Extendable NFV-Integrated Control Method Using Reinforcement Learning

Akito Suzuki*, Masahiro Kobayashi*, Yousuke Takahashi†, Shigeaki Harada*,
Keisuke Ishibashi* and Ryoichi Kawahara*
* NTT Network Technology Laboratories, NTT Corporation, 3-9-11 Midori-Cho Musashino-Shi, Tokyo, 180-8585, Japan
Email: {suzuki.akito, kobayashi.masahiro, harada.shigeaki, ishibashi.keisuke, kawahara.ryoichi}@lab.ntt.co.jp
† Technology Development, NTT Communications Corporation, 3-4-1 Shibaura, Minato-ku, Tokyo 108-8118, Japan
Email: yousuke.takahashi@ntt.com

***Abstract*—Network functions virtualization (NFV) enables telecommunications service providers to provide various network services by flexibly combining multiple virtual network functions (VNFs). To provide such services with carrier-grade quality, an NFV controller must optimally allocate such VNFs into physical networks and servers, taking into account combination(s) of objective functions and constraints for each metric defined for each VNF type. The NFV controller should also be extendable, i.e., new metrics should be able to be added. One approach for NFV control to optimize allocations is to construct an algorithm that simultaneously solves the combined optimization problem. However, this algorithm is not extendable because the problem formulation needs to be rebuilt every time, e.g., a new metric is added. Another approach involves using an extendable network-control architecture that coordinates multiple control algorithms specified for individual metrics. However, to the best of our knowledge, no method has been developed to optimize allocations through this kind of coordination. In this paper, we propose an extendable NFV-integrated control method by coordinating multiple control algorithms. We also propose an efficient coordination algorithm based on reinforcement learning. Finally, we evaluate the effectiveness of the proposed method through simulations.**

I. Introduction

Network functions virtualization (NFV) [2], [3] enables telecommunications service providers (TSPs) to provide various network services by flexibly combining multiple virtual network functions (VNFs). These network services in NFV can be provided by combining multiple VNFs (e.g., virtual machines (VMs), intrusion detection systems (IDSs)), each of which is specified for each network service type, between end-to-end hosts such as a content server and a user terminal. To provide such services, a TSP must optimally embed virtual networks consisting of VNFs and routes between the VNFs into underlying physical servers and networks. It has been reported that the inefficient management of network policies (e.g., VNF placements) accounts for 78% of data-center downtime [4], [5]; therefore, optimal resource embedding is inevitable to provide carrier-grade network services. To avoid this inefficiency, the NFV controller must determine the optimal allocations of the VNFs and optimal routes between the VNFs in the network by solving an optimization problem, which is formulated by objective function(s) (e.g., minimizing link-utilization, server-load), and constraint(s) (e.g., upper limit link-bandwidth, server-CPU). However, though significant amount of research have been conducted [6], there is no formulation and solution of a unified allocation problem taking into account all conditions consisting of combination(s) of objective functions and constraints.

This paper is motivated to address the challenge with an NFV-integrated control method, i.e., how to construct such a unified optimal allocation problem and solve it. This problem becomes more difficult due to the increase in the number of control metrics and diversification of objective functions, where the control metric is defined by parameter(s) to characterize the state of a controlled network, e.g., VNF placements and routes between the VNFs. In addition, the NFV-integrated control method also should be extendable, i.e., able to handle new metrics being added or constraints of metrics being changed. This requirement is crucial because a unified optimization problem needs to be solved quickly and easily even when a new network service starts or a new constraint needs to be taken into account.

A simple way to achieve the extendibility is to solve independently pre-specified optimization problems for each control metric. However, if we independently solve each optimization problem taking into account only the problem's constraints, we may not satisfy all the constraints. Hereafter, we call such a situation a "control conflict." (See Section II in detail.) To avoid such control conflict, the NFV-integrated control method must calculate the optimal allocation that satisfies all conditions determined by combination(s) of control metrics.

Previous studies on NFV-integrated control methods can be categorized into two approaches. One is the "*combined*" approach [5], [7]–[9], which builds a specified algorithm that simultaneously solves the combined optimization problem. However, it is not extendable because we need to reconstruct the problem formula every time the combination of control metrics change. The other approach is the "*coordinated*" approach [3], [10], [11], which involves using an extendable control architecture that coordinates multiple control algorithms pre-specified for individual metrics. However, to the best of our knowledge, no method has been developed to

An earlier version of this paper was presented with no review process in [1], which is a domestic conference in Japan.

optimize allocations through this type of coordination.

In this paper, we propose an "*extendable*" NFV-integrated control method based on the *coordinated* control architecture, which consists of multiple pre-specified control algorithms and a single coordination algorithm between the control algorithms. Our key idea for extendibility is "modularization" that divides a whole system into standardized functional elements and reduces the interdependence among the elements, which is a well known technique when designing/managing huge complex systems. We first prepare and solve each control algorithm for each metric then interactively improve the results using our coordination algorithm. We also propose an efficient coordination algorithm on the basis of reinforcement learning (RL) [12]. The learning makes it possible to learn the strategy how to find better allocations efficiently from past exploration steps. Our method requires more iterations than the *combined* approach, but it achieves extendibility through the *coordinated* control architecture.

This paper is structured as follows. Section II describes our motivation in detail. Section III describes our *extendable* NFV-integrated control method. An efficient algorithm for our method using RL is proposed in Section IV. Section V evaluates the performance and Section VI concludes the paper.

II. Challenges and Motivation

We consider the use case in which we provide secure-cloud-computing service consisting of routes between VMs via an IDS. In this case, the control metrics are routes, VM and IDS placements, and each control algorithm is pre-formulated (detailed formulation is given in Section V-B).

Independently solving each optimization problem leads to the following control conflicts. *(1) Conflict between constraints - Capacity overload:* Since each control algorithm takes into account only its constraints, there is a possibility that all constraints are not satisfied at the same time. For example, when each problem for each metric is solved at the same time, VMs and IDSs will be allocated on the same server, resulting in server overload. *(2) Conflict between objective functions - Oscillatory solution:* If each algorithm with a different objective function is conducted independently of each other, the network may become unstable. For example, if the IDS-allocation algorithm to balance server loads and the VM-allocation one to minimize electric power consumption (i.e., the number of powered-on servers) are used independently (e.g., the latter is done after the former, repeatedly), most assignment results are repeatedly changed.

The above conflicts can be avoided by sequentially solving each optimization problem for residual resources of each allocated result. However, there is no guarantee that the obtained results after conducting all the algorithms become optimal. This is because, since the result of the previous algorithm is fixed, a sub-optimal solution may be inevitable. For example, when the results of the physical distance between allocated VMs are long, an inefficient routing is inevitable.

This paper is motivated to avoid such conflicts and inefficiency described above. In addition, we address the challenge

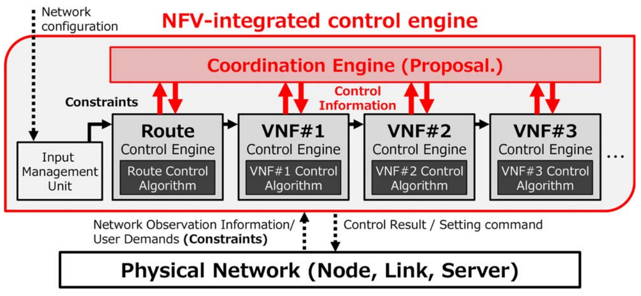

Fig. 1. Overview of our proposed NFV-integrated control method

of extendibility for changing control metrics that is considered essential in the NFV-integrated control. Our goal is to construct an *extendable* NFV-integrated method, i.e., changing and adding the NFV control metrics without changing each control-algorithm formulation. Though this paper cannot solve all the challenges of extendibility and cannot cover all use cases, to the best of our knowledge, it is the first paper to tackle this problem, and we expect that more complicated use cases can be solved by enhancing the proposed method.

III. NFV-integrated Control Method

A. Overview of proposed method

We developed an extendable NFV-integrated control method by coordinating multiple control algorithms. Our method executes hierarchical control consisting of multiple "control engines" and a single "coordination engine" (Fig. 1). A control engine has an algorithm to calculate a "solution" for each control metric and calculate the "evaluation" of the solution quantitatively. The evaluation of a solution is defined as the objective-function value if the constraints of a control algorithm are satisfied; otherwise, it returns a negative value. Using this negative value, we can determine whether the constraints are satisfied.

The coordination engine explores a solution by changing a part of the solution to improve the "comprehensive evaluation value (CEV)", which is defined as a unique value determined by all evaluations of solutions calculated by the control engines, e.g., the weighted average of each evaluation of solution. The weight of each evaluation is determined from the importance of each objective function.

B. Procedure of proposed method

We describe the procedure of the proposed method to improve the CEV. First, each control engine calculates initial solutions, then our coordination engine recursively explores the solutions to improve the CEV.

The coordination engine first changes a part of a solution on the basis of the "current" CEV, then the changed solution is sent to each control engine. Next, each control engine calculates the evaluation based on the changed solution. The coordination engine calculates a "next" CEV on the basis of the evaluation then returns to the beginning of the procedure. When the exploration is terminated, we regard the highest CEV solution among the past iterations as the final solution.

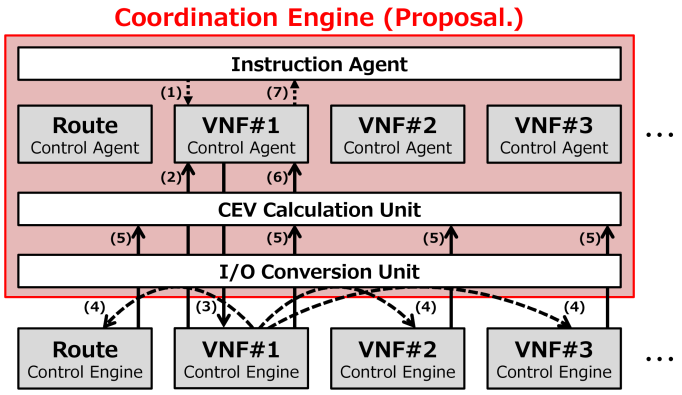

Fig. 2. Overview of coordination engine based on reinforcement learning. An example when instruction agent selects VNF#1 control agent.

TABLE I
SYMBOL DESCRIPTIONS FOR COORDINATION ALGORITHM

Symbol	**Definition**
ia	Instruction agent
$\boldsymbol{G} := \{g\}$	Control agent set
$\boldsymbol{E} := \{e\}$	Control engine set
$t \in T$	Exploration step T: Total exploration steps
T^g	Total exploration steps of control agent ($g \in \boldsymbol{G}$)
s_t^{agent}	State of each agent at step t (agent $\in \text{ia} \cup \boldsymbol{G}$)
a_t^{agent}	Action of each agent at step t (agent $\in \text{ia} \cup \boldsymbol{G}$)
r_t^{agent}	Reward of each agent at step t (agent $\in \text{ia} \cup \boldsymbol{G}$)
$Q(s_t, a_t)$	Policy value for state s_t and action a_t
α, γ	Hyper-parameter (Default: $\alpha = 0.2$ and $\gamma = 0.9$)

IV. Proposed coordination algorithm

We developed an efficient coordination algorithm by using RL to find better solutions with fewer explorations. We give an overview of RL and the proposed coordination algorithm.

A. *Overview of coordination algorithm*

Reinforcement learning solves a decision problem of what action an "agent" should take by observing the current state within a certain "environment." An agent receives a reward from the environment depending on the selected action then learns a strategy of how to maximize the received reward through a series of selected actions. In an NFV-integrated control environment, an agent's strategy indicates an efficient solution exploration to improve the CEV, and the agent observes the current solution and CEV. We base our proposed algorithm on RL because general-purpose learning is possible by just defining states, actions, and rewards and applying them to general control engines without a specific algorithm.

Specifically, we use hierarchical multi-agent RL [13], consisting of a single "instruction agent" and multiple "control agents" (Fig. 2). The instruction agent learns a selection of the control agent, and the control agent learns an efficient solution exploration for the corresponding control engine. The purpose of hierarchical multi-agent RL is to achieve the extendibility of our method by preparing a specific control agent for each control engine. Since all agents' learning algorithms are common, the learning algorithm is not affected by any change of any control algorithm.

We introduce the input/output (I/O)-conversion unit, which converts the I/O format of each control engine. Because the changed solution of a control engine may affect evaluation values of other control engines, we need to share the changed solution through converting I/O format. For example, the VNF placements affect the routes between the VNFs, therefore this unit needs to convert the VNF placements (i.e., output of a VNF control engine) into traffic demands between servers on which the VNFs are allocated (i.e., input of a route control engine). We assume that the cost of implementing the I/O conversion unit is lower than the cost of rebuilding each algorithm formulation.

We describe the procedure of our coordination algorithm. As shown in Fig. 2, the instruction agent first selects a control agent (1), then the selected control agent starts exploring solutions and learning a strategy of exploration (2)–(6). In the exploration step, the selected control agent observes the current solution (2) and sends the changed solution (3). After sharing the changed solution through the I/O-conversion unit (4), each control engine calculates the evaluation. Next, the CEV is calculated from all evaluations (5), then the control agent receives the CEV as a reward (6). Then, the instruction agent learns the strategy of selecting a control agent on the basis of the maximum CEV in the exploration (7) and selects next agent on the basis of the strategy. After repeating the procedure, the best solution is output as the final solution.

B. *Formulation of coordination algorithm*

We describe the formulation of our coordination algorithm. Table I summarizes the definitions of the variables of our coordination algorithm. For each agent leaning algorithm, we use Q-learning [12], which learns the relationship of a state, action, and reward to maximize the policy value. Policy value $Q(s_t, a_t)$ is defined as the expectation of the sum of rewards obtained in the future when action a_t is selected in state s_t.

1) Instruction-agent algorithm: The instruction agent learns how to select control agents. State is defined as the selected control agent, action as the selection of the next control agent, and reward as the maximum CEV obtained during this control-agent selection.

The instruction-agent algorithm is shown in Algorithm 1. Lines 1–2 show the initialization of state s_0^{ia} and $Q\left(s^{\text{ia}}, a^{\text{ia}}\right)$. The term "$\epsilon$ greedy" in line 4 means the action selected on the basis of the strategy that a random action is selected with probability ϵ; otherwise, an action a_{t+1}^{ia} that maximizes Q is selected (i.e. $\arg\max_{a'} Q\left(s_t^{\text{ia}}, a'\right)$) with probability $1-\epsilon$. The term "action" in line 5 shows the action of the instruction agent, which is the switch to the new control agent s_{t+1}^{ia}. The term "agent learning (s_{t+1}^{ia})" in line 6 means control-agent learning (Algorithm 2). Lines 7–8 show instruction-agent learning, which means that $Q\left(s^{\text{ia}}, a^{\text{ia}}\right)$ is updated from the relationship of state s^{ia}, action a^{ia}, and reward r^{ia}. At line 9, t^g means the number of iterations in Algorithm 2.

Algorithm 1 Instruction-agent Learning

```
1: initialize: $t \leftarrow 0,\ Q\left(s^{\text{ia}}, a^{\text{ia}}\right) \leftarrow 0$
2: initialize: $s_0^{\text{ia}} \leftarrow$ random choice from $\boldsymbol{G}$
3: while $t < T$ do
4:     $a_t^{\text{ia}} \leftarrow \epsilon$ greedy $\left(s_t^{\text{ia}}\right)$
5:     $s_{t+1}^{\text{ia}} \leftarrow$ action $\left(a_t^{\text{ia}}\right)$
6:     $r_{t+1}^{\text{ia}}, t^g \leftarrow$ agent learning $\left(s_{t+1}^{\text{ia}}\right)$
7:     $E_{td} \leftarrow r_{t+1}^{\text{ia}} + \gamma \max_{a'} Q\left(s_{t+1}^{\text{ia}}, a'\right) - Q\left(s_t^{\text{ia}}, a_t^{\text{ia}}\right)$
8:     $Q\left(s_t^{\text{ia}}, a_t^{\text{ia}}\right) \leftarrow Q\left(s_t^{\text{ia}}, a_t^{\text{ia}}\right) + \alpha E_{td}$
9:     $t \leftarrow t + t^g$
10: end while
```

Algorithm 2 Control-agent Learning

```
1: initialize: $Q\left(s^g, a^g\right) \leftarrow 0$
2: initialize: $s_0^g \leftarrow \boldsymbol{A}^e$
3: for $t = 0$ to $T^g - 1$ do
4:     $a_t^g \leftarrow \epsilon$ greedy $\left(s_t^g\right)$
5:     $s_{t+1}^g \leftarrow$ action $\left(a_t^g\right)$
6:     $\boldsymbol{T}^{\text{server}} \leftarrow$ I/O conversion
7:     $U_{\max}^{\text{server}} \leftarrow$ evaluation by VNF control engines
8:     $U_{\max}^{\text{link}} \leftarrow$ evaluation by Route control engine
9:     $r_{t+1}^g \leftarrow$ CEV calculation $\left(U_{\max}^{\text{server}}, U_{\max}^{\text{link}}\right)$
10:    $E_{td} \leftarrow r_{t+1}^g + \gamma \max_{a'} Q\left(s_{t+1}^g, a'\right) - Q\left(s_t^g, a_t^g\right)$
11:    $Q\left(s_t^g, a_t^g\right) \leftarrow Q\left(s_t^g, a_t^g\right) + \alpha E_{td}$
12:    $t^g \leftarrow t$
13:    if $s_{t+1}^g$ is $end\ state$ then
14:        return $\max_{t \in \{0,1,\cdots,t^g\}} \left\{r_t^g\right\}, t^g + 1$
15:    end if
16: end for
17: return $\max_{t \in \{0,1,\cdots,T^g-1\}} \left\{r_t^g\right\}, T^g$
```

2) Control-agent algorithm: The control agent learns how to efficiently change the solution of the control engine. State is defined as the solution of the control engine, action as the changing of the control solution of each control engine, and reward as the CEV (examples are given in Section V-B).

The control-agent algorithm when the control agent g is selected by the instruction agent is shown in Algorithm 2, which is an example of using the route and VNF control engines. Lines 1–2 show the initialization, and $\boldsymbol{A}^e$ means the initial solution corresponding to control engine e. The term "action" in line 5 means the changing part of the solution of selected control engine e. Then it outputs the result to the selected control engine e. In line 6, the result is shared among other control engines through I/O-conversion unit. Lines 7–8 show the calculation of evaluations of all control engines based on the changed solution. The term "CEV calculation" in line 9 means the calculation the CEV as the reward of control agent r_{t+1}^g. The term "end state" in line 13 means the termination condition of control-agent learning. That is, after reaching the solution that does not satisfy at least one constraint, the control agent stops the exploration. In lines 14 and 17, the control agent returns the maximum CEV during exploration as a reward for the instruction agent.

V. Evaluation

We evaluated the effectiveness of the proposed algorithm through simulations in terms of solution-exploration speed, scalability, and computation time. We implemented our co-ordination algorithm and physical network simulator using Python from scratch, and each pre-specified control algorithm using GNU Linear Programming Kit (GLPK) [14] to calculate initial solutions.

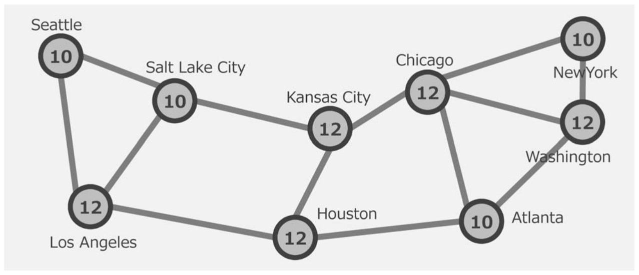

Fig. 3. Internet2 topology. Value of each node shows server capacity when N_{vm} is 20 and average server utilization is 50%. When a VM is allocated to a server, the consumed server capacity is same as the VM size.

TABLE II
Symbol descriptions for control algorithms

Symbols	Definitions
$G(N, L)$	Network graph$N$: node set, L: link set
$n \in N$	Node
link $(i, j) \in L$	Link from node i to node j
c_{ij}	Link capacity of link (i, j)
$\boldsymbol{T}^{\text{vm}} := \left\{t_{ij}^{\text{vm}}\right\}$	Traffic demands from VM i to VM j
$\boldsymbol{T}^{\text{server}} := \left\{t_{pq}\right\}$	Traffic demands from node p to node q
x_{ij}^{pq}	Proportion of passed t_{pq} on link (i, j)
$U_{\max}^{\text{link}}$	Maximum link utilization
$N_{\text{server}}, N_{\text{vm}}, N_{\text{ids}}$	Number of servers, VMs, and IDSs
$v_i \in V$	i th VM (V: VM set
$i_j \in I$	j th IDS (I: IDS set)
$s_k \in S$	k th Server (S: Server set
c_i^{vm}, c_j^{ids}	i th VM size, j th IDS size
R_k	k th server capacity
$\boldsymbol{A}^{\text{vm}} := \left\{a_{ik}^{\text{vm}}\right\}$	VM allocation (i th VMk th server
$\boldsymbol{A}^{\text{ids}} := \left\{a_{jk}^{\text{ids}}\right\}$	IDS allocation (j th IDSk th server
$U_{\max}^{\text{server}}$	Maximum server utilization

A. Evaluation conditions

We used the topology of Internet2 [15], which consists of 9 nodes (Fig. 3). We assumed that a physical server is connected to each node and the capacity of each link is 1.0 Gbps. When each virtual network request is accepted, the amounts of server and link resources consumed depend on the request size. The symbols used in the formulation are defined in Table II.

A virtual network request consists of one origin VM and one destination VM, each of which is allocated to a physical server. Each VM size is given randomly among 1, 2, or 3, which indicates the processing capacity of the VM request, such as the requested number of CPU cores. We assume that there are a certain number of VMs N_{vm}. We generate an origin–destination (OD) pair between two VMs chosen among N_{vm} randomly. The OD traffic demand between the

VMs is set to 100 Mbps. We repeat the above procedure until 50 OD pairs are generated. We assume that each OD traffic demand is routed through an IDS, which is also allocated to a physical server. Note that, if the VMs and IDS for an OD pair are allocated in the same server, the OD traffic demand on the network is regarded as 0. The IDS size indicates the processing capacity, which is fixed at 3. The number of IDSs N_{ids} is set to 5 when N_{vm} is 20. As N_{vm} increases, N_{ids} and server capacity increase proportionally. (See Fig. 3.)

B. Control algorithms

We introduce Route, VM, and IDS control algorithms ($\boldsymbol{E} = \{Route, VM, IDS\}$). The calculation procedure of the initial solution is as follows. After the VM and IDS control algorithms calculate the optimal allocations without taking into account the constraints of other control algorithms, the route control algorithm calculates the end-to-end route between VMs via an IDS.

Minimization of maximum link utilization for route control and minimization of maximum server utilization for VM and IDS controls are introduced as objective functions. Link capacity for route control and server capacity for VM and IDS controls are imposed as constraints. Here, VM allocation a_{ik}^{vm} indicates the VM solution in which a_{ik}^{vm} is 1 if v_i is assigned to s_k; otherwise, 0. Similarly, IDS allocation a_{jk}^{ids} indicates the IDS solution in which a_{jk}^{ids} is 1 if i_j is assigned to s_k; otherwise, 0.

The formulation of the route control algorithm is as follows:

$$
\begin{aligned}
\min \ &: \ U_{\max}^{\text{link}} && (1)\\
\text{s.t.} \ &: \ \sum_{j:(i,j)\in L} x_{ij}^{pq} - \sum_{j:(i,j)\in L} x_{ji}^{pq} = 0 && (2)\\
&\qquad (\forall p, q \in N, i \neq p, i \neq q) &&\\
&\ \sum_{j:(i,j)\in L} x_{ij}^{pq} - \sum_{j:(i,j)\in L} x_{ji}^{pq} = 1 && (3)\\
&\qquad (\forall p, q \in N, i = p) &&\\
&\ \sum_{p,q\in N} t_{pq} x_{ij}^{pq} \leq c_{ij} U_{\max}^{\text{link}} &&\\
&\qquad (\forall (i,j) \in L, \forall p, q \in N) && (4)\\
&\ 0 \leq x_{ij}^{pq} \leq 1 \qquad (\forall (i,j) \in L, \forall p, q \in N) && (5)\\
&\ 0 \leq U_{\max}^{\text{link}} \leq 1 && (6)
\end{aligned}
$$

The formulation of the VM control algorithm is as follows:

$$
\begin{aligned}
\min \ &: \ U_{\max}^{\text{server}} && (7)\\
\text{s.t.} \ &: \ \sum_{s_k\in S} a_{ik}^{\text{vm}} = 1 \qquad (\forall v_i \in V) && (8)\\
&\ \sum_{v_i\in V} c_i^{\text{vm}} a_{ik}^{\text{vm}} \leq R_k U_{\max}^{\text{server}} \qquad (\forall s_k \in S) && (9)\\
&\ a_{ik}^{\text{vm}} \in [0,1] && (10)\\
&\ 0 \leq U_{\max}^{\text{server}} \leq 1 && (11)
\end{aligned}
$$

The formulation of the IDS control algorithm replaces c_i^{vm} and a_{ik}^{vm} with c_j^{ids} and a_{jk}^{ids} for that of the VM control algorithm. Note that, we selected these control algorithms as an example, which is commonly used in previous studies. In our proposed method, each control algorithm is saved as a model file and it can change by only changing the model file.

C. Coordination algorithm

We introduce VM and IDS control agents as the control agents ($\boldsymbol{G} = \{VM, IDS\}$). The route control agent is not introduced here, because we do not change any routes between the VNFs unless the VNF placements change in this use case. The route control engine is used only to calculate the part of the CEV by solving the route control algorithm in each step.

The state of a control agent defines the VM or IDS allocation. The action of the control agent defines one VM or IDS migration randomly selected from the most used server on the basis of its agent strategy. Note that, in this action, only one migration is executed, and the VNF control algorithm described using (7)–(11) is not solved. The reward of the control agent defines the CEV if all constraints are satisfied; otherwise, the penalty is -100. The CEV is defined as follows:

$$r_t^g = \frac{\theta_1}{U_{\max}^{\text{link}}} + \frac{\theta_2}{\tilde{U}_{\max}^{\text{server}}}, \qquad (12)$$

where θ_1 and θ_2 are weighting parameters indicating the importance of each control-objective function. The term $\tilde{U}_{\max}^{\text{server}}$ is the maximum server utilization after aggregating VM and IDS allocations. We set the total exploration steps of instruction and control agents to $T = 10\ 000$, and $T^g = 100$ in all evaluations. In the I/O-conversion unit, $\boldsymbol{T}^{\text{server}}$ is calculated on the basis of $\boldsymbol{A}^{\text{vm}} \boldsymbol{A}^{\text{ids}}$, and $\boldsymbol{T}^{\text{vm}}$.

D. Evaluation results

1) Solution-exploration speed: Figure 4 shows the average transition of the best solutions found until the current exploring step, at which we carried out 20 calculations to randomly change the OD of VMs and show the average of the 20 results. The width of each line indicates the standard deviation ($\pm\sigma$). We compared the solution-exploration speeds of the proposed algorithm based on RL (w/ RL) and an algorithm based on changing solutions randomly (w/o RL). Note that, in the case of w/o RL, we set $\epsilon = 1$ and also skipped both agent learning steps, i.e. lines 7–8 in Algorithm 1 and lines 10–11 in Algorithm 2. We excluded the case in which an initial solution does not satisfy the constraints on route calculation to evaluate the exploration speed of both algorithms (w/ RL and w/o RL) fairly. This is because starting from an unsatisfied initial solution would drastically increase the speed for both algorithms. In this evaluation, we set $\theta_1 = 1$ and $\theta_2 = 0$ to compare the exploration speed quantitatively.

Figure 4 shows the solution-exploration-speeds of 50% and 90% average server utilization, which is the ratio of total VM demands to total server capacity. We set the average server utilization to 50% and 90%, by changing each server capacity proportionally and making minor adjustments of VM size. The initial solution of link-utilization was high in all cases due to the control conflict mentioned in Section II. By repeating the exploration, the link-utilization was improved in all cases.

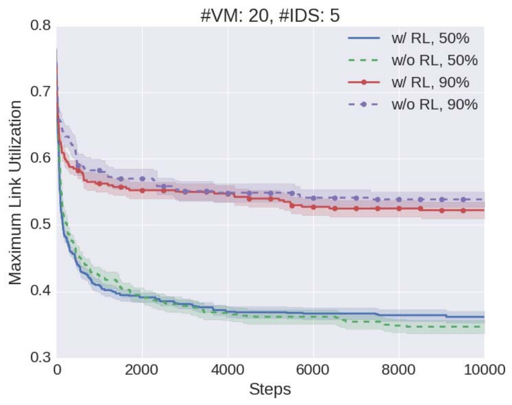

Fig. 4. Comparison of exploration speeds of w/ and w/o RL depending on average server utilization

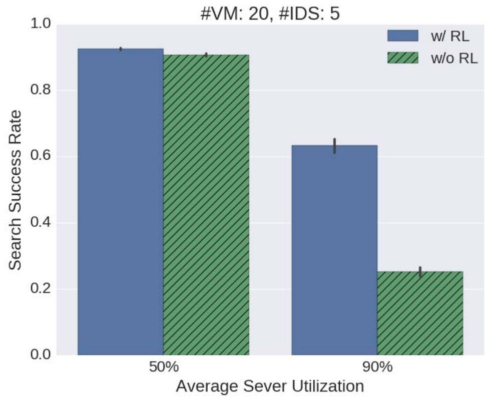

Fig. 5. Comparison of exploration success rate with standard deviation of 20 computations

Although w/ RL was effective in the 90% case, it was less effective than w/o RL in the 50% case. It indicated that RL was effective when the average server utilization was high, i.e., when finding better solutions by random explorations was difficult. We consider the reason from the following analysis.

Figure 5 compares the exploration success rates, which mean the ratios of steps in which all constraints are satisfied to all exploring steps. Although the success rates of both algorithms (w/ RL and w/o RL) were high in the 50% case, i.e., when unused server capacity was high, w/ RL had a higher success rate than w/o RL in the 90% case. Therefore, w/ RL had a higher exploration speed in the 90% case, as shown in Fig. 4. Note that a higher success rate is not always better because there is a possibility of repeating the same action or remaining in the same state many times. Despite w/ RL having a slightly higher success rate than w/o RL in the 50% case, as shown in Fig. 5, w/o RL had a higher exploration speed. The reason for this is that RL tends to select the same action in the same state.

2) Scalability: Figure 6 shows the results of our method when we varied N_{vm} from 20 to 2000. As shown in Fig. 6, resource utilization could be halved after about 10 000 exploring steps for networks with up to about 500 VMs. The best solution for 200 VMs was better than that of 20. Since the number of OD pairs and VM size ranges were fixed, the probability of accommodating two VMs and one IDS for an OD pair that generate traffic in a single server increased, i.e., most OD traffic demand was regarded as 0, as N_{vm} increased.

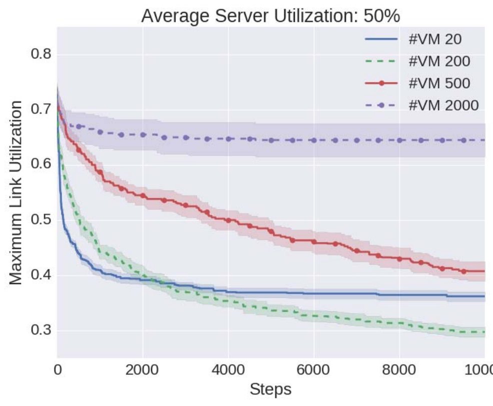

Fig. 6. Results for proposed algorithm, depending on N_{vm} and N_{ids}. As N_{vm} increases, N_{ids} increases proportionally.

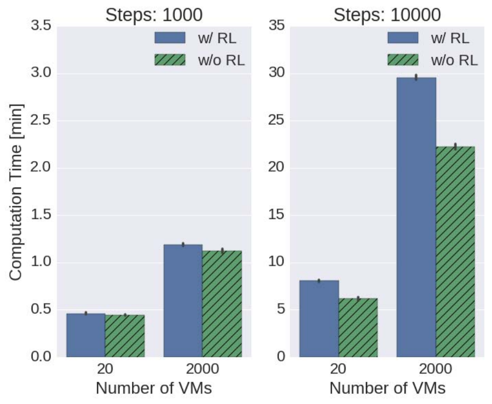

Fig. 7. Computation times with standard deviation of 10 computations, calculated using Intel core i7 4790k CPU of single core

3) Computation time: Figure 7 compares results of computation times for N_{vm} 20 and 2000. Although N_{vm} increased 100 times, the computation time increased only several times. This means that, from the viewpoint of computation time, the proposed method is scalable with respect to N_{vm}, with up to 2000 VMs. The difference between w/ RL and w/o RL is the overhead time of RL. Where the total exploration steps T is 1000, there is only a slight overhead. When T increases to 10 000, the overhead expands by about 1.3 times. The increase ratios are 1.31 and 1.33 for N_{vm} 20 and 2000, respectively, and do not depend on the N_{vm}. Considering that the exploration success rate doubles (Fig. 5), though overhead time increases by about 1.3 times, w/ RL can more efficiently explore better solutions than w/o RL.

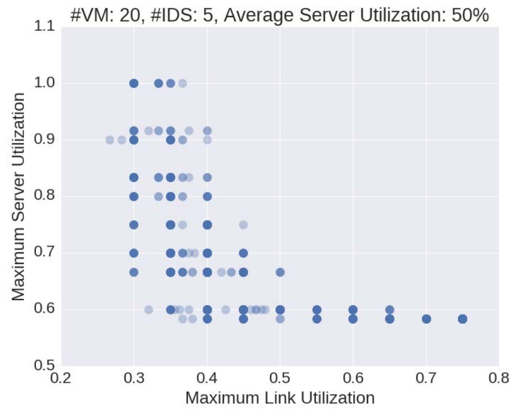

Fig. 8. Pareto curve obtained from proposed algorithm. Gradation of each point shows distribution of 1000 calculations.

4) Pareto curve: Figure 8 shows a Pareto chart for N_{vm} 20, N_{ids} 5 and the average server utilization of 50%. After all initial conditions were fixed, calculations with randomly given θ_1 and θ_2 were carried out 1000 times. Gradation of each point shows the distribution of solutions (deeper colors mean denser distributions). As shown in Fig. 8, many solutions are clearly distributed around the Pareto curve in the calculation of 10 000 steps. It means that the proposed method can suggest a wide variety of options of solutions by adjusting the weighting parameters θ_1, θ_2.

VI. Conclusion

We presented an extendable network functions virtualization (NFV)-integrated control method by coordinating multiple control algorithms. We also developed an efficient coordination algorithm on the basis of reinforcement learning, which makes it possible to find better solutions with fewer explorations by learning a strategy to improve resource-utilization efficiency from past exploration steps. The simulation evaluation revealed that the proposed algorithm can improve solution exploration when the average server utilization is high. We also found that it can reduce link-utilization by almost half after less than 10 000 steps in the case of several hundred virtual machines and a hundred intrusion detection systems.

For future work, we plan to quantitatively evaluate the extendibility of the proposed method and its applicability against more complicated use cases with realistic traffic patterns and virtual network functions (VNFs) demands. We also plan to enhance the solution-exploration-speed and scalability of our coordination algorithm by using deep reinforcement learning [16] and parallelization of agent learning [17].

References

[1] A. Suzuki *et al.*, "A method of coordinating multiple control algorithms for NFV," in *IEICE Tech. Rep.*, ser. IN2016-103, vol. 116, no. 485, 2017, pp. 37–42, (in Japanese).
[2] B. Han *et al.*, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Commun. Mag.*, vol. 53, no. 2, pp. 90–97, 2015.
[3] R. Mijumbi *et al.*, "Management and orchestration challenges in network functions virtualization," *IEEE Commun. Mag.*, vol. 54, no. 1, pp. 98–105, 2016.
[4] L. E. Li *et al.*, "Pace: Policy-aware application cloud embedding," in *Proc. IEEE INFOCOM*, 2013, pp. 638–646.
[5] L. Cui *et al.*, "Synergistic policy and virtual machine consolidation in cloud data centers," in *Proc. IEEE INFOCOM*, 2016, pp. 1–9.
[6] J. G. Herrera and J. F. Botero, "Resource allocation in NFV: A comprehensive survey," *IEEE Trans. Netw. Service Manag.*, vol. 13, no. 3, pp. 518–532, 2016.
[7] J. W. Jiang *et al.*, "Joint VM placement and routing for data center traffic engineering," in *Proc. IEEE INFOCOM*, 2012, pp. 2876–2880.
[8] M. Yoshida *et al.*, "MORSA: A multi-objective resource scheduling algorithm for NFV infrastructure," in *Proc. APNOMS*, 2014, pp. 1–6.
[9] Y. Jin *et al.*, "Towards joint resource allocation and routing to optimize video distribution over future Internet," in *Proc. IFIP Netw. Conf.*, 2015, pp. 1–9.
[10] K. Tsagkaris *et al.*, "A survey of autonomic networking architectures: towards a unified management framework," *Int. J. Netw. Manage.*, vol. 23, no. 6, pp. 402–423, 2013.
[11] K. Tsagkaris *et al.*, "Customizable autonomic network management: integrating autonomic network management and software-defined networking," *IEEE Veh. Technol. Mag.*, vol. 10, no. 1, pp. 61–68, 2015.
[12] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.
[13] R. Sun and C. Sessions, "Self-segmentation of sequences: automatic formation of hierarchies of sequential behaviors," *IEEE Trans. Syst. Sci. Cybern.*, vol. 30, no. 3, pp. 403–418, 2000.
[14] "GLPK," https://www.gnu.org/software/glpk/.
[15] "Internet2," http://internet2.edu/.
[16] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
[17] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *Proc. ICML*, 2016, pp. 1928–1937.