Other routing policies can be applied, but this is not examined in the evaluation in Section 4.

3.3.1. Activity function with performance profile

Activity α is feedback from the system and reflects the comfortableness of the VN. Let p be an objective metric, expected to be small. Activity is described as,

$$\alpha = \frac{\gamma}{1 + \exp(\delta(p - \theta))}, \tag{4}$$

where γ represents the scale of the activity value and δ represents the gradient around the threshold θ. Let γ be 1, to which the activity value gets close if $p < \theta$. Otherwise, the activity becomes 0. Note that the activity is subject to be reduced to 0 regardless of Eq. (4). Letting V_k be the number of virtual node requests with attribute k, the activity α is reset to be 0 when the number of candidates $|x_{k^*}|(s.t. x_{k^*} > 0)$ is less than V_k. This is necessary because the system state found by Yuragi does not have a sufficient number of candidate nodes. Also, when the available capacity of a physical resource is not enough to embed the found system state, α is forced to 0.

In our method, the objective metric p can be directly monitored. However, when the monitoring incurs some overhead or it is difficult to monitor p directly, the activity should be calculated by estimating p rather than finding p exactly. For the estimation, we consider making use of a performance profile. The profile database consists of the correspondences between delay and resource utilization based on a history and is maintained in some form (typically, as a table).

3.3.2. Attractor structure

The matrix $\mathbf{W}$ in Eq. (2) represents an attractor structure. It stores some equilibrium points of a virtual node mapping, and the equilibrium point is called an attractor. Each attractor is defined as $\mathbf{y} = (y_1, \ldots, y_i, \ldots, y_{KN})$, where $y_i \in \{-1, 0, 1\}$. If physical node n is one of the candidates for a virtual node with attribute k, then y_{kn} is set to 1. If node n cannot allocate attribute k due to the node attribute restriction, then y_{kn} is set to 0. Otherwise, y_{kn} is set to -1. Letting M be the number of attractors stored in $\mathbf{W}$, a set of attractors $\mathbf{Y} = (\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_M)$ can be stored by

$$\mathbf{W} = \mathbf{Y}^{+}\mathbf{Y}, \tag{5}$$

where $\mathbf{Y}^{+}$ is the pseudo inverse matrix of $\mathbf{Y}$. This way of storing attractors uses the knowledge of Hopfield neural network of associative memory [20]. When the present state is in one of the attractors, $d\mathbf{x}/dt$ in Eq. (2) becomes close to 0 and stays in the attractor.

3.4. VN calculation

When the activity gets extremely low, that is, when the observed end-to-end delay exceeds the performance objective value θ, the network manager executes then Yuragi-based VNE method in an offline calculation by using a performance profile. The performance profile enables estimating performance without running the services on actual infrastructure. Thus, the service continues to run with the extant VN while calculating a new VN. Once the Yuragi system converges to a good VN mapping, the network manager is ready to enter the VN migration phase.

3.5. VN migration

The network manager migrates each VM that needs to be transferred for a new VN mapping. The process is executed according to the "make-before-break" principle to reduce service downtime:

Step 1. Copy the VM image from the source node to the destination node. Note that the service is still running on the source node.
Step 2. Boot a VM on the destination node.
Step 3. Copy the state differences between VMs (typically implemented as "dirty pages") to the destination node recurrently.

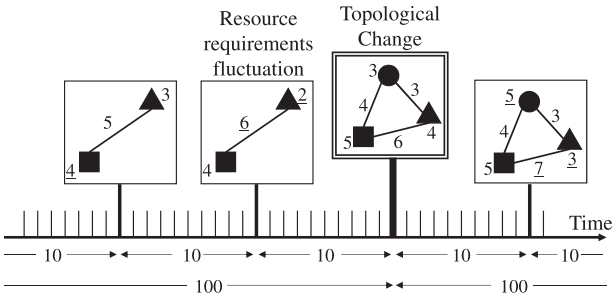


Fig. 4. Environmental fluctuations over elapsed time.

Step 4. Suspend the VM at the source node and copy the remaining state differences from source to destination.
Step 5. Switch the traffic flow to the destination node and resume the VM on the destination node. Then, the service is running on the destination node.
Step 6. Delete the VM on the source node.

The service may be suspended during the time to copy the state differences in Step 4. Thus, the service downtime is shortened to only hundreds of milliseconds [21]. Note that, the system needs to make particular provision to guarantee the user experience of specific types of applications, such as more real-time oriented services (e.g., as voice or video).

4. Evaluation by computer simulation

This section presents the results of evaluating the Yuragi-based VNE method by computer simulation.

4.1. Simulation environment

The substrate network consists of physical servers and links. The number of physical servers (physical nodes) is 50. Each node has the capability to host virtual nodes with one of the node attributes. In this environment, each node has three kinds of resource capacities for required virtual machines: CPU, memory and storage capacity. These are determined uniformly within [50, 100] for each node. For each pair of physical nodes, a physical link is randomly established between the nodes with probability 50%. As a result, we obtained a physical topology with 50 nodes and 617 links. The (integer) capacity of physical links is determined uniformly randomly within [50, 100]. During the simulation, the substrate network is fixed.

Several requests of virtual network are generated and arrive. During the simulation, the number of VN requests is set to 20 and the number of node attributes K is set to 4. Each VN request is generated as follows. The number of virtual machines (virtual nodes) is determined uniformly randomly within [2, 5]. Each virtual node belongs to an attribute, and each virtual node requires capacities for CPU, memory, and storage. Each of the required capacities is determined uniformly randomly within [1,10]. Virtual links are undirected, and each pair of virtual nodes is randomly connected through a virtual link with probability 50%. The number of virtual links is within [1,10] because the number of virtual nodes is 2–5. Each virtual link has a required bandwidth, and the required capacity is determined uniformly randomly within [1, 25].

Every 100 time steps, all 20 VN requests are regenerated in the same way as described above. In addition, at every 10 time steps, each VN request fluctuates with relatively small changes: we change the requested capacities by a random integer, which is obtained by rounding a value following the normal distribution with $\mu = 0$ and $\sigma^2 = 1$ (see Fig. 4). The service downtime caused by VN migration is regarded as negligible in the following simulation.

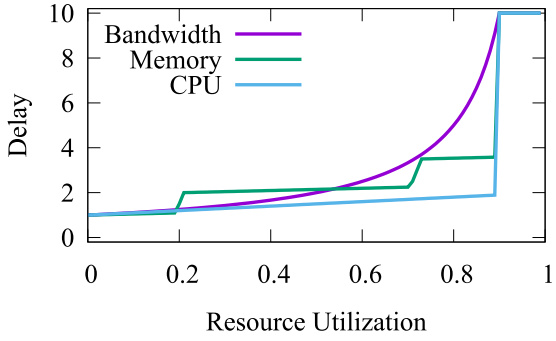


Fig. 5. Delay models used for computer simulation.

Table 1 summarizes the parameters in the simulation environment.

4.2. Delay profile

We use end-to-end delay as the objective metric. In an actual environment, the experienced delay may be available by monitoring of packet arrivals. However, when the monitoring incurs some overhead or it is difficult to monitor directly for some reason, the activity should be calculated by an estimated p rather than actually measuring p. For the estimation, we consider the use of a performance profile. For the performance profile, we prepare several delay models as a function of resource utilization, referring to bandwidth on links and the tuple (memory, CPU, storage) on servers. Fig. 5 shows the delay models. The first model simulates delay caused by bandwidth utilization in a link. A basic M/M/1-based model of delay in networks is used. The second delay model simulates memory utilization on the server and imitates the response time of an Apache web server. As shown in Fig. 5, the second model is characterized by the multi-stage elevations of delay. In the case of web service, such elevations are caused by swapping memory pages and storage disks. The last model simulates CPU utilization. The delay increases linearly as the resource utilization becomes high, except at extremely high utilization, where the delay increases rapidly. Such an increase in delay is caused by conflicts among VMs demanding more calculation power than the CPUs can provide. In the following simulation, d_{ij}, which represents the delay from virtual node i to virtual node j, is calculated as,

$$d_{ij} = w_c \sum_{n \in R_{ij}} d_n^c + w_m \sum_{n \in R_{ij}} d_n^m + w_s \sum_{n \in R_{ij}} d_n^s + w_b \sum_{l \in L_{ij}} d_l^b, \tag{6}$$

where the set R_{ij} consists of the physical nodes along the route from i to j, and the set L_{ij} consists of the physical links. Then d_n^c, d_n^m, and d_n^s are the computing delay in virtual machine n according to CPU, memory and storage, respectively. In this, d_l^b is the delay through physical link l, and w_c, w_m, w_s, and w_b are weight parameters. Each d_n^* and d_l^b follows the delay model and has a value calculated by its own utilization of physical resources.

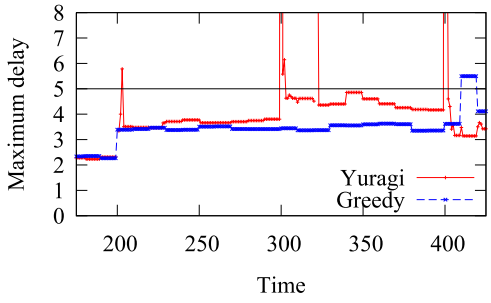
Network managers maintain delay profiles in which correspondences between resource utilization and actually measured delays are recorded. Referring to a delay profile makes it possible for the manager to estimate delays in the VN when consider a VN request to be embedded. In the simulation environment, delays are calculated with the same delay models as the delay profile. Note that, in actual usage, the delay can be easily obtained by referring to the timestamps of packets.

4.3. Heuristic method for comparison

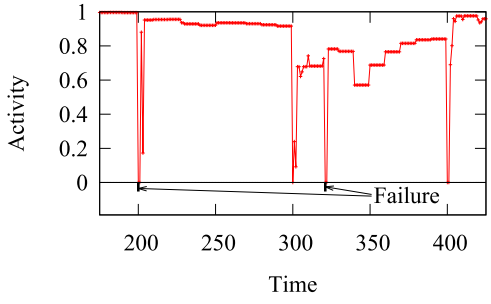
As for the benchmark of our method, a heuristic VNE method is also simulated in the same environments. The heuristic method

Table 1
List of variables and values in the simulation.

Variable	Value
Substrate network:	
- Number of nodes N	50
– Number of attribute K	4
– CPU number	[50, 100]
– Memory capacity	[50, 100]
– Storage capacity	[50, 100]
- Number of links	617
– Link capacity	[50, 100]
VN requests:	
- Number of VN	20
Each VN request:	
- Number of nodes	[2, 5]
– CPU number	[1, 10]
– Memory capacity	[1, 10]
– Storage capacity	[1, 10]
- Number of links	[1, 10]
– Link capacity	[1, 25]
Delay weight:	
- (w_c, w_m, w_s, w_b)	(0.25, 0.25, 0.25, 0.25) in Section 4.4.1. {(0.6, 0.2, 0.0, 0.20), (0.2, 0.6, 0.0, 0.20), (0.2, 0.2, 0.0, 0.60)} in Section 4.4.2.
Yuragi parameters:	
- $(\mu, \gamma, \delta, \theta)$	(20.0, 1.0, 2.0, 5.0)



(a) Maximum delay



(b) Activity

Fig. 6. Maximum delay and activity on a VN.

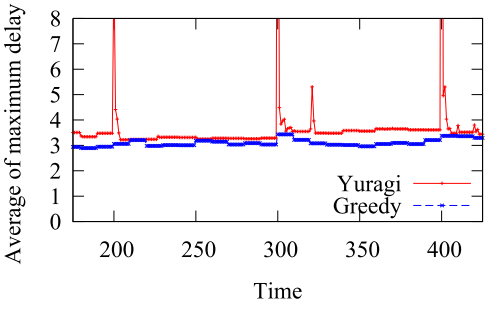


Fig. 7. Average of maximum delay for 20 VNs.

has two phases: virtual node mapping based on a greedy algorithm [22] and virtual link mapping on shortest paths. Note that we do not aim to obtain a better end-to-end delay than that provided by the greedy algorithm. We intend to obtain a reference delay to confirm that our method can find a comfortable state by using the noise-induced search, rather than simply by using low-traffic settings. The VNE method executes the following algorithm for VN requests, acting sequentially.

(1) Execute the following processes for each virtual node v.
  (1.1) Find the set S of physical nodes that accept the attribute of v and have enough unreserved resource capacities to embed v. When S is null, reject the VN request and finish.
  (1.2) Find the physical node that indicates the highest value of H among S, where H is defined as Eq. (7). Then reserve the resources of that physical node.
(2) Embed the virtual nodes according to the reservation taken in (1).
(3) For each virtual link between virtual nodes embedded in (2), find a path that is the minimum hop in physical topology. Embed the virtual links onto the paths. When a shortage of link bandwidth occurs, reject the VN request.

The greedy method aims to minimize the utilization of node and link resources. The heuristic method calculates an available resource indicator H for each physical node n, defined as

$$H(n) = C_n \times M_n \times S_n \times \sum_{l \in L(n)} B_l, \tag{7}$$

and avoids embedding a virtual node onto bottleneck resources. The values C_n, M_n, and S_n represent the available capacity of CPU, memory, and storage, respectively, on physical node n. The set $L(n)$ represents a set of physical links attached to node n, and B_l represents the available capacity of physical link l. The computational complexity is $\mathcal{O}(n \log n)$ for sorting $H(n)$, assuming the shortest path between every node pairs is available in advance.

4.4. Simulation results

In the simulation, the Yuragi-based method calculates the VN mapping at each time step and migrates the VN until the system state converges to an attractor. The threshold of activity θ in Eq.

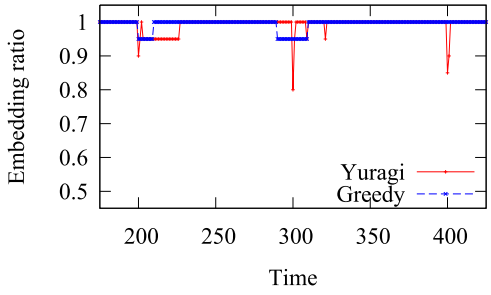


Fig. 8. Embedding ratio of VN requests.

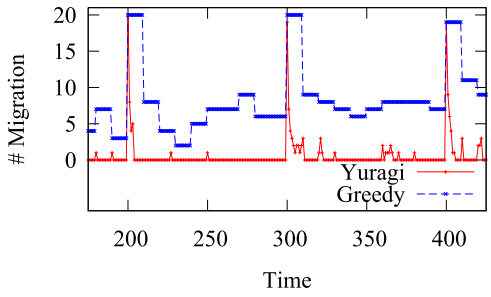


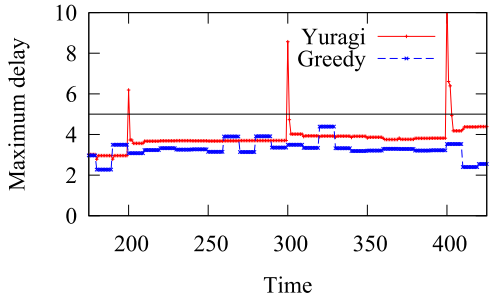
Fig. 9. The number of VN migrations.

(4) is set to 5.0, regarding the metric p as the maximum of d_{ij} for every pair of virtual nodes i and j. The greedy method executes VN migration according to the demand changes at every 10 time steps.

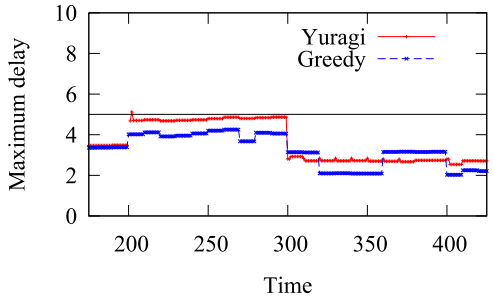
4.4.1. Performance of Yuragi-based VNE method

We first show the performance of Yuragi-based VNE method with a simple M/M/1-based delay model. We explorer its adaptability to fluctuations of VN requests by evaluating the end-to-end delay, the embedding ratio, and the number of VN migrations. Here, the weight values in Eq. (6) are set as $w_c = w_m = w_s = w_b = 0.25$ with an M/M/1-based delay model. We will examine other weight values in Section 4.4.2.

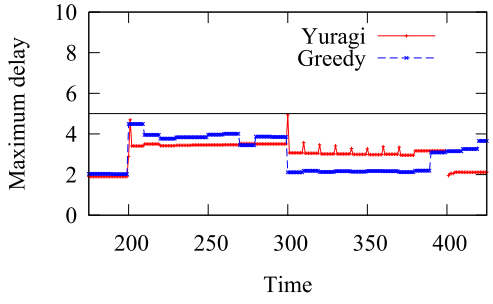
Fig. 6a shows the maximum delay on a VN request out of the 20 requests. The activity of each VN is also shown in Fig. 6b. In the figure, the region denoted as "Failure" represents a failure of embedding the VN caused by a shortage of physical resources or violation of other restrictions. Note that the demands of VN requests fluctuate with relatively small changes at every 10 time steps and the demands fluctuate greatly at every 100 time steps. Thus, the maximum delays for the Yuragi-based method exceed the threshold drastically at every 100 time steps owing to the topological changes in VN requests. The activities drop sharply, and then the VN migration starts. Within a few steps, the activities are recovered and converge to another system state. Against a small fluctuation of required capacities, occurring at every 10 time steps, VN migrations occurs only if the activities decrease sharply as seen, for instance, in time step 320 in Fig. 6a. Fig. 7 shows the mean of the maximum delay of 20 VN requests. Considering the mean of maximum delay of the 20 VN requests, the Yuragi-based method does not achieve a delay as short as that obtained by the greedy method in general. This is because the Yuragi-based method does not aim to minimize the delay or the resource utilization but, rather, to keep them smaller than a certain threshold. Making the threshold smaller might achieve a smaller delay but will result in a longer convergence time for finding an attractor. Fig. 8 shows the embedding ratio, indicating how many VN requests are accepted out of the 20 requests. The topological changes occurring at every 100 time steps cause a temporary decrease in the embedding ratio, but



(a) VN request 1



(b) VN request 2

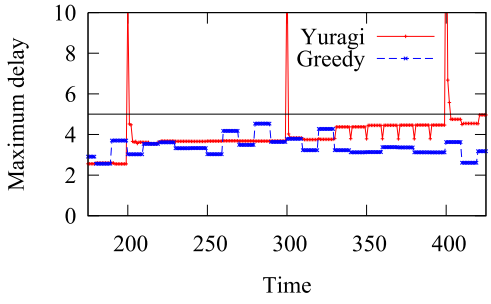


(c) VN request 3

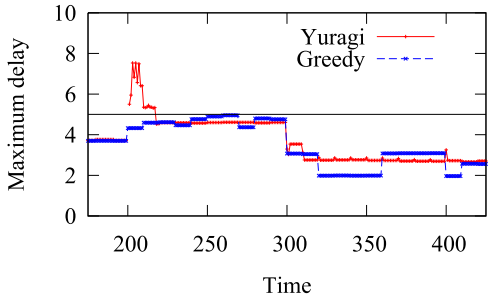
Fig. 10. Maximum delay ($w_c = 0.6, w_m = 0.2, w_s = 0, w_b = 0.2$).

both of the methods keep almost 95%–100% acceptances outside those periods.

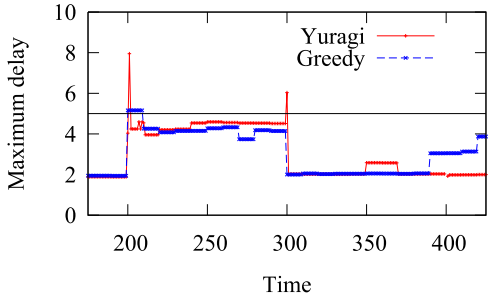
Fig. 9 shows the number of VN migrations, defined as the number of VNs whose location has been changed from the previous operation. Note that the operations are performed at every step by the Yuragi-based method and at every 10 steps by the greedy method. When the VN requests are regenerated at every 100 steps, almost all VNs are migrated for both methods. The simulation result shows that the Yuragi-based method takes fewer VN migrations in response to small fluctuations to maintain the performance objective, and therefore our method costs less in terms of VM migration. The greedy method migrates 2–11 VNs at each change to maintain the required capacity. This is because the greedy method tries to achieve better objective values, even when the improvement in delay is marginal. Note that we may develop a greedy method that requires fewer VN migrations with some additional constraints or considerations. The key point is that the greedy method makes drastic changes due to the nature of the optimization, whereas the Yuragi-based method does not. The total number of VN migrations required by the Yuragi-based method is 153 for the 250 time steps of simulation, and 215 by the greedy method. This result indicates that the Yuragi-based method adapts to demand fluctuations with about 29% fewer VN migrations than the greedy method. For the small fluctuations occurring at every



(a) VN request 1

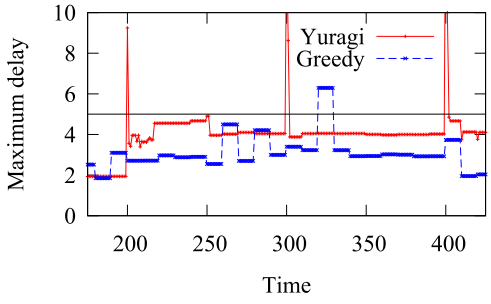


(b) VN request 2

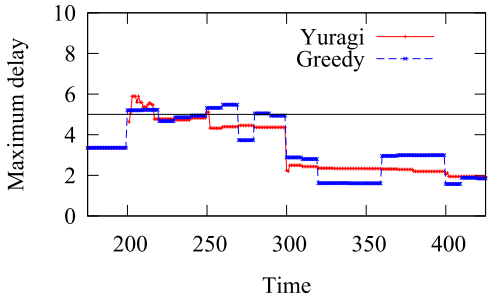


(c) VN request 3

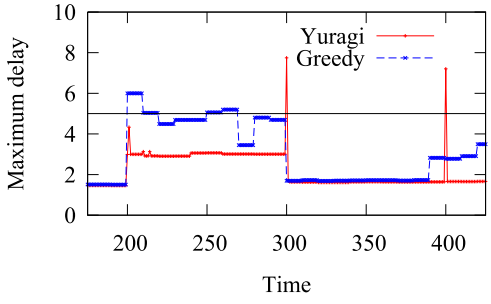
Fig. 11. Maximum delay ($w_c = 0.2, w_m = 0.6, w_s = 0, w_b = 0.2$).



(a) VN request 1



(b) VN request 2



(c) VN request 3

Fig. 12. Maximum delay ($w_c = 0.2, w_m = 0.2, w_s = 0, w_b = 0.6$).

10 time steps, the number of VN migrations with the Yuragi-based method is 33, against 156 with the greedy method.

4.4.2. Adaptability to different delay behaviors

In the previous section, we used the simple M/M/1 delay model where the delays on CPU, memory, and storage are all estimated by their resource utilization. However, in actual SDI environments, the end-to-end delay behaves in a more complicated way, depending on multiple factors such as the processing delay (which also depends on the server specification) and memory utilization on virtual machines. Here, we demonstrate that the Yuragi-based VNE method has adaptability under various types of delay behaviors. We run computer simulations with distinct sets of weight parameters (w_c, w_m, w_s, w_b). The simulation results show that our proposed VNE method can achieve its performance objective even in a situation where the heuristic method fails to obtain acceptable performance.

Figs. 10–12 correspond to the simulation results for parameter sets $(w_c, w_m, w_s, w_b) = (0.6, 0.2, 0.0, 0.2)$, (0.2, 0.6, 0.0, 0.2), and (0.2, 0.2, 0.0, 0.6), respectively. Each figure shows the maximum delays on 3 VN requests out of the 20 requests because similar tendencies are observed on the other VN requests. In Figs. 10–12, the Yuragi-based method mostly keeps the maximum end-to-end delay lower than the threshold of 5.0. Note that sharp increases of the end-to-end delay are observed at time steps 200, 300, and 400, but these delays are not crucial because they are caused by the change of VN request. The Yuragi-based VNE method gradually adapts to the VN requests and soon finds a good VN mapping.

The greedy method sometimes violates the performance threshold in response to some VN request fluctuations (see, for example, at time step 320 on VN request 1 in Fig. 12). The greedy method, a heuristic for optimization, does not always achieve the lowest end-to-end delay. More importantly, the violation cannot be solved and may continue for a while because the greedy method has already optimized its objective and has no way to improve the performance. Those violations of the threshold occur due to a gap between estimated delays and actual delays (i.e., due to the lack of a precise delay model). Note that optimization approaches (including heuristic approaches) will always have such gaps unless they use a precise delay model. Especially in an SDI framework, defining a delay model to decrease the gap gets more difficult because the model depends on complicated factors.

As for the Yuragi-based method, it shows its adaptability under uncertain delay behaviors. An advantage of the method is that it finds a VNE solution with direct measurements of end-to-end delay, where the models of delay behaviors are not used in our method and thus no longer necessary. The delay models used in the simulations may not completely imitate actual delay profiles,

but we believe that the Yuragi-based method is feasible even when the actual end-to-end delay behaves in more complicated or non-deterministic manner.

5. Conclusion

This paper presented a VNE method based on the Yuragi principle as applied to SDI frameworks. A system driven by the Yuragi principle achieves adaptability to environmental changes, and the dynamics is described as an attractor selection model. In attractor selection models, the system behavior is governed by an activity measure and small perturbations. When activity is high, the control state of the system is in a good condition and stays in that state. When activity becomes low or the condition becomes uncomfortable due to environmental changes, the system looks for another stable state. The Yuragi-based VNE method decides the mapping of virtual nodes by means of attractor selection, where the network mapping is regarded as the system state and the activity is defined as a certain performance objective. The end-to-end delay in SDI frameworks depends on application processes and other factors. That makes it difficult to pre-estimate experienced delay accurately and causes degradation of VNE control performance. Nevertheless, our Yuragi-based method shows its adaptability under such uncertain delay conditions. In the evaluation, we considered the end-to-end delay as the activity. Simulation results show that the method provides shorter delays and adapts to the request fluctuations by rearranging the VN mapping in response to drastic changes in environments. The Yuragi-based method decreases VN migrations by about 29% relative to a heuristic method to adapt to fluctuations in required resource capacities.

In future work, we will investigate a method of constructing the attractor structure to improve the convergence time or some other performance measure. We suppose that our proposed method is performed in a centralized SDN controller. Recently, distributed controllers for a single infrastructure are being studied toward wide-area SDN and large-scale SDN. It is worth studying how our noise-induced method can be extended to account for mutually interfering situations. We should also demonstrate the behavior of our proposed method in real implementation. Our method will cause a delay in the SDN controller, which is not included in the computer simulations. It is worth analyzing the impact of executing our method.

Acknowledgment

This research was supported in part by a Grant-in-Aid for Scientific Research (A) (No. JP15H01682) from the Japan Society for the Promotion of Science (Japan Society for the Promotion of Science).

References

[1] C. Filsfils, S. Previdi, B. Decraene, S. Litkowski, R. Shakir, Segment routing architecture, Internet draft `draft-ietf-spring-segment-routing-06.txt` (2015). Work in progress

[2] A. Hakiri, A. Gokhale, P. Berthou, D.C. Schmidt, T. Gayraud, Software-defined networking: challenges and research opportunities for future internet, Comput. Networks 75, Part A (2014) 453–471. https://doi.org/10.1016/j.comnet.2014.10.015.

[3] B. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, T. Turletti, A survey of software-defined networking: past, present, and future of programmable networks, IEEE Commun. Surv. Tut. 16 (3) (2014) 1617–1634.

[4] P. Bhaumik, S. Zhang, P. Chowdhury, S.S. Lee, J. Lee, B. Mukherjee, Software-defined optical networks (SDONs): a survey, Photonic Network Commun. 28 (1) (2014) 4–18.

[5] S. Sezer, S. Scott-Hayward, P. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, N. Rao, Are we ready for SDN? Implementation challenges for software-defined networks, IEEE Commun. Mag. 51 (7) (2013) 36–43.

[6] X. Cheng, S. Su, Z. Zhang, H. Wang, F. Yang, Y. Luo, J. Wang, Virtual network embedding through topology-aware node ranking, ACM SIGCOMM Comput. Commun. Rev. 41 (2) (2011) 38–47.

[7] J. Lischka, H. Karl, A virtual network mapping algorithm based on subgraph isomorphism detection, in: Proceedings of the 1st ACM Workshop on Virtualized Infrastructure Systems and Architectures, 2009, pp. 81–88.

[8] A. Fischer, J. Botero, M. Till Beck, H. de Meer, X. Hesselbach, Virtual network embedding: a survey, IEEE Commun. Surv. Tut. 15 (4) (2013) 1888–1906.

[9] N. Chowdhury, M. Rahman, R. Boutaba, Virtual network embedding with coordinated node and link mapping, in: Proceedings of IEEE INFOCOM, 2009, pp. 783–791.

[10] R. Guerzoni, R. Trivisonno, I. Vaishnavi, Z. Despotovic, A. Hecker, S. Beker, D. Soldani, A novel approach to virtual networks embedding for SDN management and orchestration, in: Proceedings of IEEE NOMS, 2014, pp. 1–7.

[11] X. Chen, C. Li, Y. Jiang, Optimization model and algorithm for energy efficient virtual node embedding, IEEE Commun. Lett. 19 (8) (2015) 1327–1330.

[12] I. Fajjari, N. Aitsaadi, G. Pujolle, H. Zimmermann, Adaptive-VNE: a flexible resource allocation for virtual network embedding algorithm, in: Proceedings of IEEE GLOBECOM, 2012, pp. 2640–2646.

[13] L.-S. Peh, W.J. Dally, A delay model for router micro-architectures, IEEE Micro 21 (1) (2001) 26–34.

[14] K. Inoue, S. Arakawa, S. Imai, T. Katagiri, M. Murata, Adaptive VNE method based on Yuragi principle for software defined infrastructure, in: Proceedings of IEEE HPSR, 2016, pp. 191–196.

[15] B. Addis, D. Belabed, M. Bouet, S. Secci, Virtual network functions placement and routing optimization, in: Proceedings of IEEE CloudNet, 2015, pp. 171–177.

[16] L. Nonde, T. El-Gorashi, J. Elmirghani, Energy efficient virtual network embedding for cloud networks, IEEE J. Lightwave Technol. 33 (9) (2015) 1828–1849.

[17] J. Whiteaker, F. Schneider, R. Teixeira, Explaining packet delays under virtualization, ACM SIGCOMM Comput. Commun. Rev. 41 (1) (2011) 38–44.

[18] G. Wang, T. Ng, The impact of virtualization on network performance of Amazon EC2 data center, in: Proceedings of IEEE INFOCOM, 2010, pp. 1–9.

[19] S. Farokhi, E.B. Lakew, C. Klein, I. Brandic, E. Elmroth, Coordinating CPU and memory elasticity controllers to meet service response time constraints, in: Proceedings of IEEE ICCAC2015, 2015, pp. 69–80.

[20] Y. Baram, Orthogonal Patterns in Binary Neural Networks, NASA Technical Memorandum No. 100060, 1988.

[21] F. Travostino, P. Daspit, L. Gommans, C. Jog, C.D. Laat, J. Mambretti, I. Monga, B.V. Oudenaarde, S. Raghunath, P.Y. Wang, Seamless live migration of virtual machines over the MAN/WAN, Future Gener. Comput. Syst. 22 (8) (2006) 901–907.

[22] M. Yu, Y. Yi, J. Rexford, M. Chiang, Rethinking virtual network embedding: substrate support for path splitting and migration, ACM SIGCOMM Comput. Commun. Rev. 38 (2) (2008) 17–29.

Koki Inoue received his M.E. in information science and technology in 2016 from Osaka University, where he is currently a postgraduate studying for a Ph.D.