Master's Thesis

Title

Implementation and Experimental Evaluation of Dynamic and Adaptive NFV Management System Based on Biochemical Reactions

Supervisor

Professor Morito Matsuoka

Author

Shuto Sugita

February 5th, 2020

Graduate School of Information Science and Technology

Osaka University

Master's Thesis

Implementation and Experimental Evaluation of Dynamic and Adaptive NFV Management System Based on Biochemical Reactions

Shuto Sugita

Abstract

Network Functions Virtualization (NFV) is now paid much attention to for its cost-effectiveness and flexibility to realize various types of network services. In order to operate the NFV system efficiently, NFV management should be adaptive and dynamic. The NFV management includes decision of placement of Virtualized Network Functions (VNFs) on servers, resource allocation to VNFs, and flow routing in accordance with service function chaining requests. Furthermore, in order to quickly respond to environmental fluctuations and to maintain the scalability of NFV services, a distributed control is more suitable than a centralized one. One way to achieve such behaviors is to exploit biochemical mechanisms with autonomous dispersibility and self-organizing properties.

Our research group has proposed a construction method of service space in virtualized network system based on biochemically-inspired tuple space model. In this method, the behaviors in the virtualized network system are described by biochemical reactions in tuple spaces. This makes it possible to determine a VNF to be provided in each distributed server, share server resources by a plurality of types of VNFs, and execute VNFs in a distributed manner. Since biochemical reaction equations are defined and executed independently in each tuple space, this method can realize the autonomous and decentralized behaviors in the NFV system. In the previous studies, the behaviors of the proposed method have been confirmed with computer simulations. Also, the proposed method has been implemented and evaluated in a simple experimental environment. However, the applicability to the actual NFV system has not been clearly shown.

In this thesis, we implement the proposed method as components on the NFV framework in cooperation with other components. In detail, we exploit an NFV environment using Open Platform for NFV (OPNFV), which is an open source project aiming to build a software environment

to implement the NFV system, and implement the proposed NFV management system on the framework. By extensive experimental evaluations, we show that the proposed NFV management system works properly for realizing video streaming services in terms of adaptively allocating server resources to VNFs in accordance with the amount of traffic. It is also shown that the proposed system can dynamically distribute incoming flows to multiple VNFs for load balancing.

Keywords

Network Functions Virtualization (NFV)
Software Defined Networking (SDN)
Service Function Chaining (SFC)
Open Source Software
Biochemical Mechanism

Contents

List of Figures

List of Tables

1 Introduction

Recently, the amount and types of devices connected to the information network are rapidly increasing because of the spread of smartphones and tablet-type devices and the development of Internet of Things (IoT) [1] technologies. As a result, the network traffic has increased rapidly, and network services have also become more diverse. Conventionally, dedicated hardware middleboxes are introduced to accommodate a large amount of traffic that cannot be handled by existing equipment and to introduce new network services. However, this method requires high cost in terms of capital expenditure (CAPEX), such as initial installation cost for preparing the equipment itself and the physical space for installing the equipment. It also increases operating expense (OPEX), such as power costs for equipment operation. Besides, since the maintenance and operation of the equipment and the response to environmental changes such as a system failure and a sudden increase in traffic demand take time, the flexibility is low.

Network Functions Virtualization (NFV) [2] is a technology that can address such problems. In NFV, a network function, which has been implemented by a dedicated hardware middle-box, is implemented as a software running on a general-purpose server. The network function implemented as a software is called Virtualized Network Function (VNF). As a VNF, a wide variety of network functions such as a firewall [3], a Deep Packet Inspection (DPI) [4], a Network Address Translation (NAT) [5], and an Evolved Packet Core (EPC) [6] have been implemented [7]. By implementing a network function as a VNF, the network function can be managed as a software. This makes it possible for a plurality of network functions to share one server resource, or to execute one network function on a plurality of servers in a distributed manner. This makes costs lower in terms of both CAPEX and OPEX, and also makes network services have flexibility to environmental fluctuations. Figure 1 shows an example of an NFV system.

A flow receiving the NFV service may have a Service Function Chaining (SFC) request, which describes the order of VNFs to be applied. Figure 1 depicts how VNFs are applied to a flow in an NFV system in accordance with the SFC request. In order to operate the NFV system efficiently, the NFV management should be adaptive and dynamic. The NFV management includes decision of placement of VNFs on servers, resource allocation to VNFs, and flow routing in accordance with SFC requests. Furthermore, in order to quickly respond to environmental fluctuations and to maintain the scalability of NFV services, a distributed control is more suitable than a cen-

tralized one [8]. One way to achieve such behaviors is to exploit biochemical mechanisms with autonomous dispersibility and self-organizing properties.

Our research group has proposed a construction method of service space in virtualized network system based on biochemically-inspired tuple space model [9, 10]. In this method, the server is represented as a tuple space. Other factors such as service requests, service demands and server resources are represented as chemical substances in the tuple space. The behaviors in the virtualized network system are described by biochemical reactions in tuple spaces. Configuring a network by connecting a plurality of tuple spaces, this method can express movement and diffusion of services and requests in a network system including a plurality of servers.

By applying this method to an NFV environment, it is possible to determine a VNF to be provided in each distributed server, share server resources by a plurality of types of VNFs and execute VNFs in a distributed manner. Since biochemical reaction equations are defined and executed independently in each tuple space, this method can realize the autonomous and decentralized behaviors in the NFV system. In the previous studies, the behaviors of the proposed method have been confirmed with computer simulations [11, 12]. Also, the proposed method has been implemented and evaluated in a simple experimental environment. However, the applicability to the actual NFV system has not been clearly shown.

The standardization activities of NFV defines a framework composed of multiple components such as NFV Management and Orchestration (MANO) and NFV Infrastructure (NFVI), and the implementation has been progressing in recent years. NFV reference architectural framework, which is defined in [13], is shown in Figure 2. Each component is modularized, and the administrator can build the NFV environment by selecting and combining the required components from various types of implementations. Furthermore, the integration of Software Defined Networking (SDN) with the NFV framework is discussed in [14]. By implementing the above-mentioned proposed method as components on the standardized framework, it is possible to confirm the applicability and effectiveness of the proposed method in the actual NFV environment.

In this thesis, we implement the proposed method as components on the NFV framework in cooperation with other components. In detail, we exploit an NFV environment using Open Platform for NFV (OPNFV) [15], which is an open source project aiming to build a software environment to implement the NFV system, and implement the proposed NFV management system on the framework. The proposed method is implemented as four components, which work as a

VNF and modules in NFV Orchestrator (NFVO), Virtualized Infrastructure Manager (VIM), and SDN controller. According to the result of biochemical reactions with observed statistics as input, the proposed system allocates CPU resources to VNFs and configures SDN switches for packet forwarding dynamically and adaptively.

By extensive experimental evaluations, we confirm that the proposed NFV management system works properly for realizing video streaming services that receives many concurrent service requests. In detail, for the traffic generated using `httperf` [16], we confirm that allocation of CPU resources to VNFs and flow routing in accordance with the SFC request by the OpenDaylight [17] application works properly, meaning that all packets of the flows are processed by the appropriate VNF without large latency. For determining flow paths, the proposed system exploits Network Service Header (NSH) [18], which is now under standardization activities. We confirm that the proposed system can dynamically and adaptively distribute incoming flows to multiple VNFs for load balancing.

The rest of this thesis is organized as follows. Section 2 explains the tuple space model using biochemical reactions and how to apply the model to the NFV system. Section 3 describes the implementation design of the proposed method on the NFV framework. Section 4 describes a network constructed using OPNFV and a specific implementation of the proposed method. Section 5 explains the preparation for experimental evaluations and shows the results of preliminary experiments. Section 6 presents the experimental evaluation results and gives discussions. Finally, Section 7 concludes this thesis and presents some directions for future research.

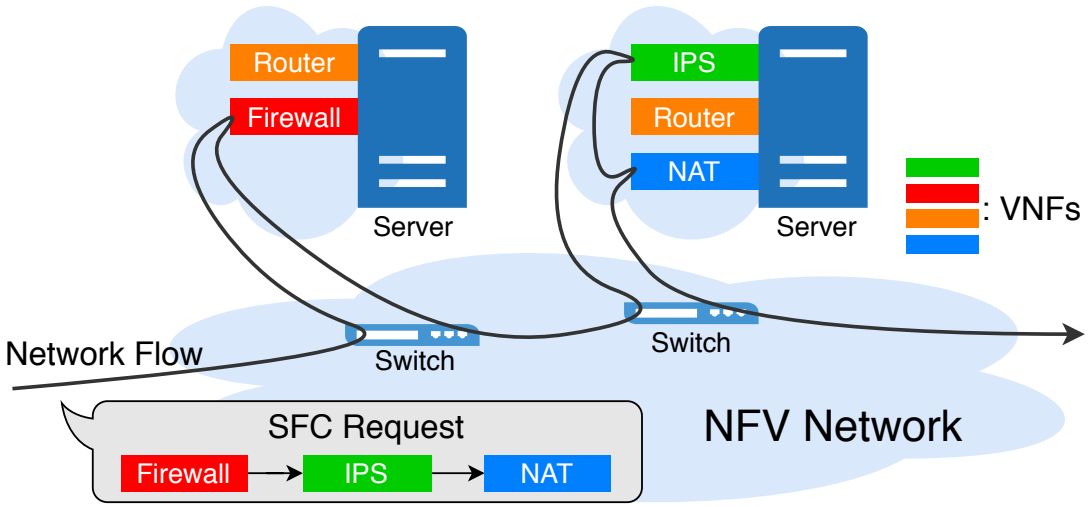


Figure 1: NFV system

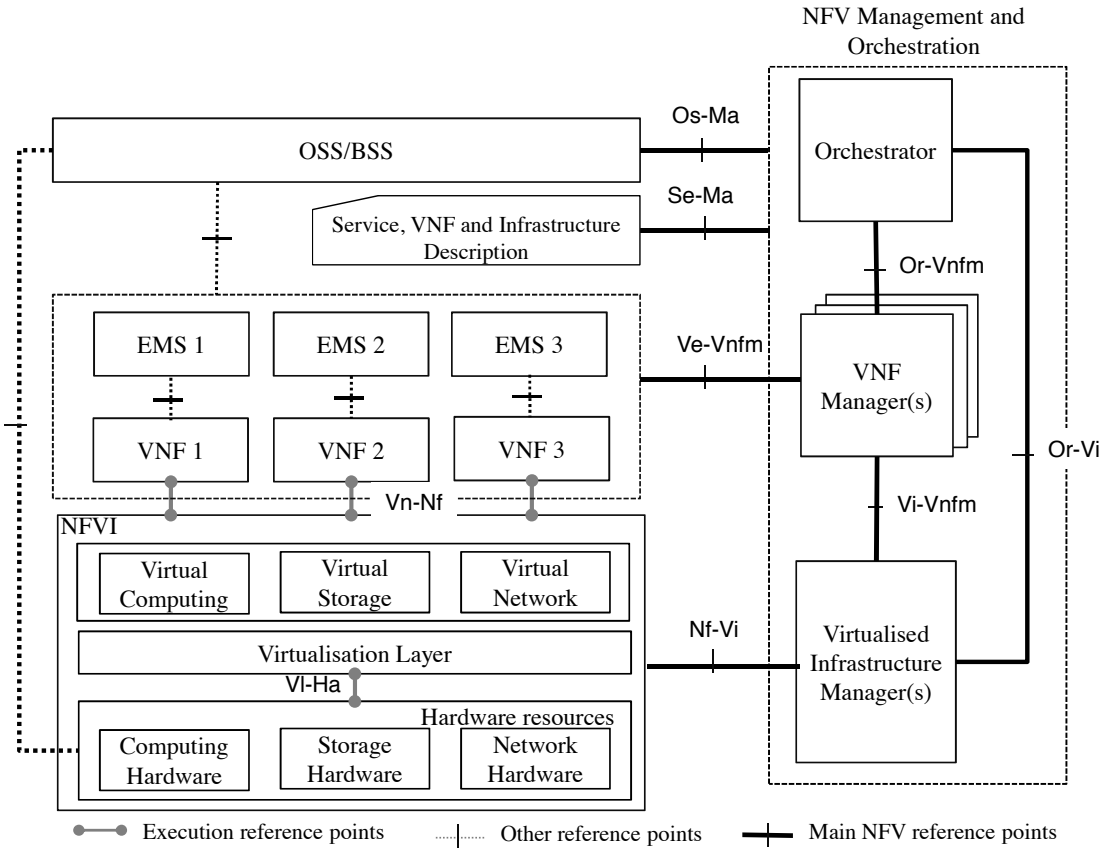


Figure 2: NFV reference architectural framework [13]

2 NFV Management System Based on Tuple Space Model using Biochemical Reactions

In this section, we briefly introduce the NFV management system based on biochemical reactions in tuple spaces, which was proposed in [12].

2.1 Tuple Space Model

A tuple space model in [9] is one of the models that describes a distributed system. Figure 3 depicts the tuple space model in this thesis. A component of the distributed system is modeled as a tuple space. Each tuple space is defined as the place where biochemical reactions occur. Then, tuples in the tuple space correspond to chemical substances, and the amount of the tuples corresponds to the concentration of chemical substances. The concentration is increased and decreased by biochemical reactions as defined in tuple spaces.

A rate of a chemical reaction is determined by the product of the concentration of each reactant and the rate constant defined in the chemical reaction equation. By the following expression, we consider a chemical reaction where defines X and Y are reactants, Z is a product, and a is a reaction rate constant.

$$X \,|\, Y \xrightarrow{a} Z$$

In this reaction, when the concentrations of reactants X and Y are respectively x and y, the reaction rate is axy. Due to this property, the reaction rates in chemical reactions are controlled by the concentrations of reactants and the rate constant defined in the chemical reaction equations. In addition, a network can be configured by connecting multiple tuple spaces. It is possible to achieve the interaction among multiple tuple spaces by defining biochemical reactions that describe the diffusion and movement of tuples among tuple spaces. Since biochemical reactions in each tuple space occur independently, autonomous and decentralized behaviors in a networked system can be described.

2.2 Application to NFV System

Figure 4 depicts how to apply the tuple space model to the NFV system. A tuple space is associated with a server that deploys and executes VNFs. Tuples in the tuple spaces correspond to demands of

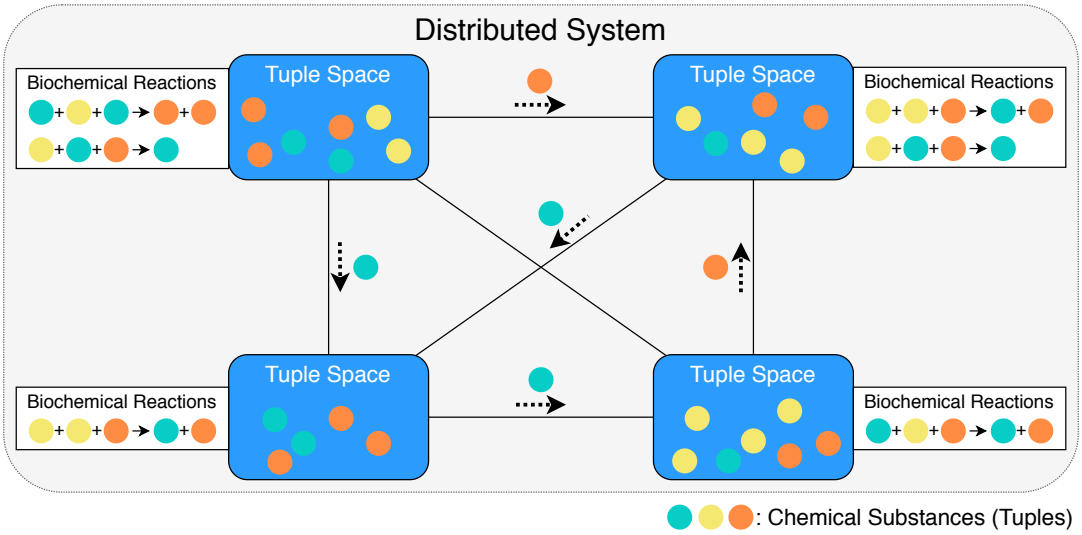


Figure 3: Tuple space model with biochemical reaction

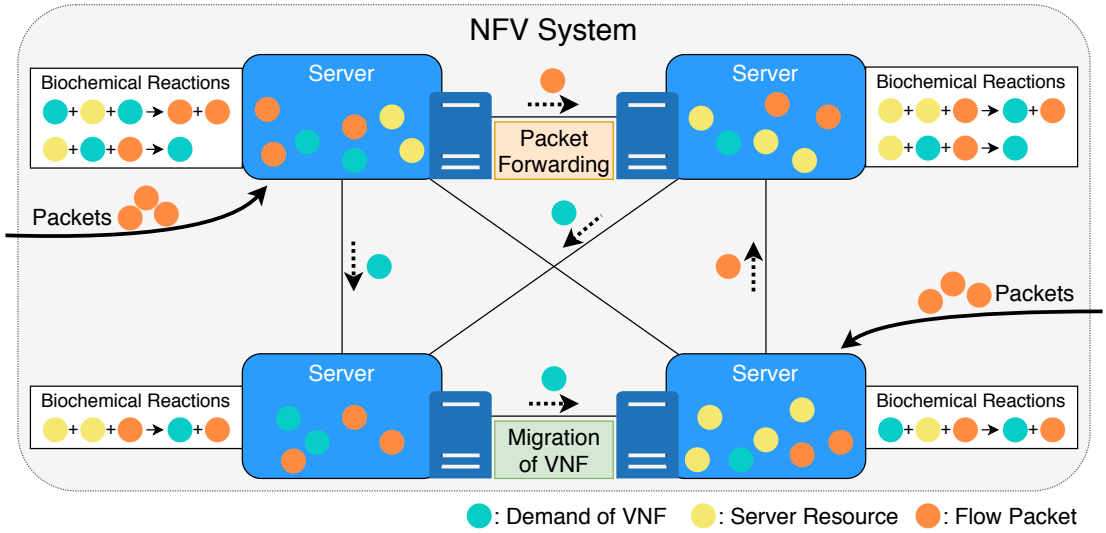


Figure 4: Application of tuple space model to NFV system

VNFs, flow packets, server resources, and so on. The behaviors in the NFV system are described by biochemical reaction equations in tuple spaces. Biochemical reaction equations are defined to adaptively and autonomously determine placement of VNFs on the servers, resource allocation to each VNF, and flow paths in accordance with SFC requests, the amount of traffic of the flows, and the amount of server resources. Table 1 shows the correspondence between the tuple space model and the NFV system.

An SFC request for a flow, described by a series of VNFs, $f_0, f_1, f_2, ..., f_n$ is described as follows.

$$c = \{f_0, f_1, f_2, ..., f_n\} \tag{1}$$

When VNF f_0 is executed to the flow with SFC request c, c changes as follows.

$$c \leftarrow c \backslash \{f_0\} = \{f_1, f_2, ..., f_n\} \tag{2}$$

A VNF that is executed at first in c is denoted by $f_h(c)$. In this thesis, subscripts f and c of chemical substances represent a VNF and an SFC request respectively. In what follows, we present biochemical reaction equations that achieve various behaviors for the NFV system.

2.2.1 Resource Allocation and Execution of VNFs

The placement of VNFs on physical/virtual servers and resource allocation to them should be determined according to the demands of VNFs. In detail, VNFs with low demand should have low priority to be executed in the servers, and vice versa. When a packet of a flow with SFC request c arrives at a server providing VNF $f_h(c)$, VNF $f_h(c)$ is applied to the packet. Then, when c includes multiple VNFs, c changes as in Equation (2). On the other hand, when c is composed of one VNF, the packet leaves the system. These behaviors can be described by Reactions (3) and (4).

$$VNF_{f_h(c)} | PKT_c \rightarrow \begin{cases} VNF_{f_h(c)} | VNF_{f_h(c)} \\ \quad | toserve(VNF_{f_h(c)}, PKT_c) | PKT_{c \backslash f_h(c)} & (c \backslash f \neq \emptyset) \\ VNF_{f_h(c)} | VNF_{f_h(c)} \\ \quad | toserve(VNF_{f_h(c)}, PKT_c) & \text{(otherwise)} \end{cases} \tag{3}$$

$$VNF_f \rightarrow 0 \tag{4}$$

Substance VNF_f with a large concentration value means that execution of the VNF f is highly demanded. Substance PKT_c represents a packet constituting a flow with SFC request c. Substance $toserve(\,VNF_f, PKT_c)$ indicates a result of applying VNF f to a packet of a flow with c. Reaction (3) indicates that VNF $f_h(c)$ is executed to packets of a flow with c, and the concentration of $VNF_{f_h(c)}$ increases to represent the demand increase for VNF $f_h(c)$. Reaction (4) indicates that $VNF_{f_h(c)}$ decays to represent the demand decrease for VNF $f_h(c)$.

As mentioned earlier, the execution rate of a biochemical reaction is determined in proportion to the product of the concentration of each reactant of the reaction. Therefore, in Reaction (3), as the concentrations of $VNF_{f_h(c)}$ and PKT_c increase, the reaction rate increases without limitation. However, performances of actual servers are limited by server resources such as CPU performance and memory size. Therefore, Reaction (3) does not fully describe the behavior of the NFV system. To describe the above constraints, enzyme-catalyzed reactions in biochemical reactions are exploited [19]. In enzyme-catalyzed reactions, the reaction rate can be controlled by the concentration of the catalyst, which does not affect the reaction itself. The basic equation of the enzyme-catalyzed reaction is shown in Equation (5), where E is an enzyme, S is a substrate, ES is an enzyme-substrate complex, and P is a product.

$$E|S \leftrightarrows ES \rightarrow E|P \tag{5}$$

The execution rate of the enzyme-catalyzed reaction is calculated by introducing an enzyme-substrate complex into the reaction [20]. Here, Reaction (3) is extended into the following Reactions (6)–(8), which can describe the constraints of server resources by applying the enzyme-catalyzed reactions mechanism.

$$VNF_f|RSRC \;\rightleftarrows\; RS\_VNF_f \tag{6}$$

$$RS\_VNF_{f_h(c)}|PKT_c \;\rightleftarrows\; MEDIATE_c \tag{7}$$

$$MEDIATE_c \;\rightarrow\; \begin{cases} VNF_{f_h(c)}|\,VNF_{f_h(c)}|RS\_VNF_{f_h(c)} & \\ \quad |toserve(\,VNF_{f_h(c)}, PKT_c)|PKT_{c\backslash f_h(c)} & (c\backslash f \neq \emptyset) \\ VNF_{f_h(c)}|\,VNF_{f_h(c)}|RS\_VNF_{f_h(c)} & \\ \quad |toserve(\,VNF_{f_h(c)}, PKT_c) & (\text{otherwise}) \end{cases} \tag{8}$$

The concentrations of substances $RSRC$, RS_VNF_f, and $MEDIATE_c$ respectively represent the amount of available resources of a server, the amount of server resources reserved for VNF f,

and the amount of server resources allocated to the flow packets with SFC request c. Reaction (6) indicates that server resources are reserved in accordance with the demand of each VNF, and that the reservation is controlled by the concentration of $RSRC$. Reaction (7) indicates that the server resources reserved for the VNF is allocated to the packet. Reaction (8) indicates that the VNF is executed to the packet and the demand for the VNF increases.

2.2.2 Diffusion of VNFs

A highly-demanded VNF is required to be executed by a plurality of servers, and the servers that execute the VNF should be located in places with high demand. To describe the diffusion of highly-demanded VNFs to other servers, the following reaction is introduced.

$$VNF_f \quad \rightarrow \quad VNF_f^{\rightsquigarrow} \tag{9}$$

This reaction indicates that a highly-demanded VNF in a server diffuses to the surrounding connected servers. When the destination server of the diffusion has packets for VNF f, they are processed in accordance with Reactions (6)–(8), resulting the evolution of VNF_f at the server. On the other hand, when the destination server has no corresponding packet, VNF_f would decay in accordance with Reaction (4). These behaviors mean that the diffused VNFs grow only on servers that have a corresponding demand for the VNF.

2.2.3 Packet Forwarding

For packets that remain unprocessed on a server due to a lack of server resources, the packets should move to other servers that can process. The forwarding direction of packets should be determined so that the packets would approach servers executing the corresponding VNFs with enough server resources. To achieve these behaviors, a gradient field is exploited to determine the moving directions of packets. A gradient field for each VNF is constructed based on the demand of VNFs and the available resources on each server. For that purpose, Reactions (10)–(14) are

introduced.

$$VNF_f|RSRC \rightarrow VNF_f|RSRC|GRAD_f \quad (10)$$

$$RS\_VNF_f \rightarrow RS\_VNF_f|GRAD_f \quad (11)$$

$$GRAD_f \rightarrow 0 \quad (12)$$

$$GRAD_f \rightarrow GRAD_f^{\rightsquigarrow}(GRAD_f^{-}) \quad (13)$$

$$PKT_c \rightarrow PKT_c^{\rightsquigarrow}(GRAD_{f_h(c)}^{+}) \quad (14)$$

Substance $GRAD_f$ constructs a gradient field for VNF f. Reaction (10) indicates that $GRAD$ is generated at a rate proportional to the concentrations of VNF and $RSRC$. Reaction (11) indicates that $GRAD$ is generated at a rate proportional to the concentrations of RS_VNF. Note that this is modified from the one in [12] since VNF was double counted in the original one. Reaction (12) indicates that $GRAD$ decays at a rate proportional to its concentration. Reaction (13) indicates that $GRAD$ spreads to the surrounding servers with the smaller concentration of $GRAD$. The diffusion direction of the substance forming the gradient field is stochastically determined in accordance with the concentration ratio of $GRAD$ in the connected tuple spaces. Therefore, the gradient field is constructed so that the server providing VNFs with enough resources becomes a summit with the largest concentration of $GRAD$, and the surrounding servers have smaller concentrations of $GRAD$ in accordance with the distance from the summit. Reaction (14) describes the movement of PKT to the surrounding servers with large concentration of $GRAD$. The forwarding direction of a packet is also stochastically determined in accordance with the concentration ratio of $GRAD$ in the connected tuple spaces. Figure 5 depicts the movement of a packet with SFC request $\{f_0, f_1, f_2\}$. The gradient fields are respectively generated for each VNF. First, a packet moves in the direction of the summit of the gradient field for f_0. Next, after the packet is applied with f_0, it moves in the direction of the summit of the gradient field for f_1. Then, after the packet is applied with f_1, it moves in the direction of the summit of the gradient field for f_2. Finally, after the packet is applied with f_2, the packet leaves the system.

2.2.4 Summary

We summarize all reaction equations in the NFV system in Reactions (15)–(24), where F is the set of all VNFs provided by the NFV system, C is the set of all SFC requests that may exist in the

Table 1: Correspondence between tuple space model and NFV system

Tuple Space Model	NFV System
Tuple Spaces	General-purpose Physical/Virtual Servers
Chemical Substances	Demand of VNFs, Flow Packets, Server Resources, Gradient Fields for VNFs
Biochemical Reactions	Apply VNFs to Packets, Demand Increase of VNFs, Decay of VNFs, Server Resource Allocation to VNFs, Diffusion of VNFs, Packet Forwarding

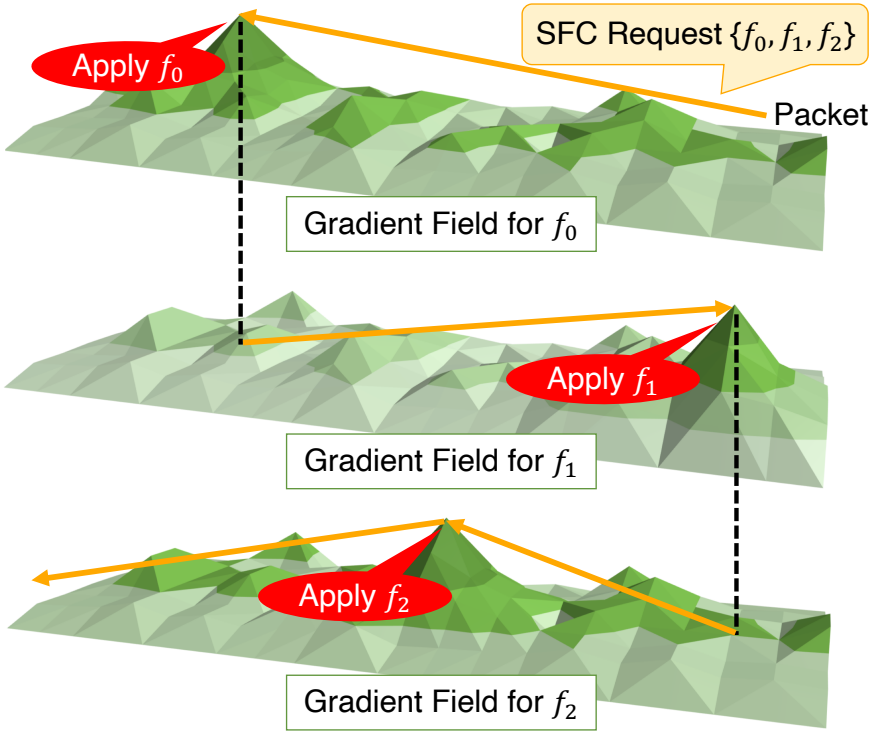


Figure 5: Movement of a packet in accordance with gradient fields

NFV system, and c_0–c_{11} are the reaction rate constants for Reactions (15)–(24). These reactions are defined for each tuple space.

$$\forall f \in F, VNF_f | RSRC \underset{c_1}{\overset{c_0}{\rightleftarrows}} RS\_VNF_f \tag{15}$$

$$\forall c \in C, RS\_VNF_{f_h(c)} | PKT_c \underset{c_3}{\overset{c_2}{\rightleftarrows}} MEDIATE_c \tag{16}$$

$$\forall c \in C, MEDIATE_c \xrightarrow{c_4} \begin{cases} VNF_{f_h(c)} | VNF_{f_h(c)} | RS\_VNF_{f_h(c)} \\ \quad | toserve(VNF_{f_h(c)}, PKT_c) | PKT_{c \backslash f_h(c)} & (c \backslash f \neq \emptyset) \\ VNF_{f_h(c)} | VNF_{f_h(c)} | RS\_VNF_{f_h(c)} \\ \quad | toserve(VNF_{f_h(c)}, PKT_c) & \text{(otherwise)} \end{cases} \tag{17}$$

$$\forall f \in F, VNF_f \xrightarrow{c_5} 0 \tag{18}$$

$$\forall f \in F, VNF_f \xrightarrow{c_6} VNF_f^{\rightsquigarrow} \tag{19}$$

$$\forall f \in F, VNF_f | RSRC \xrightarrow{c_7} VNF_f | RSRC | GRAD_f \tag{20}$$

$$\forall f \in F, RS\_VNF_f \xrightarrow{c_8} RS\_VNF_f | GRAD_f \tag{21}$$

$$\forall f \in F, GRAD_f \xrightarrow{c_9} 0 \tag{22}$$

$$\forall f \in F, GRAD_f \xrightarrow{c_{10}} GRAD_f^{\rightsquigarrow}(GRAD_f^-) \tag{23}$$

$$\forall c \in C, PKT_c \xrightarrow{c_{11}} PKT_c^{\rightsquigarrow}(GRAD_{f_h(c)}^+) \tag{24}$$

3 Implementation Design of Proposed Method with NFV Framework

In this section, we describe the implementation design of the proposed method with the NFV framework proposed by ETSI ISG [13, 14].

3.1 NFV Framework and its Integration with SDN

Figure 2 illustrates the NFV framework [13]. As such, three main working domains are identified in NFV.

Virtualized Network Functions (VNFs) VNFs are software implementations of network functions that run on virtual machines.

NFV Infrastructure (NFVI) NFVI is an infrastructure for running VNFs and manages both physical and virtual resources.

NFV Management and Orchestration (NFV MANO) NFV MANO manages life cycles and resource orchestration of NFVI and VNFs.

NFV MANO architectural framework identifies the following functional blocks.

Virtualized Infrastructure Manager (VIM) VIM is responsible for controlling and managing the NFVI compute, storage and network resources.

NFV Orchestrator (NFVO) NFVO has two main responsibilities: the orchestration of NFVI resources across multiple VIMs; the lifecycle management of network services.

VNF Manager (VNFM) VNFM is responsible for the lifecycle management of VNF instances, such as creation, update, and deletion.

Furthermore, the integration of SDN with the NFV framework is discussed in [14]. Figure 6 shows NFV framework with SDN integration. SDN controller is used to manage network resources in NFVI. SDN switch is included in physical and virtual network resources in NFVI.

3.2 Implementation Design of Proposed Method

In order to apply the proposed method to the actual NFV environment, we implement the proposed method on the SDN-integrated NFV framework, described in the previous subsection. This sub-

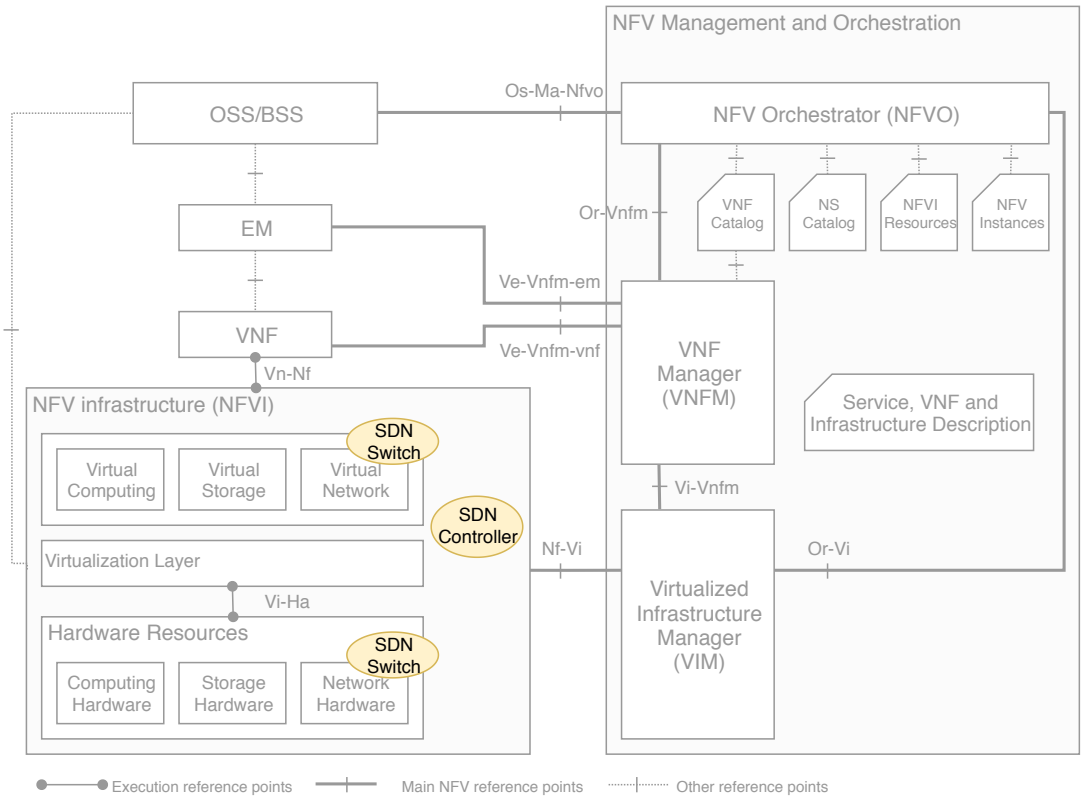


Figure 6: NFV framework and its integration with SDN

section describes the implementation design of the proposed method on the framework described in the previous subsection.

In the proposed method, NFV management such as resource allocation to VNFs and flow routing is conducted based on the results of biochemical reactions. In order to implement the proposed method as components of the framework, the proposed method is divided into the following functions.

Calculation of biochemical reactions

It calculates the biochemical reactions for each tuple space, corresponding the server operating VNFs. For the autonomous decentralized control, this is preferably executed in a distributed manner. Therefore, this function should be realized as a VNF.

Acquisition of flow rates and remaining data sizes waiting to be processed in VNFs

When a new flow enters the system, a concentration of PKT, corresponding to the size of packets of the flow, needs to be injected into the corresponding tuple space. Also, the unprocessed packets in the VNF need to be considered. These mean that flow rates and remaining data sizes waiting to be processed in VNFs are interpreted into the concentration of PKT in the corresponding tuple space. Acquisition of flow rates and the remaining data sizes waiting to be processed in VNFs should be realized as a function in VIM since it is necessary to acquire the statistics from SDN switch.

Resource allocation to VNFs

This function allocates resources to the VNF according to the concentrations of substances in tuple spaces. Resource allocation to VNFs should be realized as a function in VIM.

Packet forwarding

Packets should be forwarded in the system according to the concentrations of $GRAD$ in the network. This function is composed of three sub-functions: management of the flow path candidates, selection of a flow path based on the concentrations of $GRAD$, and configuring SDN switches for packet processing. The first sub-function is strongly related to the SFC management, it should be realized as a function in NFVO. The second and third sub-functions should be realized as functions in SDN controller.

Exchange of the concentrations among components

As described above, since the tuple spaces are executed in a dispersed manner, it is necessary that the components exchange the concentrations as necessary. This is realized by directly connecting the components or via other components. These exchanges should be via one component so that the environmental changes are appropriately taken into consideration. Since this function is related to the orchestration of the entire system, it should be realized as a function in NFVO.

Mutual conversion between the concentrations and the actual system values

For components using the concentrations, mutual conversion between the concentrations and the actual system values is required. This is realized by each component using the concentrations or the component that the concentrations are exchanged via. Since the function that exchanges the concentrations among components can aggregate the concentrations, the conversion function should be integrated to the exchange function.

Based on the above discussions, we construct the following four components executing these functions. Figure 7 depicts the placement of the components and information flows on the NFV framework.

Biochemical Reactions Tuple Space (BRTS)

BRTS is responsible for calculation of biochemical reactions for each tuple space. For distributed execution, this is implemented as a VNF for each tuple space.

Biochemical Reactions Manager (BRM)

BRM is responsible for acquisition of flow rates and remaining data sizes waiting to be processed in VNFs and resource allocation to VNFs. Specifically, it acquires statistics for each VNF from SDN switches and tells the VNF resource allocation to NFVI. This is implemented as a module in VIM.

Biochemical Reactions Controller (BRC)

BRC is responsible for packet forwarding. Specifically, it performs probabilistic path selection, based on the concentrations of $GRAD$ in the tuple spaces, and configures SDN switches for packet processing, as explained in Subsection 4.3. This is implemented as a module in SDN controller.

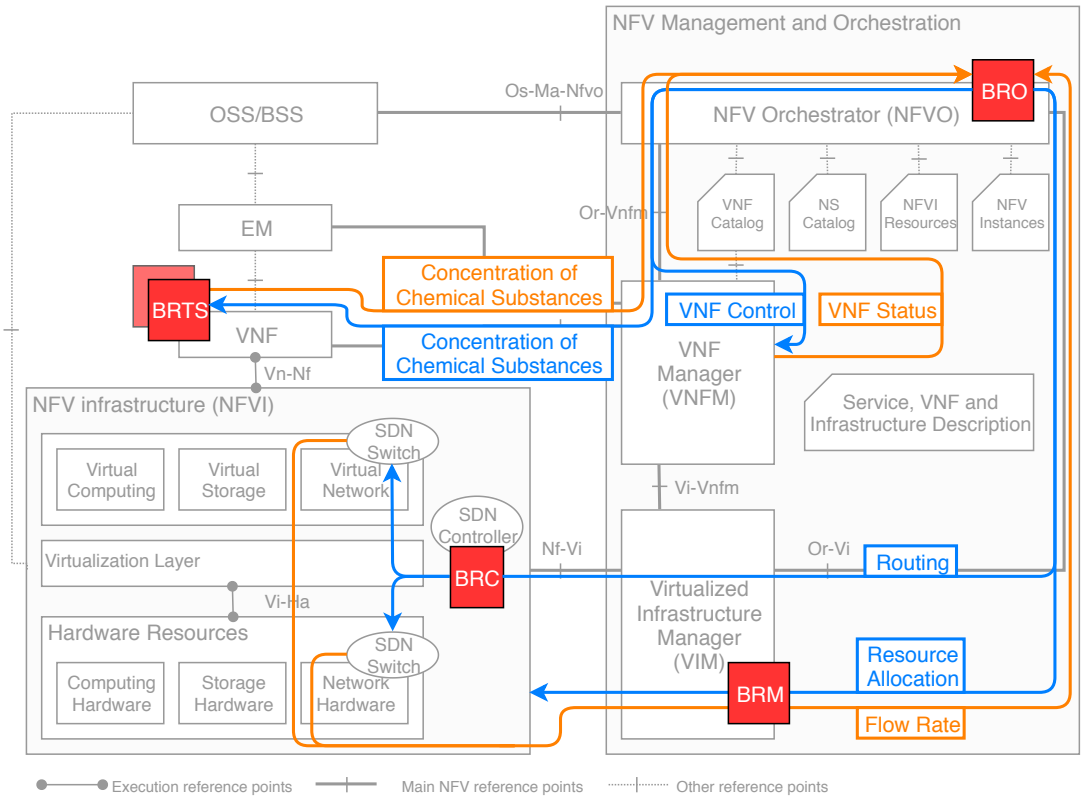


Figure 7: Compoment placement of the proposed system and information flows

Biochemical Reactions Orchestrator (BRO)

BRO orchestrates the entire proposed system and is responsible for exchange of the concentrations among components and mutual conversion between the concentrations and the actual system values. Specifically, it acquires the concentrations from BRTSs, convert them to the resources to allocate for VNFs, and tells the values to BRMs. Besides, it acquires the statistics from BRMs, converts them to the concentration of PKT, and tells the values to BRTSs. The exchange of substances between BRTSs is via this component. In addition, BRO gives BRC the concentrations of $GRAD$ and the flow path candidates. This is implemented as a module in NFVO.

4 Implementation

In this section, we describe the details of the implementation of the proposed method.

4.1 OPNFV

Implementation of NFV is in progress in open source projects such as CloudNFV [21] and Open-MANO [22]. In this thesis, we implement the NFV environment using OPNFV [15]. OPNFV is an open source project founded by the Linux Foundation, which aims to implement the entire NFV framework by integrating existing open source projects such as OpenStack [23]. In OPNFV, an NFV environment is implemented using a Linux kernel virtualization platform called Kernel-based Virtual Machine (KVM) [24], and the environment is composed of a plurality of virtual machines and virtual switches.

Figure 8 depicts a typical NFV environment constructed by OPNFV. There are two types of virtual machines: Undercloud and Overcloud. Undercloud is a virtual machine that contains components for provisioning and managing OpenStack nodes required for building an OpenStack environment, and does not correspond to any component in the NFV framework. On the other hand, Overcloud is an OpenStack platform environment built by the Undercloud, and is composed of two types of nodes: Compute nodes and Controller nodes. Compute nodes are used to implement NFVI and VNFs, and VNFs are implemented as virtual machines on the compute nodes. Controller nodes are used to implement NFV MANO and SDN controller, and perform resource management and control of the virtual machines on the compute nodes using OpenStack. They also control flow paths using the SDN controller.

The following four types of networks exist in the NFV environment.

admin network Network used by control plane

tenant network Network used for tenant traffic

storage network Network used for storage I/O

public network Network to connect the system to external networks and to OpenStack dashboard

When a virtual machine is created in a compute node using OpenStack, virtual switches and virtual networks are automatically configured on Controller nodes and Compute nodes. A virtual

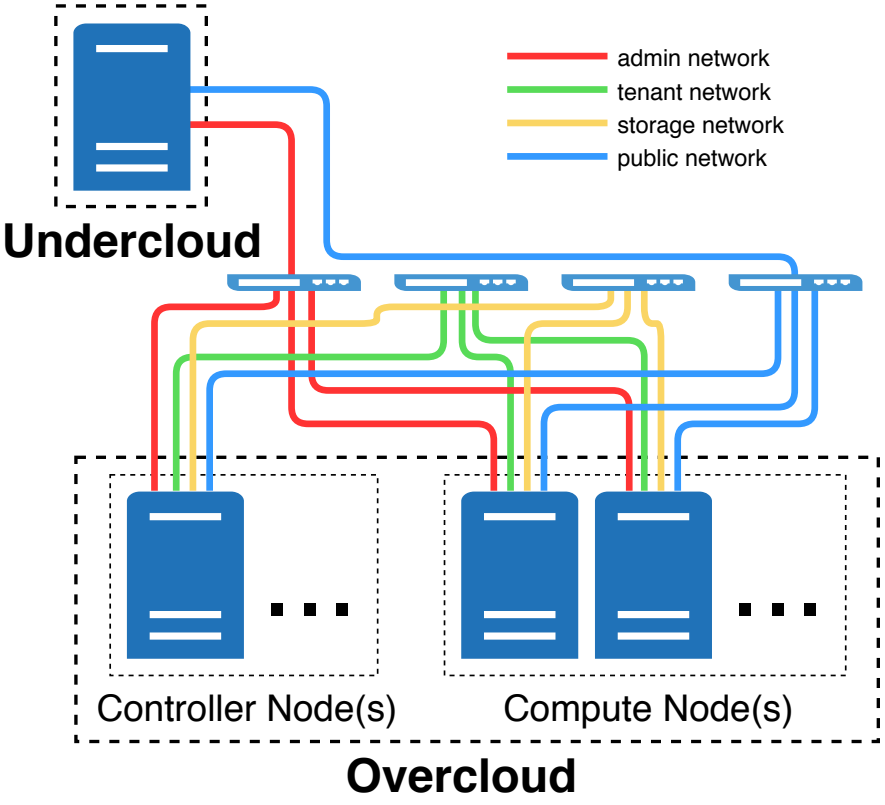


Figure 8: A typical NFV environment constructed by OPNFV

machine running in the OpenStack cloud is called an instance, which is connected to the tenant network and the public network.

In this implementation, Open vSwitch (OVS) [25] is used as SDN switch, and OpenDaylight [17] is used as SDN controller. In OpenDaylight, OpenFlow is used as an SDN protocol.

4.2 NFV Management System Based on Biochemical Reactions

4.2.1 Overview

A system that implements the proposed method basically performs processing at fixed intervals, that is denoted as *execution time step* in what follows. This is to avoid an increase in the calculation overhead of execution of biochemical reactions. Figure 9 depicts a sequence diagram of the proposed system. BRO, BRTS, and BRM are inter connected by TCP connections and proceed synchronously. In the figure, the red section is the corresponding process, which is processed for each tuple space and each execution time step. The process related to BRC is performed independently from the above process when an unknown flow enters the system. In the figure, the green section indicates the corresponding process.

The outline of BRO, BRTS, and BRM process and communication contents are described below. A TCP connection between BRO and BRTS is on the public network, and that between BRO and BRM is on the admin network. Note that the processes in tuple spaces are executed in a parallel fashion.

1. For each tuple space, the following four steps are conducted.
    (a) BRO establishes a TCP connection to BRTS.
    (b) BRO tells BRTS the length of execution time step.
    (c) BRO establishes a TCP connection to BRM.
    (d) BRO tells BRM a list of process IDs for each VNF that the BRM manages.
2. For each execution time step, the following steps are conducted.
    (a) When there are diffusion substances buffered by BRO, determine the diffusion destinations.

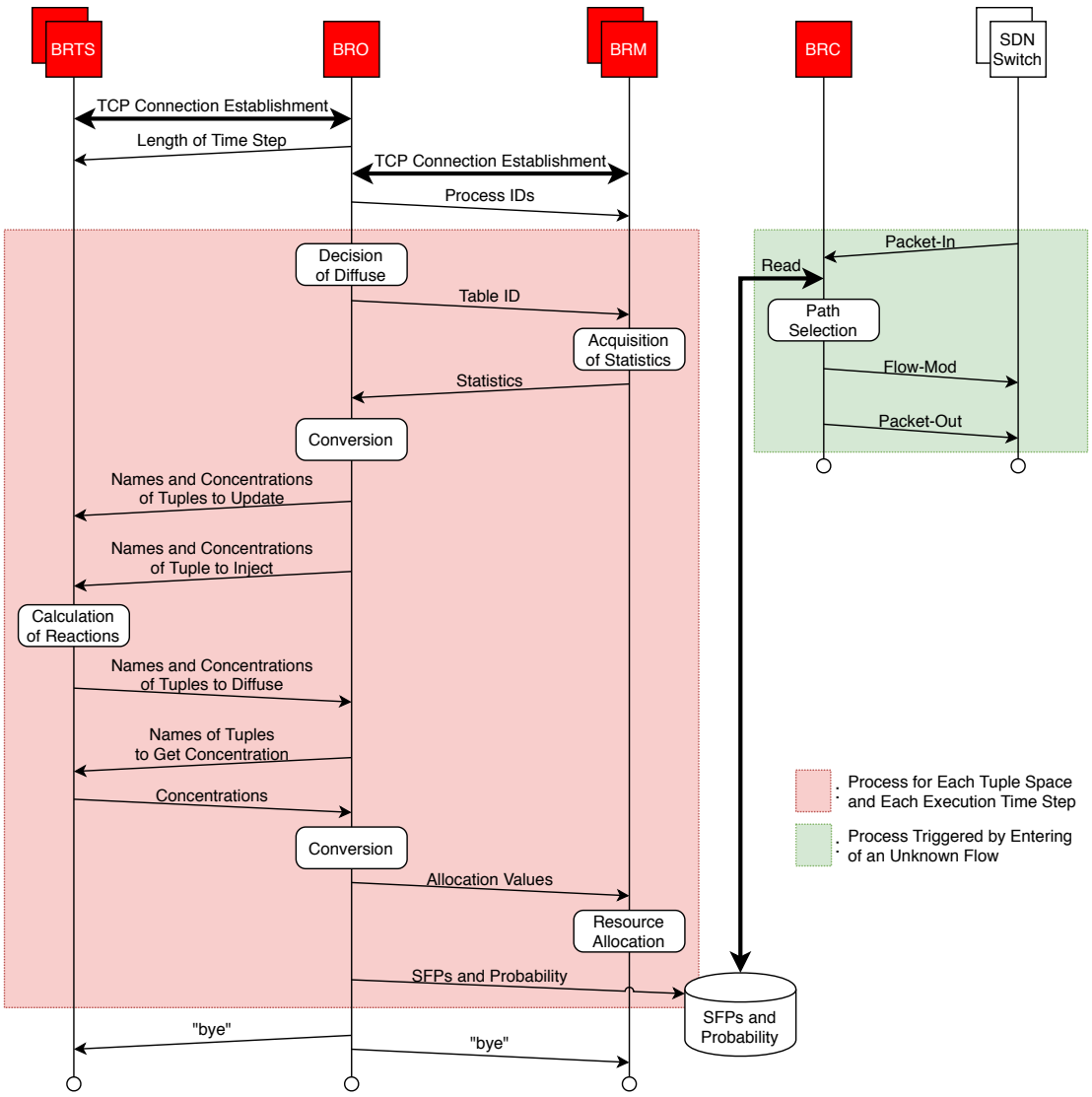


Figure 9: A sequence diagram of the proposed system

(b) For each tuple space, the following sub-steps are executed.

i. BRO tells BRM a list of flow table IDs that have the necessary statistics.

ii. BRM obtains statistics of the requested flow tables, and tells BRO a list of the number and the size of packets applied to those flow tables.

iii. BRO converts the obtained statistics into the concentration of PKT.

iv. BRO tells BRTS a list of the the name and the concentration of substances whose concentration should be updated.

v. BRO tells BRTS a list of the the name and the concentration of substances whose concentration should be added.

(c) For each tuple space, BRTS calculates biochemical reactions to update each substance concentration.

(d) For each tuple space, the following sub-steps are executed.

i. BRTS tells BRO a list of the the name and the concentration of substances which should be diffused.

ii. BRO tells BRTS a list of the the name and the concentration of substances whose concentration needs to be obtained.

iii. BRTS tells BRO a list of the concentration of requested substances.

iv. BRO converts the concentration of $MEDIATE$ into CPU resource allocation for VNF.

v. BRO tells a list of allocation values of CPU resource allocation for each VNF.

vi. BRM executes CPU allocation for each VNF.

(e) BRO updates the file where the SFPs handled by the system and their selection probabilities calculated from the concentration of $GRAD$ in each tuple space are described.

The details of the processing of each component and its implementation are described in the following subsections.

4.2.2 Biochemical Reactions Tuple Space (BRTS)

BRTS is responsible for calculating biochemical reactions. This component is implemented as a VNF.

Definition of biochemical reactions

Biochemical reactions are defined in a file outside the program and is read at the start of BRTS execution. The definitions include reactants, products, and their reaction rate constants. Also, the initial value of each substance concentration is defined separately.

τ-Leaping method

In order to simulate the biochemical system, we exploit τ-leaping method in [26], which is one of stochastic simulation algorithms that can capture the inherent stochasticity in many biochemical systems such as the chemically reacting system and the cell system. The method collectively executes the biochemical reactions at each execution time step τ so that the time change of the concentration of the chemical substance can be obtained in a short time. We briefly explain the procedures of τ-leaping algorithm for a server in the proposed method as follows.

Step 1 Set τ for the time step of the simulation

Step 2 Calculate the reaction rates of biochemical reactions by the product of reactants concentration and reaction rate constants

Step 3 Determine the number of executions of biochemical reactions during time τ, using a Poisson random variable

Step 4 Execute biochemical reactions at the number of times derived in Step 3, and update the concentrations of each substance

Step 5 Progress simulation time by τ

Step 6 Return to Step 2

As the value of τ increases, the simulation can proceed faster while the results becomes different from the actual behavior. So τ should be carefully chosen. There is an algorithm that

decides the optimal τ [27], but it takes a long calculation time when executing the program. In [11], the value of τ was determined by performing some preliminary experiments and adjusted so that the simulation ended quickly with little error. In this thesis, the length of execution time step of the proposed system is set identical to τ.

Further, when the τ is set to a large value, a problem may occur that concentrations of some substances take negative values. One example of the solution for this problem is shown in [28]. In this thesis, when the concentration of a substance takes a negative value, we conduct the τ-leaping method twice with halved value of τ recursively.

Addition of concentration by substance injection

The substance to inject at a certain execution time step is buffered and added as a concentration value immediately before the calculation of the τ-leaping method. The injection amount is determined according to the value of τ.

4.2.3 Biochemical Reactions Manager (BRM)

BRM is responsible for acquisition of statistics and resource allocation to VNFs. This component is executed on each Compute node implementing VNFs.

Acquisition of the statistics

In OpenFlow protocol, processing statistics are held as flow tables in switches. There are two ways to acquire the statistics of switches: using ODL API, and directly executing OVS commands on the machine running the switch. Acquiring statistics using ODL API, the statistics are updated approximately every three seconds in the environment of this thesis. This resulted in large observational errors. On the other hand, executing OVS commands on the machine running the switch, the statistics are updated almost immediately. Hence, the statistics are acquired by executing OVS commands.

Resource allocation to VNFs

In this thesis, CPU is treated as the resource to be allocated.

In KVM, the CPU resource allocation to a virtual machine can be limited by setting the parameters *cpu period* and *cpu quota*. These parameters determine the upper limit of the

CPU time to the virtual machine. When the total CPU time allocated to the virtual machine reaches the value of cpu quota during the period specified by the cpu period, no more CPU processing time is allocated to the virtual machine until the period of the CPU period ends. However, in order to change the CPU resource allocation limit using this method, it is necessary to restart the VNF server.

Hence, in the implementation of this thesis, the CPU resource allocation is realized by using `cpulimit` [29]. `cpulimit` is a simple program which limits the CPU usage of a process, expressed in percentage. Also, it is able to adapt itself to the overall system load, dynamically and quickly.

4.2.4 Biochemical Reactions Controller (BRC)

BRC is responsible for flow packet routing as described in Subsection 4.3. This component is executed on a Controller node.

The BRC starts the process triggered by a Packet-In message [30] from a switch. A Packet-In message is the message mainly used when the SDN switch inquires of the SDN controller about processing of a packet of an unknown flow in the OpenFlow protocol. The message also contains information about the packet that caused it to be sent. In the implementation of this thesis, a Packet-In message is sent when a flow whose path has not been determined enters the system.

When the BRC receives the Packet-In message, it refers to the data updated by the BRO and the flow path is determined stochastically as explained in Subsection 4.3. Next, based on the selected path, the flow table is modified by sending a Flow-Mod message to the switch. Finally, the packet sent to the BRC as a Packet-In message is corrected if necessary, and the packet is sent out to the switch as a Packet-Out message. Then, following packets of the flow are regarded as known flow packets at the switch.

Note that based on the experiment results, we insert 20 ms wait between sending a Flow-Mod message and a Packet-Out message. The detailed discussion can be found in Subsection 5.6.4.

4.2.5 Biochemical Reactions Orchestrator (BRO)

BRO is responsible for orchestrating of the entire proposed system. This component is implemented on a Controller node.

Calculate the flow rate and the remaining data sizes waiting to be processed in VNFs

From the statistics received from BRMs, the flow rates and the remaining data size waiting to be processed in VNFs are calculated.

From BRMs, statistics about the last-stage flow table that outputs packets from the interface are acquired. From this, the total input size of packets to the VNF and the total output size of packets from the VNF can be obtained. When the total input size of packets to the VNF at the execution time step t is $I(t)$ and the total output size of packets from the VNF at the execution time step t is $O(t)$, the input size of packets to the VNF in a certain execution time step t $d_{in}(t)$ can be calculated by Equation (25), and the remaining data size waiting to be processed in the VNF $d_{inside}(t)$ can be calculated by Equation (26).

$$d_{in}(t) = I(t) - I(t-1) \tag{25}$$

$$d_{inside}(t) = p(t-1) + I(t) - O(t) \tag{26}$$

$d_{in}(0) = 0$ and $d_{inside}(0) = 0$ are defined since $I(t-1)$ and $p(t-1)$ do not exist in the first execution time step. The flow rates can be derived from the result of Equation (25) and the length of execution time step.

In Equation (26), when a packet loss occurs in a VNF, $d_{inside}(t)$ will remain high even after the processing is over. However, this is not considered in this implementation. Additionally, since the statistics recognized by the switch are measured in a state when a packet is first input to the switch, a gap of size may occur depending on the presence or absence of the NSH. This is dealt with by dividing the flow to which the packet is applied depending on the presence or absence of the NSH and applying correction. Furthermore, the packet sent to the SDN controller by the Packet-In message is directly output from the interface by the Packet-Out message, and the flow table is not applied. Therefore, the obtained statistics do not include information on such packet. For this, BRO corrects statistics by reading a file that BRC writes the size of packets output by the Packet-Out message to.

Mutual conversion between the substance concentrations and the actual system values

BRO handles the conversion from the substance concentrations to the actual system values, and vice versa. Therefore, we consider the convergence values of each substance when defining the Reactions (15)–(24). For simplicity, here we assume a server that handles one

VNF and one SFC request. When λ is injection amount of PKT per unit time and the concentrations of VNF, $RSRC$, RS_VNF, $MEDIATE$, PKT, $GRAD$, and $toserve(\,VNF, PKT)$ at the execution time step t are $V(t)$, $Rs(t)$, $Rv(t)$, $M(t)$, $P(t)$, $G(t)$, and $T(t)$, Equations (27)–(33) hold from the chemical kinetics. Note that, the diffusion reaction is not described here for the reason described later.

$$\frac{dV(t)}{dt} = -V(t)Rs(t)c_1 + Rv(t)c_2 + 2M(t)c_5 - V(t)c_6 \tag{27}$$

$$\frac{dRs(t)}{dt} = -V(t)Rs(t)c_1 + Rv(t)c_2 \tag{28}$$

$$\frac{dRv(t)}{dt} = V(t)Rs(t)c_1 - Rv(t)c_2 - Rv(t)P(t)c_3 + M(t)c_4 + M(t)c_5 \tag{29}$$

$$\frac{dM(t)}{dt} = Rv(t)P(t)c_3 - M(t)c_4 - M(t)c_5 \tag{30}$$

$$\frac{dP(t)}{dt} = -Rv(t)P(t)c_3 + M(t)c_4 + \lambda \tag{31}$$

$$\frac{dG(t)}{dt} = V(t)Rs(t)c_8 + Rv(t)c_9 - G(t)c_{10} \tag{32}$$

$$\frac{dT(t)}{dt} = M(t)c_5 \tag{33}$$

Further, since the amount of resources in the entire server is constant, when the amount of resources in the entire server is $R(0)$, Equation (34) always holds.

$$R(0) = Rs(t) + Rv(t) + M(t) \tag{34}$$

From the above equations and that $\frac{dX(t)}{dt} = 0$ is hold when $X(t)$ is in the steady state, the convergence values of each substance other than $toserve(\,VNF, PKT)$ is obtained as in the Equations (35)–(40). Here, the diffusion reaction can be ignored since the input and output should be equal in the steady state.

$$\lim_{t\to\infty} V(t) = \frac{2\lambda}{c_6} \tag{35}$$

$$\lim_{t\to\infty} Rs(t) = \frac{c_2c_6(R(0)c_5 - \lambda)}{c_5(2\lambda c_1 + c_2c_6)} \tag{36}$$

$$\lim_{t\to\infty} Rv(t) = \frac{2\lambda c_1(R(0)c_5 - \lambda)}{c_5(2\lambda c_1 + c_2c_6)} \tag{37}$$

$$\lim_{t\to\infty} M(t) = \frac{\lambda}{c_5} \tag{38}$$

$$\lim_{t\to\infty} P(t) = \frac{(c_4 + c_5)(2\lambda c_1 + c_2c_6)}{2c_1c_3(R(0)c_5 - \lambda)} \tag{39}$$

$$\lim_{t\to\infty} G(t) = \frac{2\lambda(R(0)c_5 - \lambda)(c_2c_8 + c_1c_9)}{c_5c_{10}(2\lambda c_1 + c_2c_6)} \tag{40}$$

Besides, the generation rate of $toserve(VNF, PKT)$ is as shown in Equation (41).

$$\frac{dT(t)}{dt} = \lambda \tag{41}$$

From the above equations, it is expected that in the case of $\lambda \rightarrow R(0)c_5$, the VNF cannot process the packet and the concentration of PKT diverges. Therefore, let $\lambda_{max} = R(0)c_5$ be the maximum value of λ that can be processed. For the sake of simplicity in implementation, $R(0)$ and c_5 are fixed regardless of tuple spaces or VNFs, and do not change from the start of the system.

The conversion from $d_{in}(t)$ bytes, which is the input data size observed at a certain execution time step t, to the concentration value of the PKT to be input at the next execution time step can be expressed by the following Equation (42). In addition, according to the definition of the substance, the concentration of $MEDIATE$ is converted into a CPU resource allocation amount $L\%$ for the VNF. The conversion can be represented by Equation (43).

$$PKT = \frac{8d_{in}(t)}{F_{max}T}\lambda_{max} \tag{42}$$

$$L = \frac{min(M(t), R(0))}{R(0)}(L_{max} - L_{min}) + L_{min} \tag{43}$$

In these equations, T s is the length of an execution time step, F_{max} bps is the maximum flow rate that can be processed when CPU resources are allocated to the VNF at maximum, and $L_{max}\%$ and $L_{min}\%$ are the maximum and minimum amounts of CPU resources that can be allocated to the VNF. For conversion from the remaining data size waiting to be processed in the VNF to the concentration of PKT to be updated at the next execution time step, Equation (42) is used as $d_{in}(t) \leftarrow d_{inside}(t)$. Here, when $d_{inside}(t)$ has a negative value due to an observation error, it is treated as 0. The values of F_{max}, L_{max}, and L_{min} depend on the relationship between the flow rate for the VNF and the CPU usage of the VNF. These values are different for each VNF, and are confirmed experimentally in Subsection 5.6.2.

Exchange of substances between BRTSs

When exchanging the diffusion substances, two buffers are used; the concentration of the substance received by the BRTS diffusion reaction is added to the first buffer, and the diffusion destination is determined at the beginning of the next execution time step, and is moved to the second buffer. For VNF, the diffusion destination is determined at random, and for

$GRAD$, the diffusion destination is determined stochastically according to the inverse ratio of the concentration of $GRAD$ in the connected tuple space. For example, considering the case where the $GRAD$ concentrations in tuple spaces a and b are 1,000 and 3,000, respectively, $GRAD$ in first buffer is moved to the second buffer of a or b with a probability of 0.75 and 0.25, respectively. According to the concentrations recorded in the second buffer, the substances are injected into the tuple space at that execution time step.

4.3 Flow Routing

In this subsection, we describe the routing of flow packets in the implementation of the proposed method.

4.3.1 Realization of Service Function Chaining

We first outline the implementation of SFC using NSH proposed by IETF [18].

NSH is a header added to flow packets to control the flow with an SFC request in the NFV system. Figure 10 depicts the format of NSH. NSH is composed of three fields: Base Header, Service Path Header, and Context Header. Base Header contains the basic information of NSH such as version, header length, and payload information. Service Path Header contains the identifier of a flow path and the state of SFC of the flow. Context Header contains the metadata. Figure 11 depicts the format of NSH Service Path Header, which stores information necessary to fulfill SFC requests. Service Path Header is composed of Service Path Identifier (SPI) and Service Index (SI). SPI uniquely identifies the flow path by having an ID of Service Function Path (SFP), which is described below. SI indicates location information on the path. Therefore, by using the combination of SPI and SI, the next VNF to be executed to the flow is determined.

Figure 12 depicts an example of the implementation design of SFC using NSH described in RFC 8300 [18]. Here, Service Function (SF) means a function to be executed on a flow packet. Service Function Forwarder (SFF) forwards the flow packet to the SF or SFF. Whereas an SFC request includes a chain of functions, an SFP represents a flow path with the detailed location information of a server where a required SF exists. This is used to forward the packet to the target server according to the SFC request. Service Classifier (SC) is located at the entrance of the NFV system, determines the SFP for the flow, and inserts the NSH into the packet.

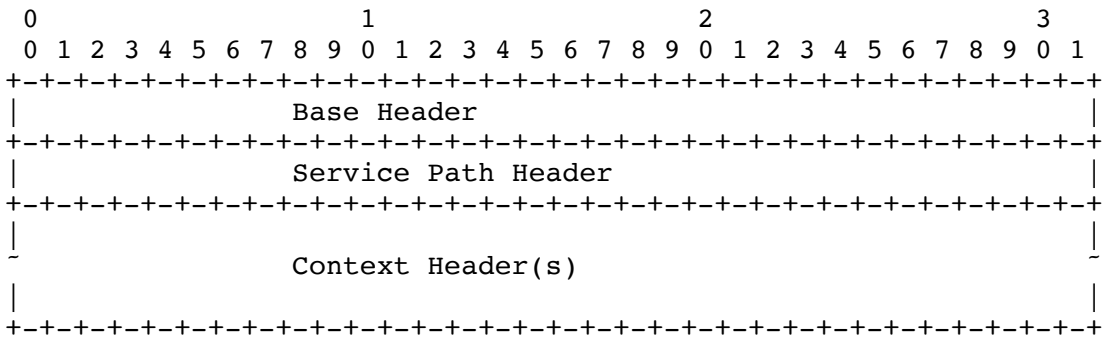


Figure 10: The format of NSH [18]

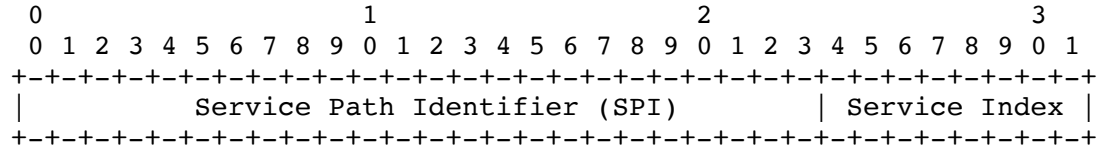


Figure 11: The format of NSH Service Path Header [18]

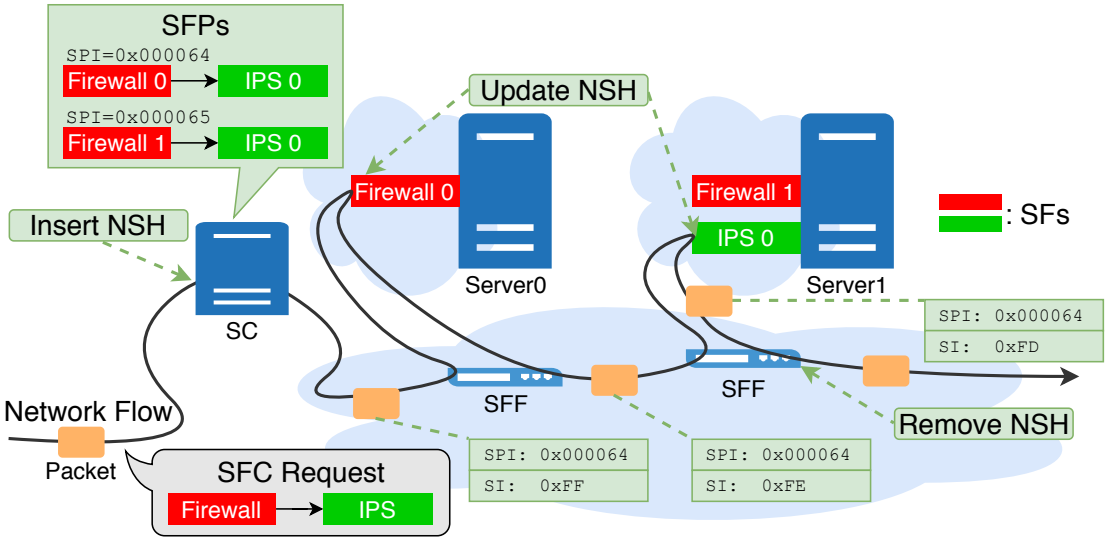


Figure 12: Example of implementation design of SFC using NSH

In Figure 12, an example where a packet of a flow having {Firewall → IPS} as an SFC request arrives at the NFV system is considered. Since the system has two firewalls, Firewall 0 and Firewall 1, and one IPS, IPS 0, two SFPs of {Firewall 0 → IPS 0} and {Firewall 1 → IPS 0} can be considered as SFPs for the flow. These candidates are managed by the SC. SC selects an SFP from the candidates for the flow, and inserts the NSH into the packet of the flow arriving at the system. When the SF processes a packet, the SF decrements the value of SI by one and forward the packet to the corresponding SFF. When all SFs to process have been processed, the SFF removes the NSH from the packet. Otherwise, the SFF forwards the packet to the next SF. In Figure 12, {Firewall 0 → IPS 0} is selected as SFP for the flow. SC inserts into the packet the NSH in which `0x000064` is set as the SPI value and `0xFF` is set as the SI value. The SFF sends the packet to Firewall 0, Firewall is applied to the packet, and Firewall 0 decrements the SI value of the packet by one. Then, the packet is forwarded by SFF to IPS 0, IPS is applied to the packet, and IPS 0 decrements the SI value of the packet by one. Finally, the SFF removes the NSH of the packet and output the packet from the system.

For SFs that do not support NSH, an SFC proxy [31] performs mediation processing. The SFC proxy is installed between the SF and the SFF, and removes the NSH to packets to be send to SF and inserts the NSH again to packets which SF has been applied to.

4.3.2 Stochastic Selection of Flow Path

In the proposed method, the path of a packet is determined stochastically for each packet in each tuple space. However, in NFV, the path of a packet is determined for each flow by centralized control. In addition, when the path is determined stochastically for each packet, the packet may loop, and an extra load may be applied to the system. Therefore, we propose stochastic selection of a flow path for each flow using the proposed method. After the SFP is selected stochastically in this way, the flow packets are forwarded using the NSH.

SC, SFF, and SF in Subsection 4.3.1 correspond to BRC, which is an SDN controller, SDN switch, and VNF, respectively. BRO aggregates the $GRAD$ information defined for each server's VNF and updates the SFP list and its selection probability. When a packet of a new flow arrives at the SDN switch, the BRC stochastically selects an SFP to be applied to the flow based on a BRO's list of SFPs corresponding to the SFC request and the selection probability. This selection is made in proportion to the $GRAD$ concentrations corresponding to the VNFs in the SFP. According to

the selected SFP, BRC configures SDN switches for packet forwarding and NSH insertion, update, and removing.

Figure 13 depicts an example of the stochastic selection of flow paths using the proposed method. In the figure, an example where a packet of a flow having $\{\text{Firewall} \rightarrow \text{IPS}\}$ as an SFC request arrives at the NFV system is considered. Since the system has two firewalls, Firewall 0 and Firewall 1, and one IPS, IPS 0, two SFPs of $\{\text{Firewall } 0 \rightarrow \text{IPS } 0\}$ and $\{\text{Firewall } 1 \rightarrow \text{IPS } 0\}$ can be considered as SFPs for the flow. In this case, the concentrations of $GRAD$ in the tuple spaces corresponding to Firewall 0 and Firewall 1 are 3,000 and 1,000, respectively.

Therefore, $\{\text{Firewall } 0 \rightarrow \text{IPS } 0\}$ or $\{\text{Firewall } 1 \rightarrow \text{IPS } 0\}$ is selected as SFP with a probability of 0.75 and 0.25, respectively. According to the selected SFP, configurations for SDN switches are executed, and packets of the flow packets are forwarded.

Since flow routing is performed for each flow by the above method, Reaction 24 is not defined in each tuple space.

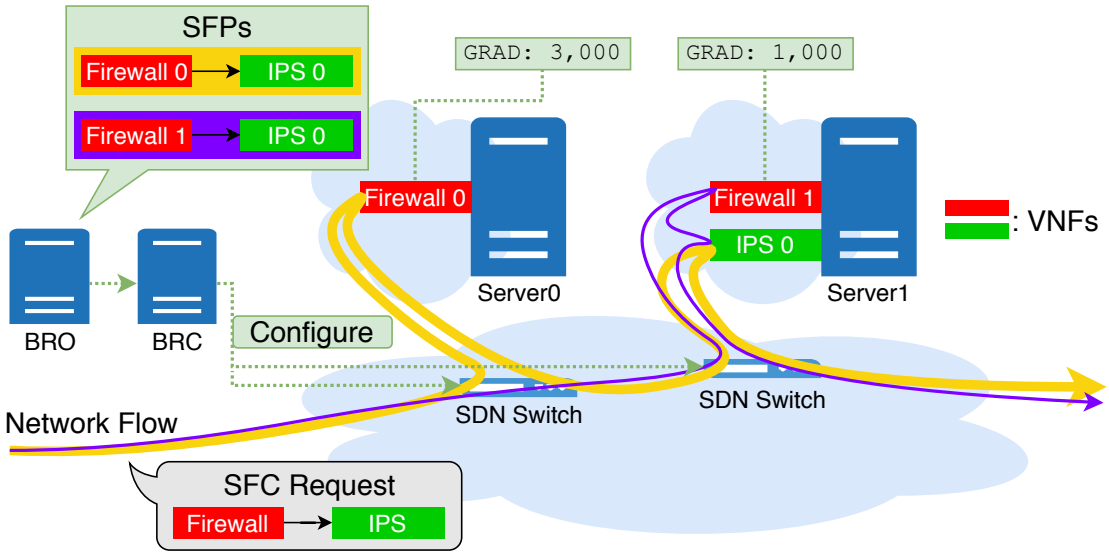


Figure 13: Stochastic determination of flow path with the proposed method

5 Experiment Setup

We describe the preparations and preliminary experimental results for experimental evaluations.

5.1 Application Scenario

In this thesis, we consider a scenario of applying VNFs according to SFC requests for video streaming service, as depicted in Figure 14. This environment consists of VNF servers, clients, and a media server. SFC requests consists of firewall and IPS. There are three VNF servers in the network, where one VNF is executed in each server. VNF 0 is for the firewall and VNF 1 and VNF 2 realize IPS.

The client first sends a packet of a request flow to the media server. When the packet enters the NFV environment, the system recognizes that a new flow arrives with an SFC request of $\{$Firewall $\rightarrow$ IPS$\}$. Since the system has one firewall, VNF 0, and two IPS, VNF 1 and VNF 2, two SFPs of $\{$VNF 0 $\rightarrow$ VNF 1$\}$ and $\{$VNF 0 $\rightarrow$ VNF 2$\}$ can be considered for the flow. The system selects one SFP from the candidates for the flow.

Next, the media server sends a packet of a response flow to the client. The flow is recognized as a new flow with an SFC request of $\{$Firewall$\}$. For the flow, one SFP of $\{$VNF 0$\}$ is considered as the SFP candidate.

For both directions, after an SFP request for a new flow is accepted to the system, the following packets of the flow can traverse between the client and the media server.

5.2 Environment of Experiments

Figure 15 depicts the network constructed for the experimental evaluations. The network consists of three physical servers: a client, a media server, and an NFV server, each connected by 1 Gbps ethernet. Table 2 shows the specifications of the three physical servers.

We constructed the NFV environment using OPNFV Fraser 6.2 in the NFV server. In Figure 15, Undercloud corresponds to Undercloud in OPNFV, Controller0 corresponds to a Controller node in OPNFV, and Compute0, Compute1, and Compute2 correspond to Compute nodes in OPNFV. For the virtual switches implemented by OVS, `br-admin` is used for the admin network, `br-tenant` is used for the tenant network, `br-storage` is used for the storage network, and `br-external` is used for the public network. Furthermore, an instance vnf0 realizing VNF 0

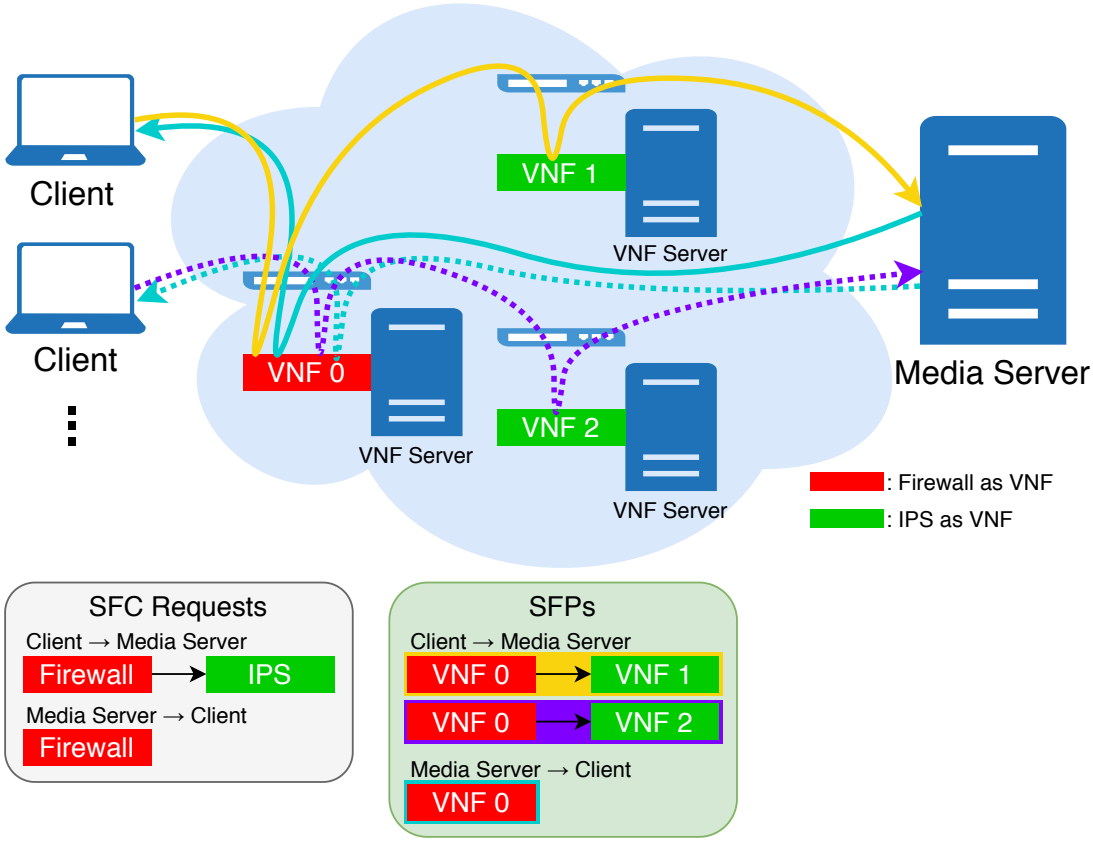


Figure 14: Application scenario

Table 2: Specifications of physical servers

	CPU	RAM	OS	Kernel
NFV Server	Intel®Xeon®E5-2690 0 2.90 GHz, 8 cores	32 GB	CentOS 7.5.1804	3.10.0-862
Client	Intel®Xeon®E3-1290 V2 3.70 GHz, 4 cores	16 GB	Ubuntu 18.04.1 LTS	4.15.0-43
Media Server	Intel®Xeon®E3-1290 V2 3.70 GHz, 4 cores	16 GB	Ubuntu 18.04.1 LTS	4.15.0-43

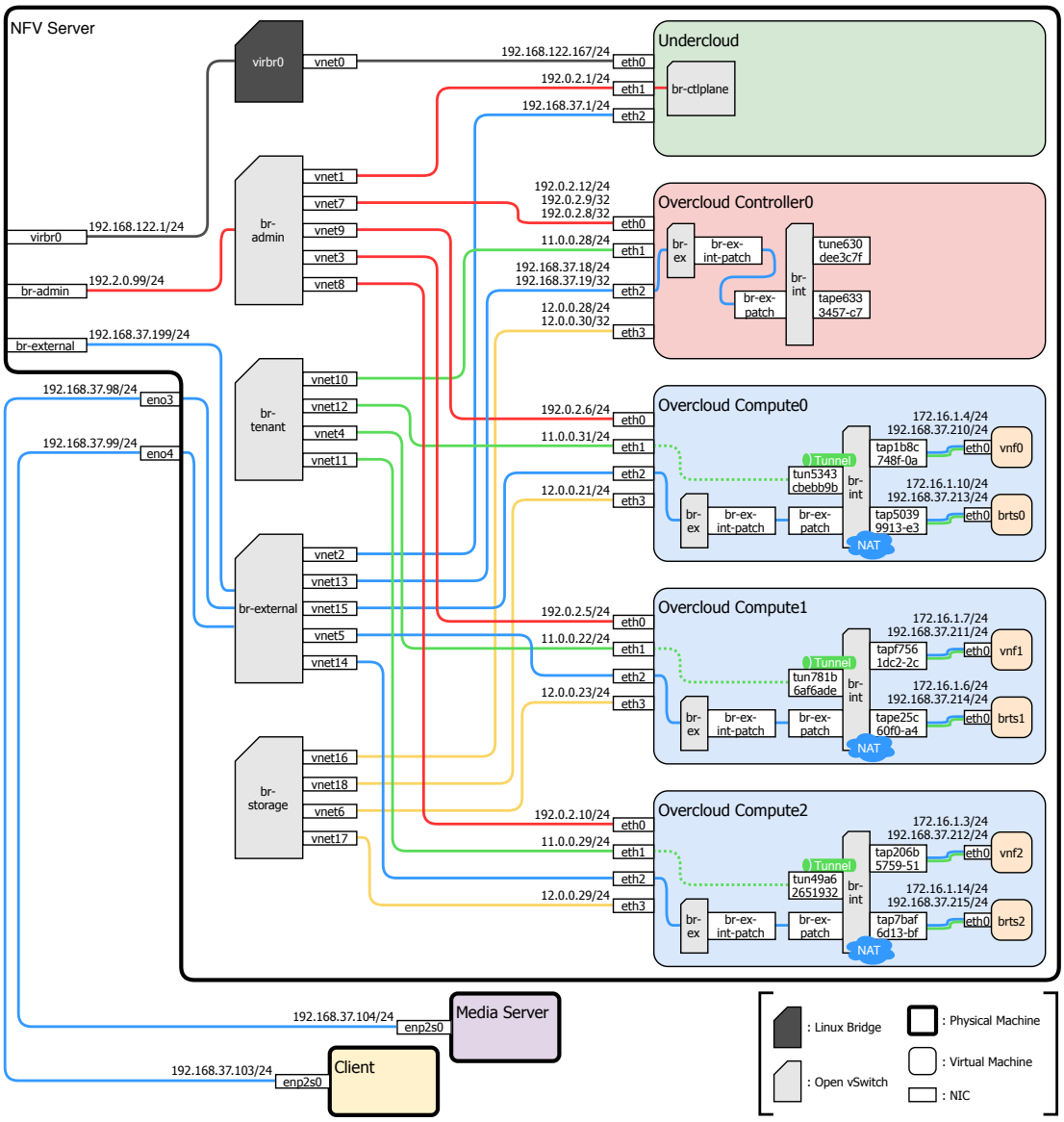


Figure 15: Experimental network configuration

exists in Compute0, an instance vnf1 realizing VNF 1 exists in Compute1, and an instance vnf2 realizing VNF 2 exists in Compute2. Each compute node also has instances brts0, brts1, and brts2 for implementing BRTS of the compute node as a VNF. BRTSs realized by brts0, brts1, and brts2 are called BRTS 0, BRTS 1, and BRTS 2, respectively. Table 3 shows the specifications of the virtual servers.

The service of media server is implemented by NGINX [32] and is provided by HTTP. The client and the media server are both connected to the public network.

5.3 VNF Implementations

The firewall is implemented by Netfilter [33]. Netfilter is a framework for packet filtering and packet processing such as NAT in Linux kernel 2.4 or later. In this thesis, IPS is implemented by combining Netfilter and Snort [34]. Snort is an open source network intrusion detection and prevention system provided by Cisco Systems, Inc.

Figure 16 depicts packet processing flow in Netfilter. In Netfilter, packet processing is performed by five chains of PREROUTING, FORWARD, INPUT, OUTPUT, and POSTROUTING. When a packet to which a firewall is to be applied is input from a network interface, after passing through PREROUTING, FORWARD, and POSTROUTING, the packet is output from a network interface. When a packet to which IPS is applied is input to a network interface, after passing through PREROUTING and INPUT, the packet is sent to Snort, which is implemented as a local process. The packet to which Snort processing has been applied passes through OUTPUT and POSTROUTING, and then is output from a network interface. In the setting of Netfilter and Snort, filtering and alert rules are not set, and only simple forwarding is performed.

5.4 Flow Emulation for Video Streaming Service

We use `httperf` [16] to generate an HTTP flow imitating a video streaming service. The specific flow setting is determined based on [35], describing a video streaming service such as YouTube. On YouTube, the playback bit rate of 1080p video is about 3.60 Mbps, and data transmission is performed at a higher speed than that. Two types of data transmission mechanisms are adopted on YouTube. In the latter half of one mechanism, 32 to 128 KB of data is acquired in one round trip, and the data is transmitted at an average of about 6.13 Mbps. We generate the flows imitating

Table 3: Specifications of virtual servers

	CPU	OS	RAM	Kernel
Undercloud	Intel®Xeon®E5-2690 0 2.90 GHz, 1 core	CentOS 7.4.1708	8 GB	3.10.0-693
Controller	Intel®Xeon®E5-2690 0 2.90 GHz, 1 core	CentOS 7.4.1708	16 GB	3.10.0-693
Computes	Intel®Xeon®E5-2690 0 2.90 GHz, 1 core	CentOS 7.4.1708	8 GB	3.10.0-693
Instances	Intel®Xeon®E312xx 2.90 GHz, 1 core	CentOS 7.7.1908	512 MB	3.10.0-862

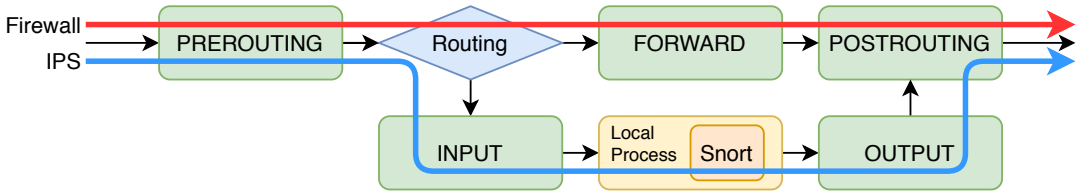


Figure 16: Packet processing flow in Netfilter

this by placing a 64 KB file on the media server and making 10 requests per connection from the client. With this setting, when one connection is generated per second, a flow of 5 Mbps will flow, and a flow of the 1080p video playback bit rate or higher will flow.

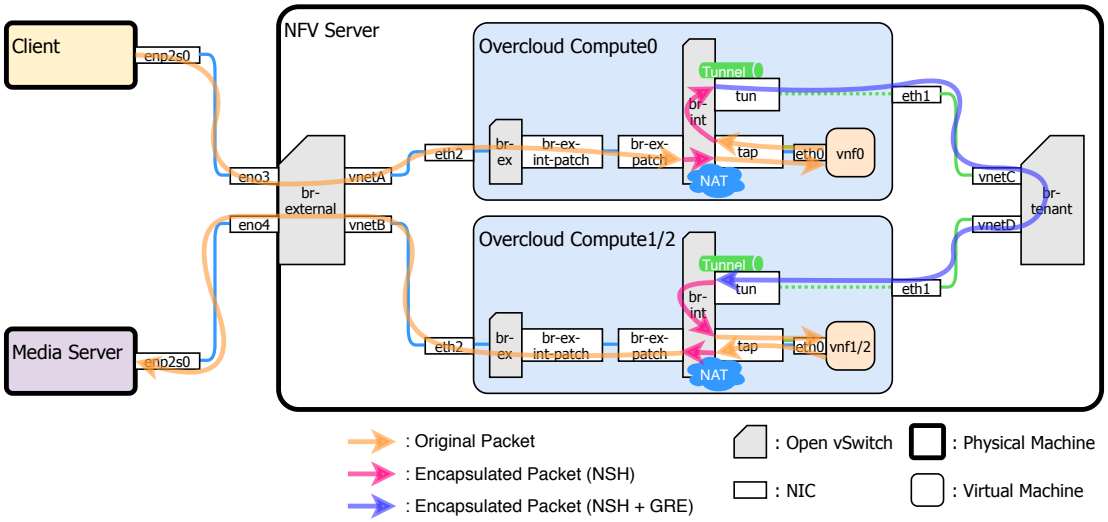
In the experiment, a flow from a plurality of clients is imitated by simultaneously generating a plurality of the above-described connections. Hereinafter, the number of connections generated per second set by `httperf` is referred to as *connection rate*.

5.5 Realization of Flow Routing by OpenFlow

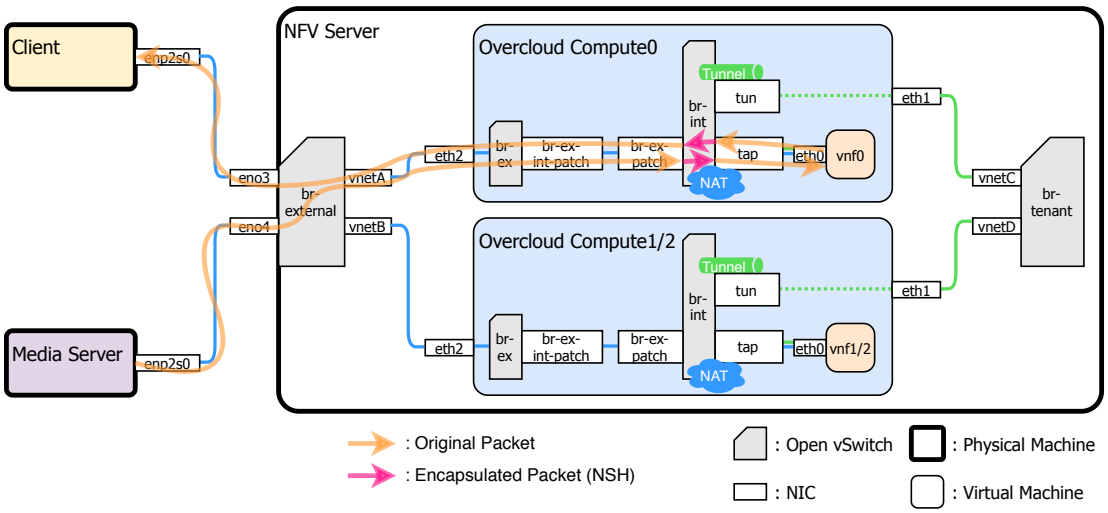
Figure 17 depicts an overview of the flow processing in this experiment. Since VNFs handled in this experiment do not support NSH by default, `br-int` plays the role of an SFC proxy. The colors of the arrows indicate the state of each packet of the flow, orange indicates the original packet, pink indicates the packet encapsulated by NSH, and purple indicates the packet encapsulated by NSH and GRE. Note that modification of the header is not shown in this figure.

The packets of the flow from the client to the media server are processed as follows.

1. A packet that enter the NFV server from the client are forwarded to `br-int` of Compute0.
2. `br-int` of Compute0 inserts proper NSH into the packet.
3. `br-int` of Compute0 removes the NSH of the packet and then forwards the packet to vnf0.
4. Firewall is applied to the packet at vnf0 and vnf0 forwards the packet to `br-int` of Compute0.
5. `br-int` of Compute0 inserts proper NSH into the packet.
6. `br-int` of Compute0 forwards packets to `br-int` of Compute1/2 by GRE tunneling.
7. `br-int` of Compute1/2 removes NSH and forwards packet to vnf1/2.
8. IPS is applied to the packet at vnf1/2 and vnf1/2 forwards the packet to `br-int` of Compute1/2.
9. `br-int` of Compute1/2 inserts proper NSH into the packet.
10. `br-int` of Compute1/2 removes NSH and then forwards the packet to the media server.



(a) Flow from Client to Media Server



(b) Flow from Media Server to Client

Figure 17: Flow routing on the experimental network

Besides, the packets of the flow from the media server to the client are processed as follows.

1. A packet that enter the NFV server from the client are forwarded to `br-int` of Compute0.
2. `br-int` of Compute0 inserts proper NSH into the packet.
3. `br-int` of Compute0 removes the NSH of the packet and then forwards the packet to vnf0.
4. Firewall is applied to the packet at vnf0 and vnf0 forwards the packet to `br-int` of Compute0.
5. `br-int` of Compute0 inserts proper NSH into the packet.
6. `br-int` of Compute0 removes NSH and then forwards the packet to the client.

These processes are realized using the OpenFlow protocol.

In the OpenFlow protocol, a flow table is defined in the SDN switch. Zero or more entries can be defined in one table, and each entry has the following elements.

Priority This indicates the priority in which the entry is referenced, with higher values giving priority.

Rule This indicates the condition to which the entry is applied, and the judgment is made based on the contents of the packet header, etc.

Action This indicates the processed performed when the entry is applied, such as header rewriting, transfer to other tables, and output from interface.

Statistics This records statistics such as the number of applied packets.

In the following description, flow tables are described as shown in Figure 18.

SPI and SI in Subsection 4.3.1 correspond to NSP and NSI, respectively. Then, $\{$VNF 0 $\rightarrow$ VNF 1$\}$, $\{$VNF 0 $\rightarrow$ VNF 2$\}$, and $\{$VNF 0$\}$ are made to correspond to NSP values 0, 1, and 2, respectively. Figures 19 to 24 show the entries in the flow table set for each switch. When the entries shown here are not applied, the default entries are applied. The forwarding among the tables is not represented as an action, but is represented by an arrow in these figures. Here, `TunID` is the ID used for default tunneling in the OpenStack environment, and `SrcPort` is the TCP port

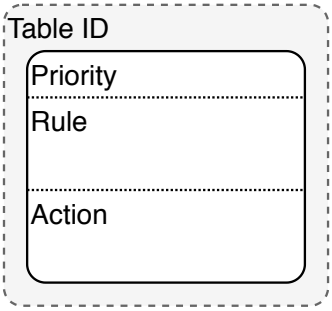


Figure 18: Example of flow table

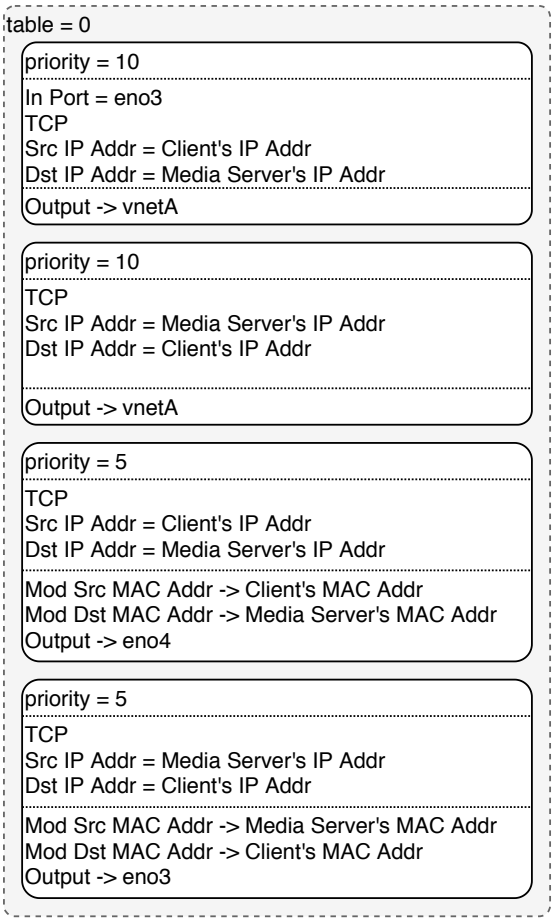


Figure 19: Flow entries on `br-external` of the NFV Server

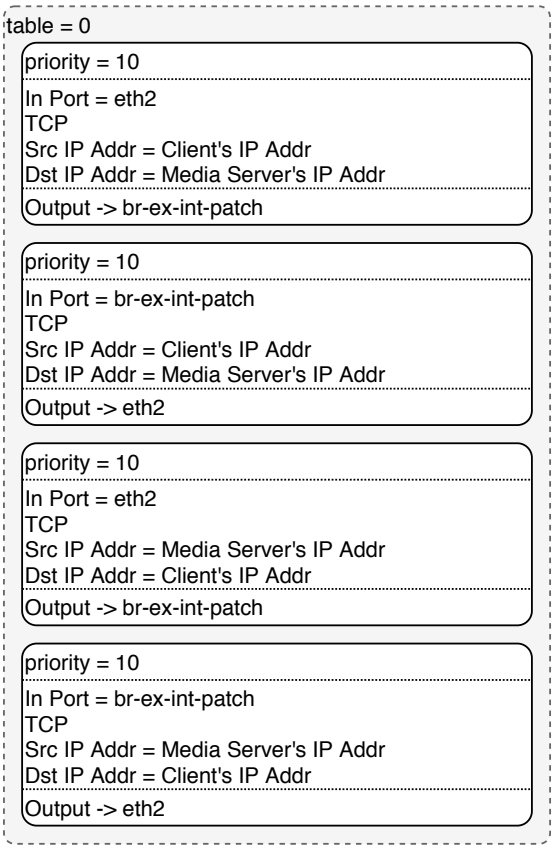


Figure 20: Flow entries on `br-ex`s of Compute0, Compute1, and Compute2

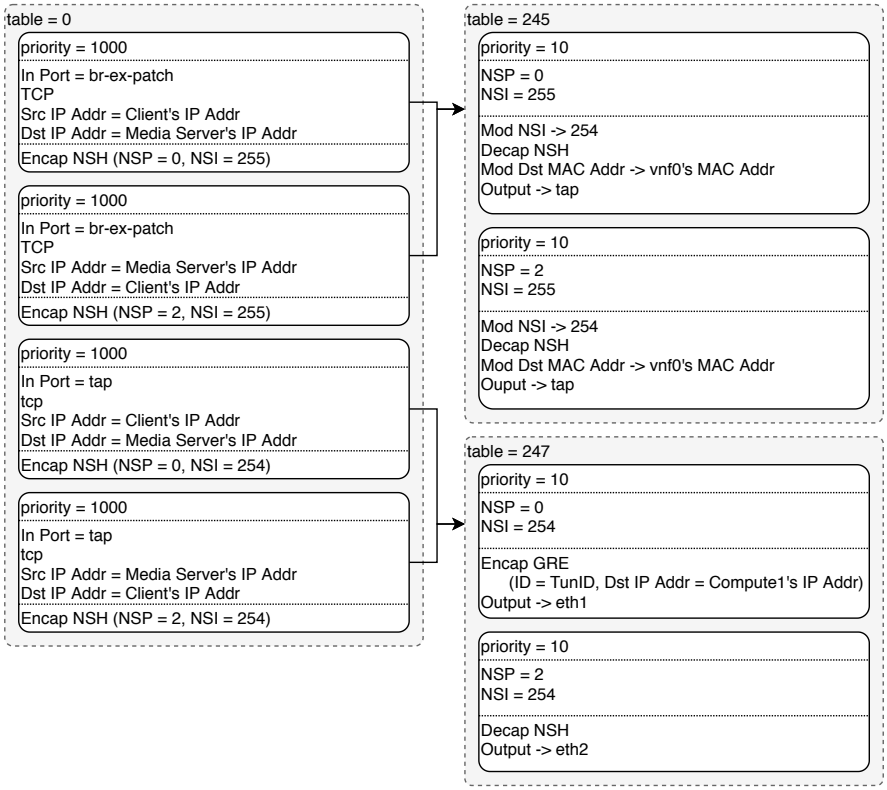


Figure 21: Flow entries on `br-int` of Compute0 (static routing)

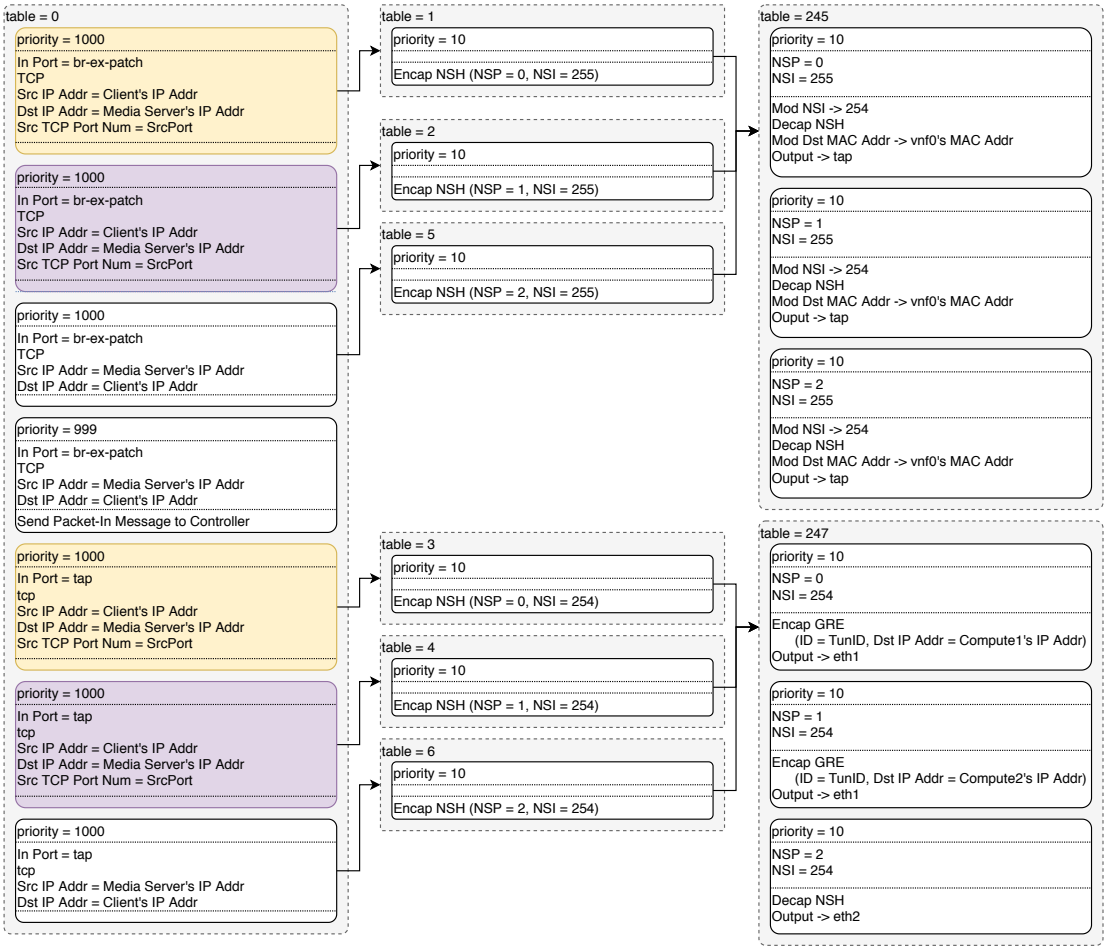


Figure 22: Flow entries on `br-int` of Compute0 (dynamic routing)

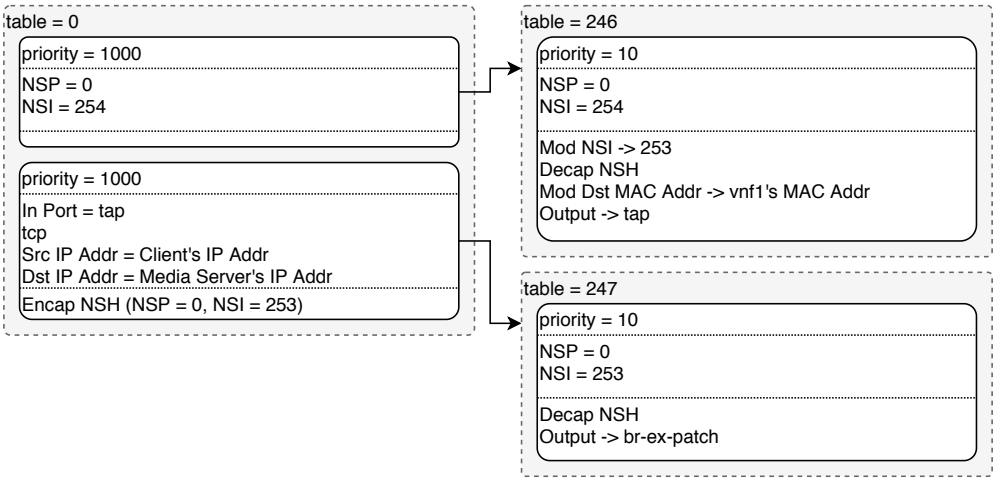


Figure 23: Flow entries on `br-int` of Compute1

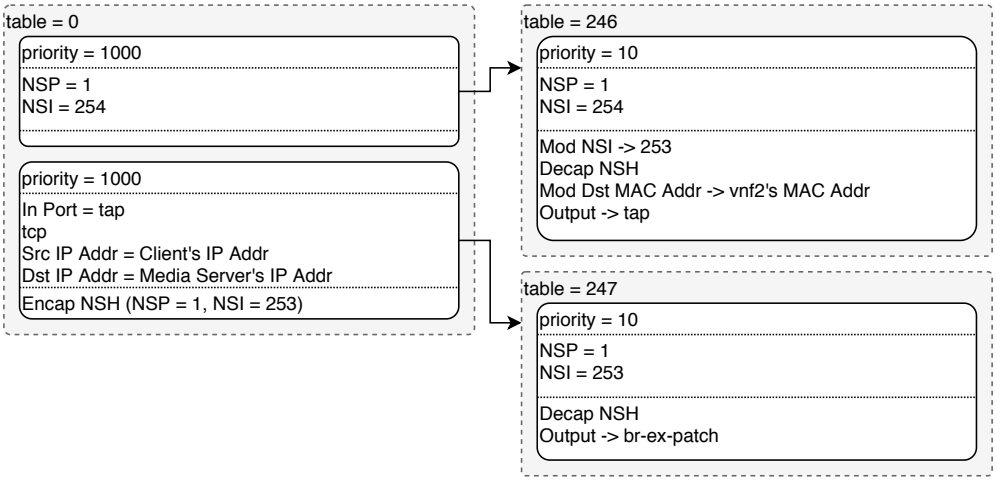


Figure 24: Flow entries on `br-int` of Compute2

number of the client uses. The IP address of the compute node used for tunneling by GRE is that of the tenant network.

Of the entries shown, entries other than those shown in Figures 21 and 22 are configured statically for all experiments. Figure 21 shows entries that are statically configured when the SFP $\{$VNF 0 $\rightarrow$ VNF 1$\}$ is statically selected for all flow packets from the client to the media server without performing path control. On the other hand, the entries shown in Figure 22 are entries that are configured when two candidates of SFPs, $\{$VNF 0 $\rightarrow$ VNF 1$\}$ and $\{$VNF 0 $\rightarrow$ VNF 2$\}$, are stochastically selected for a packet of a flow from the client to the media server by performing path control. The entries shown in yellow and purple in Figure 22 are dynamically added by BRC when the SFP is selected by BRC each time a new flow comes in. When a new flow from the client to the media server enters the system, a Packet-In message is sent to the controller from the `br-int` of Compute0. BRC selects SFP, the yellow entry is added when $\{$VNF 0 $\rightarrow$ VNF 1$\}$ is selected, and the purple entry is added when $\{$VNF 0 $\rightarrow$ VNF 2$\}$ is selected.

5.6 Preliminary Experiment

5.6.1 Correlation of the Length of Execution Time Step and CPU Usage of BRTS

As described in Subsection 4.2.2, this implementation sets the length of execution time step as the initial value of τ in the τ-leaping method. Therefore, it is expected that the amount of calculation will vary with the length of execution time step. BRTS is implemented as a VNF, and its CPU usage is required to be as low as possible. Hence, we determine the optimal execution time step experimentally.

In this experiment, we set various values as the initial value of τ and observe CPU usage of BRTS. We execute BRTS on the instance without any connection with other components. In the tuple space, Reactions (15)–(18) are defined, and each reaction rate constant is 1.0. In addition, the initial values of the concentrations are set to 1,000 for VNF, 1,000 for $RSRC$, and 0 for the others. Then, we perform experiments executing BRTS for 100 seconds with $\lambda = 400$. When the initial value of τ is set to 100 ms, 500 ms, 1 s, 5 s, and 10 s, we observe the CPU usage of the instance executing BRTS during the experiment using the `top` command every second, and set the average value as a result. Each experiment is performed 5 times, and the results are shown in Figure 25. The figure shows the averages of the results, the minimum results, and the maximum

results for each initial value of τ.

From this result, it is understood that the CPU usage rate of the instance executing BRTS changes depending on the set value of the initial value of τ. When the initial value of τ is small, the CPU usage increases because the number of executions of the τ-leaping method increases simply. Additionally, when the initial value of τ is big, the CPU usage also increases because the number of recursive executions of the τ-leaping method increases, since the number of reactions increases and the concentration value becomes a large negative value. In below experiments, 1 second, which has the lowest average CPU usage, is set as the length of each execution time step.

5.6.2 Correlation of Flow Rate and CPU Usage of VNF

We determine the constant values for each type of VNF, which are used in Equations (42) and (43). First, L_{max}, which is the maximum value of the CPU resource allocated to the VNF, is set to 90% in order to be able to respond to environmental fluctuations such as rapid fluctuations in flow rates and reductions in server resources.

It is known that there is a linear relationship between the flow rate processed by VNF and the CPU usage of VNF [36, 37]. Hence, we determine F_{max} and L_{min} experimentally, where F_{max} bps is the maximum flow rate that can be processed when maximum CPU resources are allocated to the VNF and $L_{min}\%$ is the minimum value of CPU resources allocated to the VNF. The connections are disconnected in 10 seconds. We execute `httperf` for 10 seconds with the settings described in Subsection 5.4 and change the connection rate from 0 to 50, in 1 increments.

In this experiment, only one VNF is applied, and in VNF, firewall or IPS is realized according to the experiment. In order to carry out the experiment, flow routing through only one VNF is realized by configuring for switches on the path. According to the scenario described in Subsection 5.1, when experimenting with firewall, we set up so that packets in both directions between the client and the media server are forwarded to VNF, and when experimenting with IPS, only packets going from client to server are forwarded to VNF. Here, operations related to NSH are omitted, and switches perform only simple forwarding. No filtering or alert rules are set for VNFs.

We observe the CPU usage of the instance realizing VNF during the experiment using the `top` command every second. The second to ninth CPU usage values are extracted from the result, and the averaged value is used as the CPU usage of VNF. Also, in order to observe whether the request is being processed properly, the number of executed connections and the number of responses are

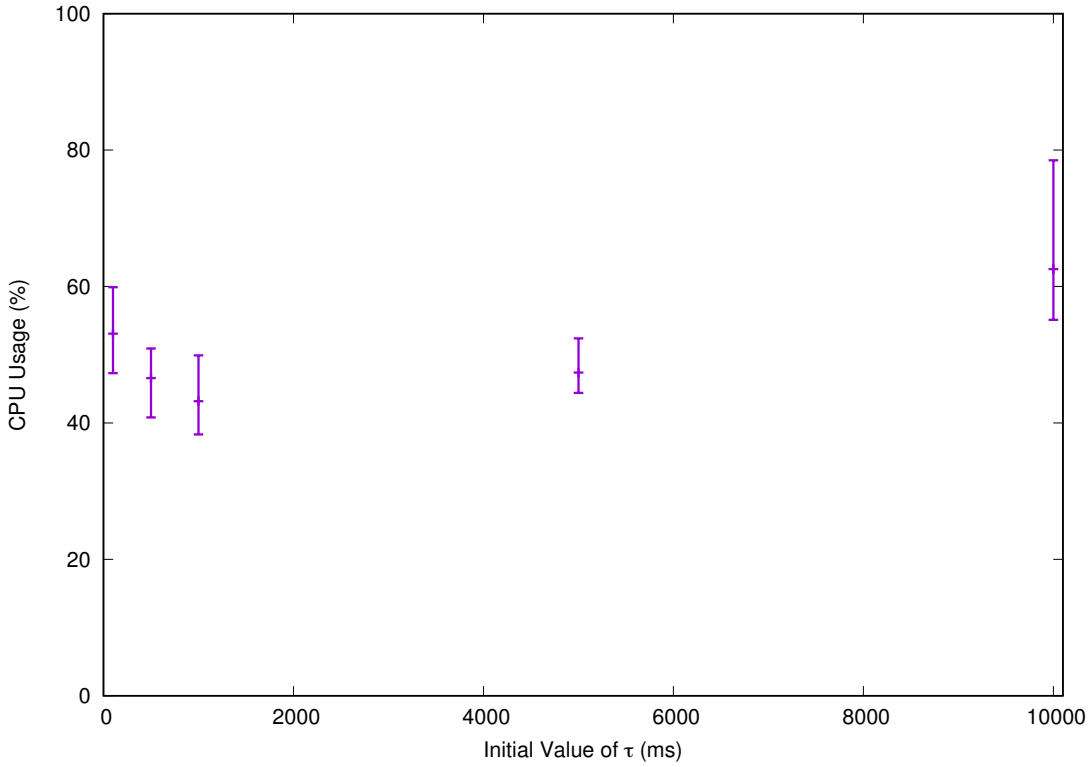


Figure 25: Effect of the initial value of τ on CPU usage of BRTS

obtained from the execution result of `httperf`. The experiment is performed 10 times for each connection rate.

Figure 26 shows the relationships between the connection rate and the number of responses per second, and Figure 27 shows the relationships between the number of responses per second and CPU usage of VNF. For Figure 26, the line for the ideal case with 10 responses per connection is also plotted. For Figure 27, the result of the linear approximation described later is also plotted. From Figure 26, when the connection rate increases, the number of responses per second corresponding to the connection rate cannot be obtained for both VNFs. From Figures 26 and 27, this problem has not occurred at least until the connection rate at which CPU usage of VNF exceeds 80%. Therefore, it is expected that the CPU becomes a bottleneck and the number of responses per second drops.

Next, values until the CPU is overloaded are extracted, and linear approximation is performed. For each VNF, extract the result when the connection rate is from 0 to 15, and perform the linear approximation by the least squares method. The results are shown in Equations (46)-(47) . Here, y_{fw} and y_{ips} indicate the CPU usage (%) and x'indicates the number of responses per second. The significant figure is two digits. The coefficient of determination of each equation is also shown.

$$y_{fw} = 0.53x' + 10, R^2 = 0.96 \tag{44}$$

$$y_{ips} = 0.50x' + 10, R^2 = 0.95 \tag{45}$$

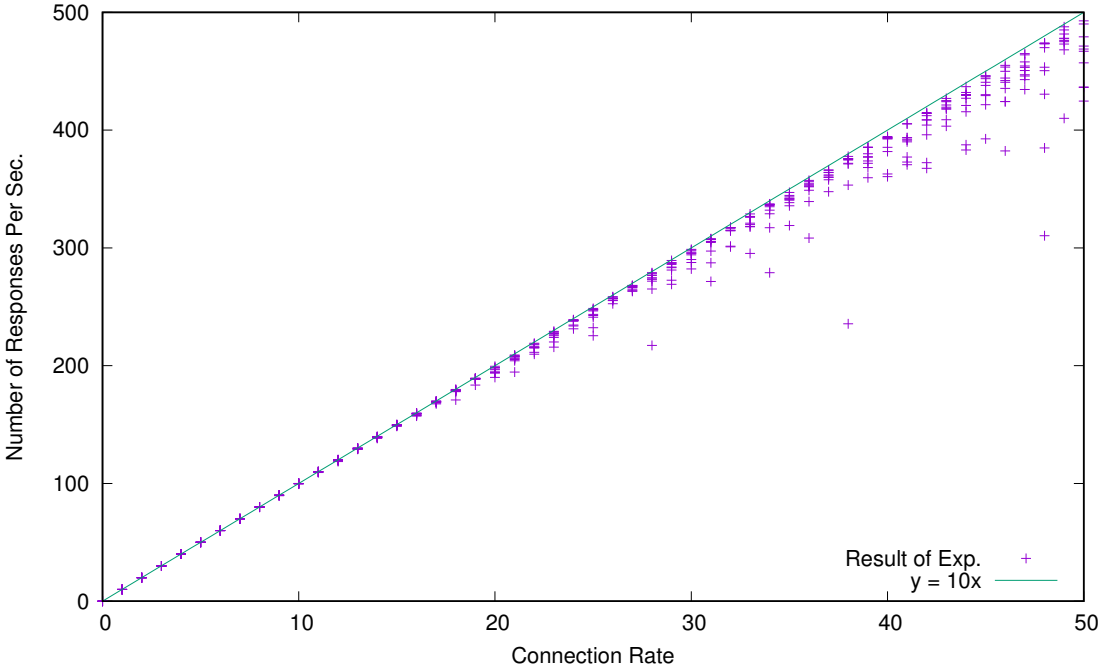
As shown in Figure 27, the CPU usage varies, so the margin α is set to allow sufficient CPU resources to be allocated.

$$y_{fw} = 0.53x' + 10 + \alpha \tag{46}$$

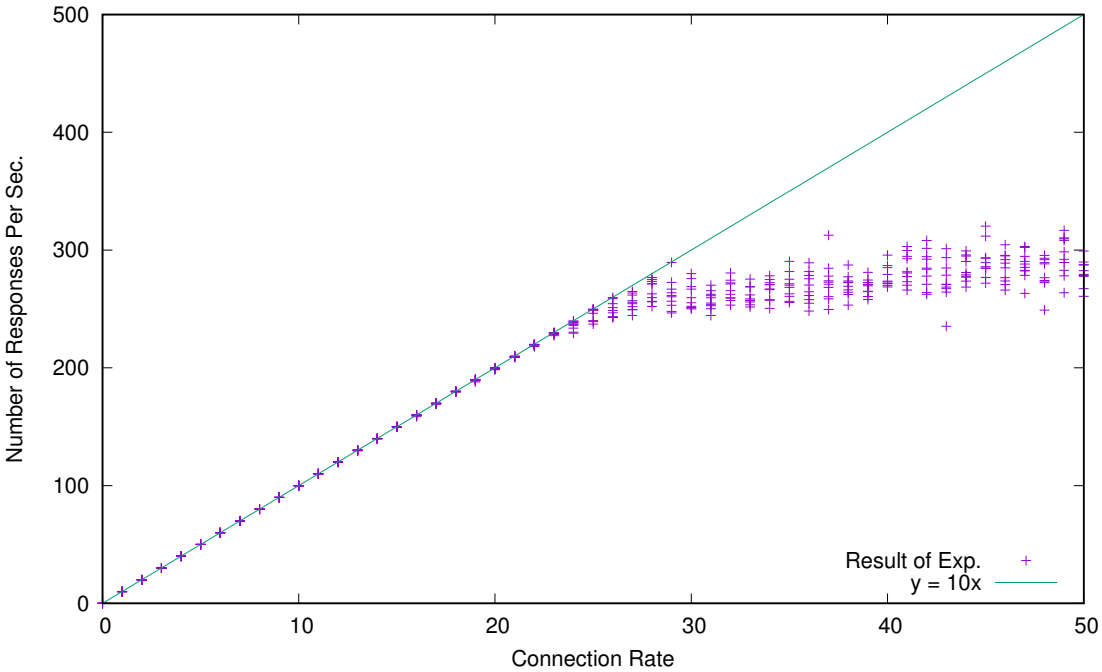
$$y_{ips} = 0.50x' + 10 + \alpha \tag{47}$$

Figure 27 also plots each approximate line with $\alpha = 0$ and $\alpha = 12$ for both equations. Based on this result, when the margin α is set to 12, the CPU usage is less than the expected value for all attempts, so this value is set.

Since Equations (46) and (47) represent the number of responses per second, they are converted into equations about to the flow rate. Confirming the total size of packets processed by VNF when the response is returned correctly in the setting of this experiment from the OVS log, these are 707,920 bytes for firewall and 17,618 bytes for IPS. Therefore, Equations (48) and (49)

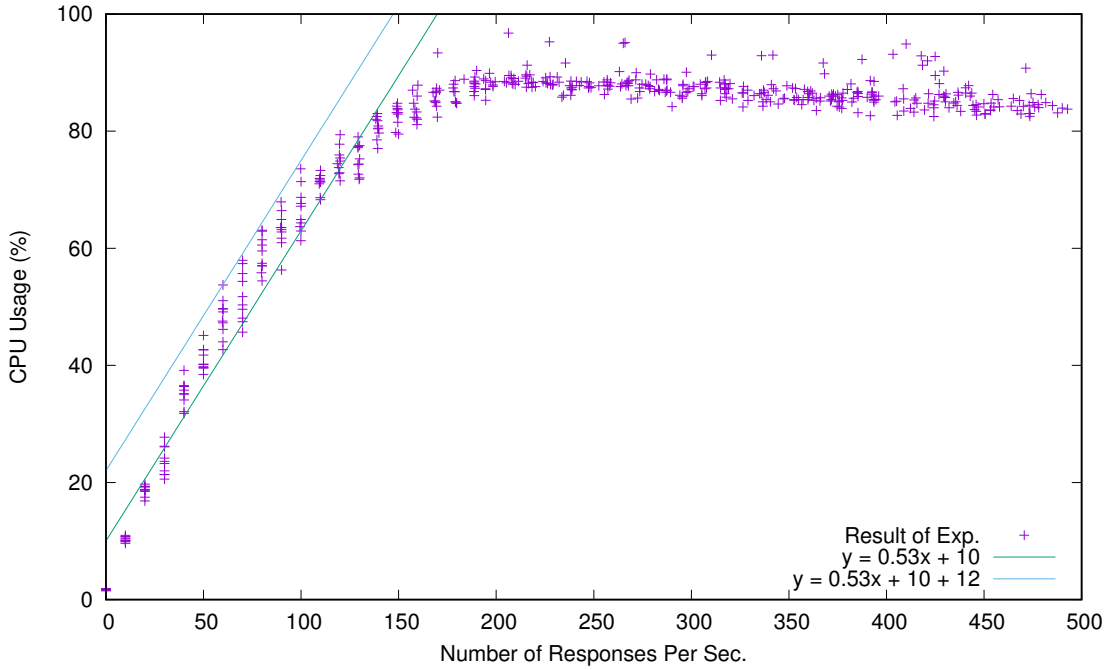


(a) Firewall

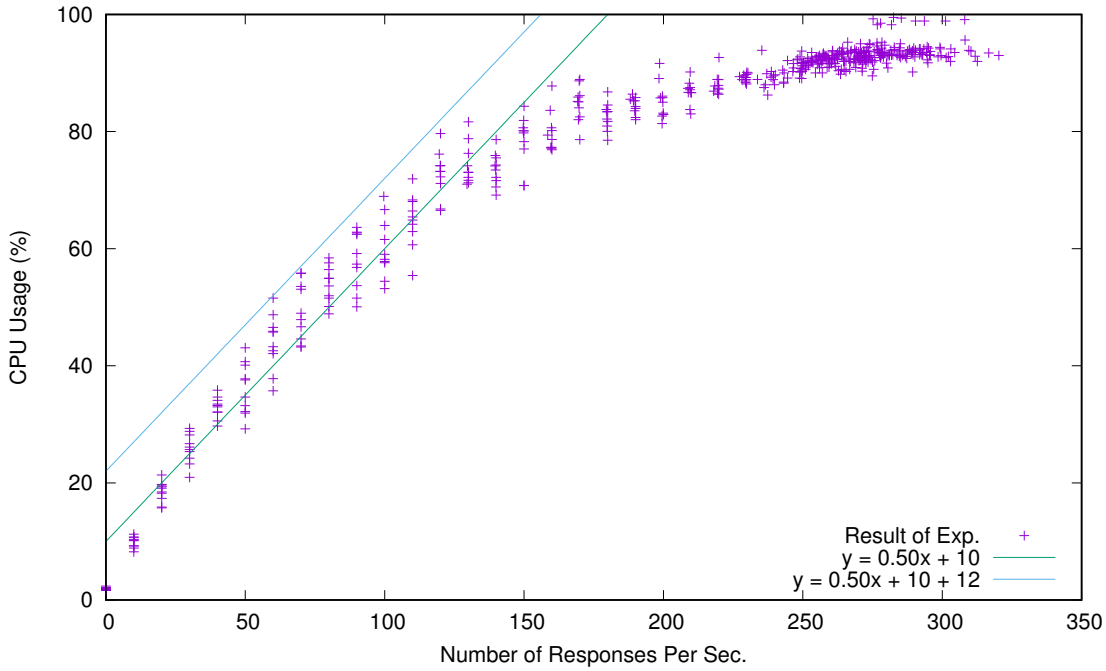


(b) IPS

Figure 26: Relationships between the connection and the number of responses per second



(a) Firewall



(b) IPS

Figure 27: Relationships between the number of responses per second and CPU usage of VNF

for calculating the amount of CPU resource to be allocated to the VNF from the flow rate can be defined. Here, x indicates a flow rate (Mbps).

$$\begin{aligned} y_{fw} &= 0.53x' + 10 + 12 \\ &= \frac{0.53 \times 1024 \times 1024 \times 10}{707920 \times 8} x + 10 + 12 \\ &\approx 0.98x + 10 + 12 \end{aligned} \tag{48}$$

$$\begin{aligned} y_{ips} &= 0.50x' + 10 + 30 \\ &= \frac{0.50 \times 1024 \times 1024 \times 10}{17618 \times 8} x + 10 + 12 \\ &\approx 150x + 10 + 12 \end{aligned} \tag{49}$$

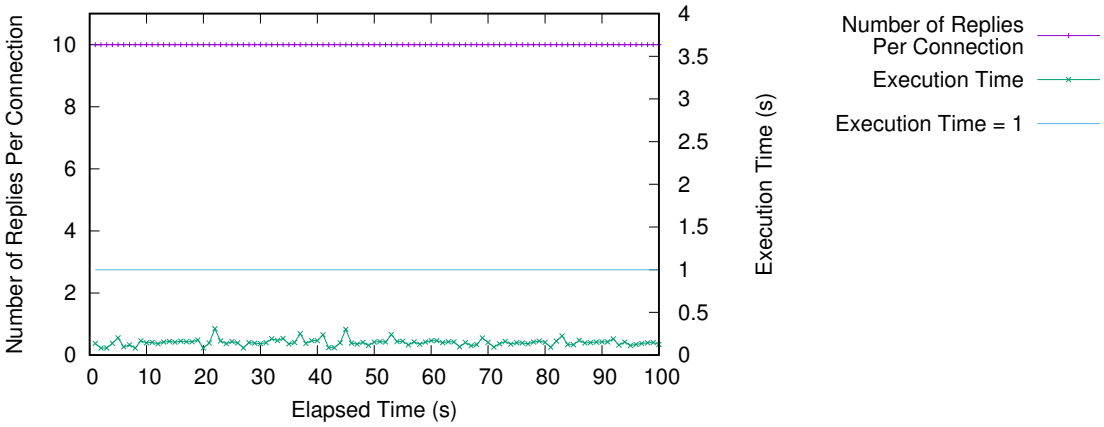
From these equations, the minimum amount of CPU resources allocated to VNFs in both the firewall and the IPS L_{min} is 22%. Further, since the maximum value of the CPU resource amount allocated to the VNF is set to 90%, it can be calculated as $F_{max,fw} \approx 69$ Mbps and $F_{max,ips} \approx 0.45$ Mbps. These constants are used in the following experiments. Note that these constants can be changed online depending on the status of the server, etc.

5.6.3 Execution Results with Insufficient CPU Resources

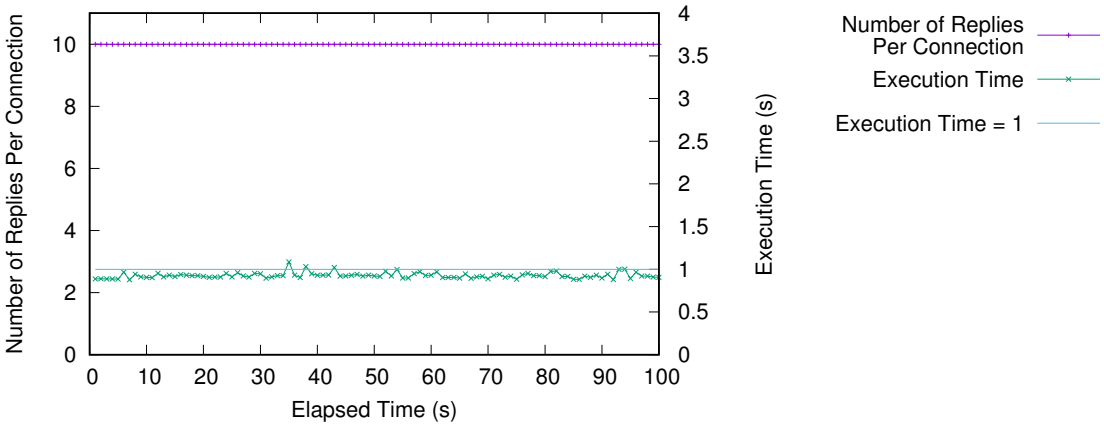
When an insufficient value is selected as the CPU resources to be allocated to the VNFs in the path, an experiment is performed to determine what the execution results of `httperf` will be like.

In the scenario shown in Subsection 5.1, we consider the flow is routed statically without using VNF 2. For each VNF, we use `cpulimit` with the settings shown in Table 4 and allocate 100% as the CPU resource allocation amount and 22%, which is the lowest value that could be allocated in this experiment. Using the flow described in the description in Subsection 5.4, we execute `httperf` every 1 second for the connection rate of 1, 5, and 10, respectively. The connections are set to be disconnected in 1 second. The number of replies per connection and the execution time for each connection rate obtained from the execution result of `httperf` are shown in Figures 28 to 30.

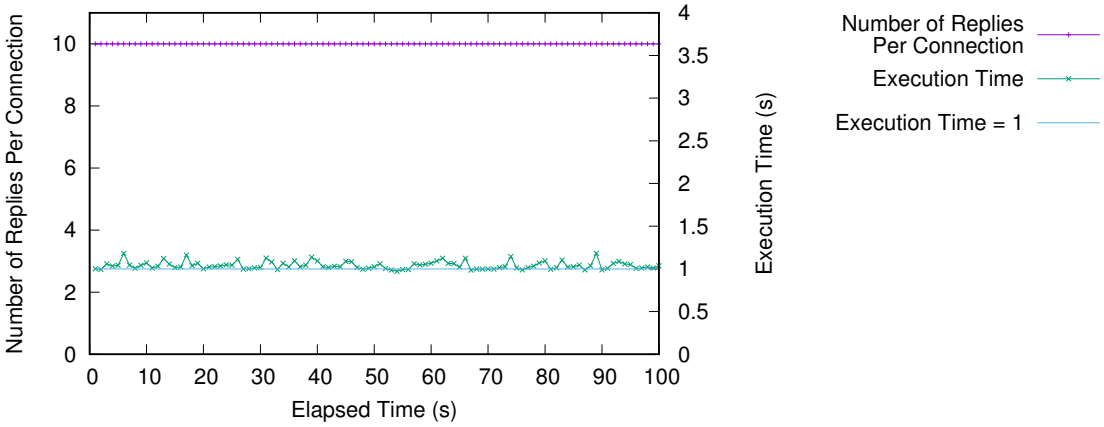
First, in the result of Figure 28, the numbers of replies per connection are always 10 for all connection rates. This indicates that all 10 requests in one connection are correctly processed. The execution time is all within 1.2 seconds. In `httperf`, a connection is generated every number



(a) Connection rate = 1

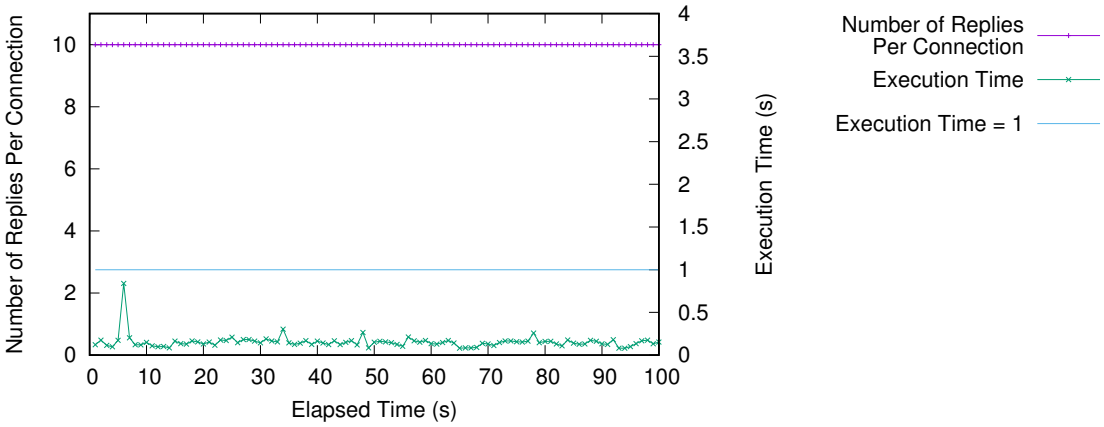


(b) Connection rate = 5

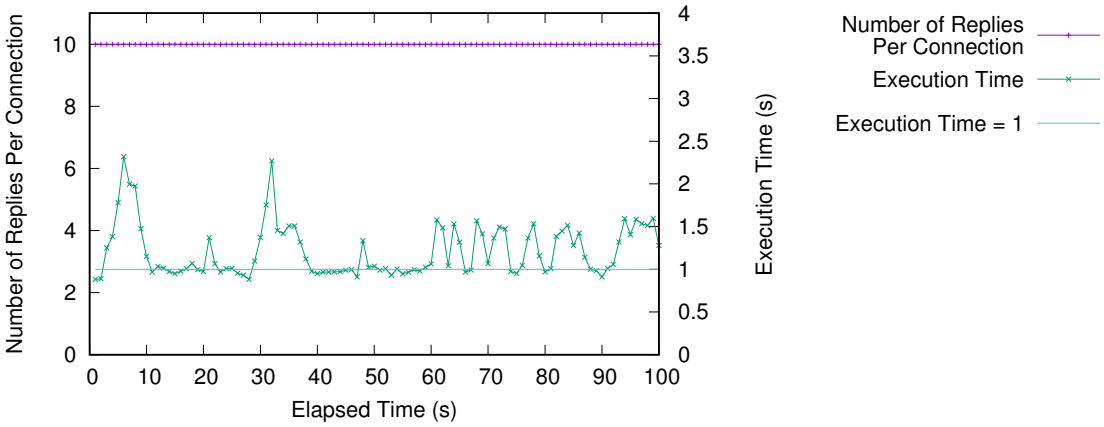


(c) Connection rate = 10

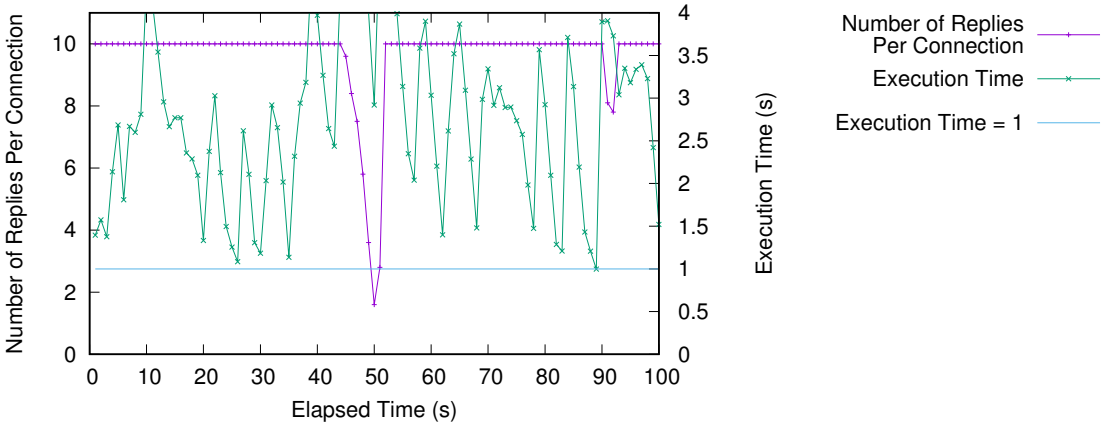
Figure 28: `httperf` performance (no limitations)



(a) Connection rate = 1

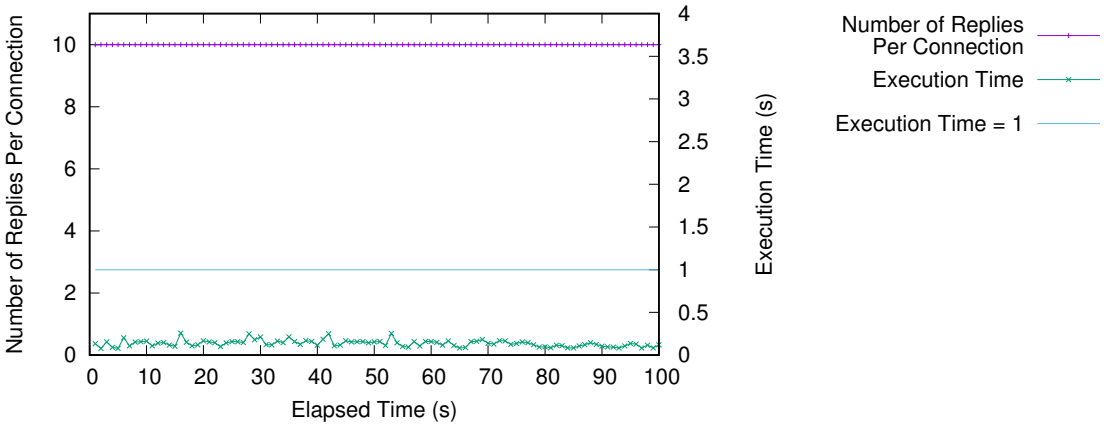


(b) Connection rate = 5

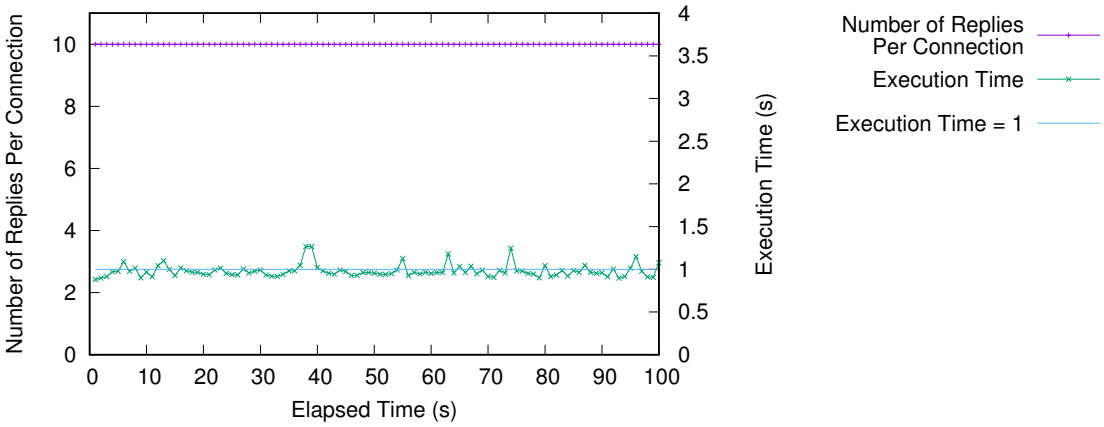


(c) Connection rate = 10

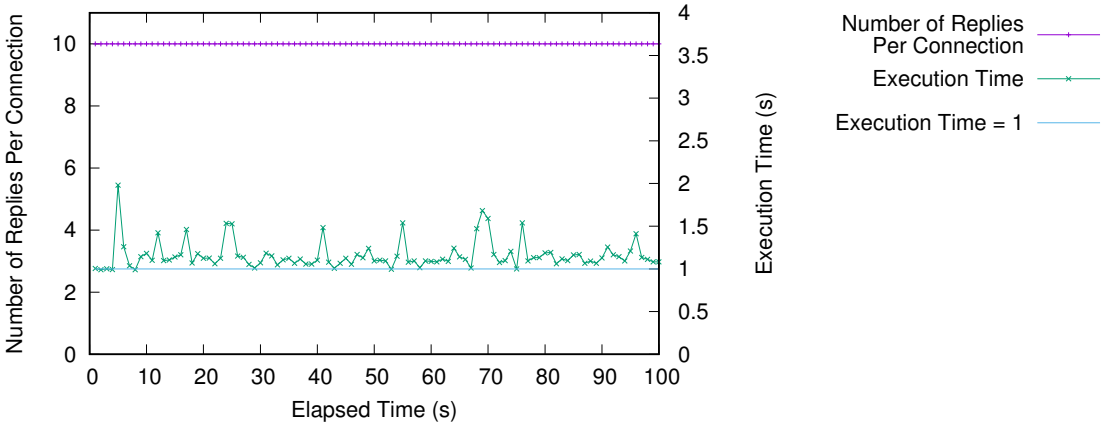
Figure 29: `httperf` performance (set limitation to firewall)



(a) Connection rate = 1



(b) Connection rate = 5



(c) Connection rate = 10

Figure 30: `httperf` performance (set limitation to IPS)

of seconds obtained by dividing 1 second by the connection rate. Therefore, when the connection rate is low, the execution time may be significantly less than 1 second. Conversely, when the connection rate is high, the generation of the connection is performed at the end of the execution time, so that even when the processing is performed correctly, the execution time may exceed 1 second.

Next, looking at the result of Figure 29, where CPU resources allocated to VNF 0 that realizes firewall is set to 22%, when the connection rate is 10, the number of replies per connection is sometimes less than 10. This indicates that the processing for the request is not performed correctly as a result of frequent packet loss or timeout of the connection due to processing delay. In addition, when the connection rate is 5 or 10, the execution time of `httperf` may greatly exceed every 1 second, and the execution time of `httperf` tends to be longer when the connection rate is higher. This also indicates that the processing for the request is not performed correctly.

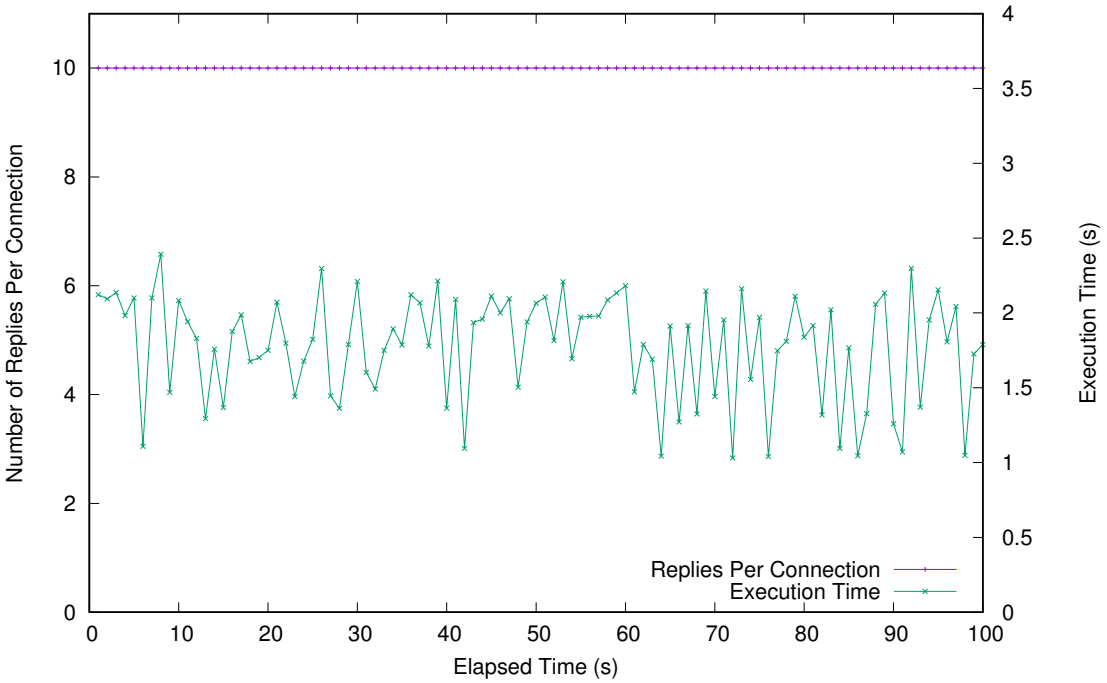
Finally, looking at the result of Figure 30, where CPU resources allocated to VNF 1 that realizes IPS is set to 22%, the numbers of replies per connection are always 10 for all connection rates. However, when the connection rate is 10, the execution time of `httperf` may greatly exceed 1 second.

From the above results, when sufficient CPU resources are not allocated to the VNF, the number of replies per connection will be less than 10, or the execution time will take much more than 1 second. In the following experiments, we confirm that these problems do not occur by appropriate allocation of CPU resources to VNFs using the proposed method.

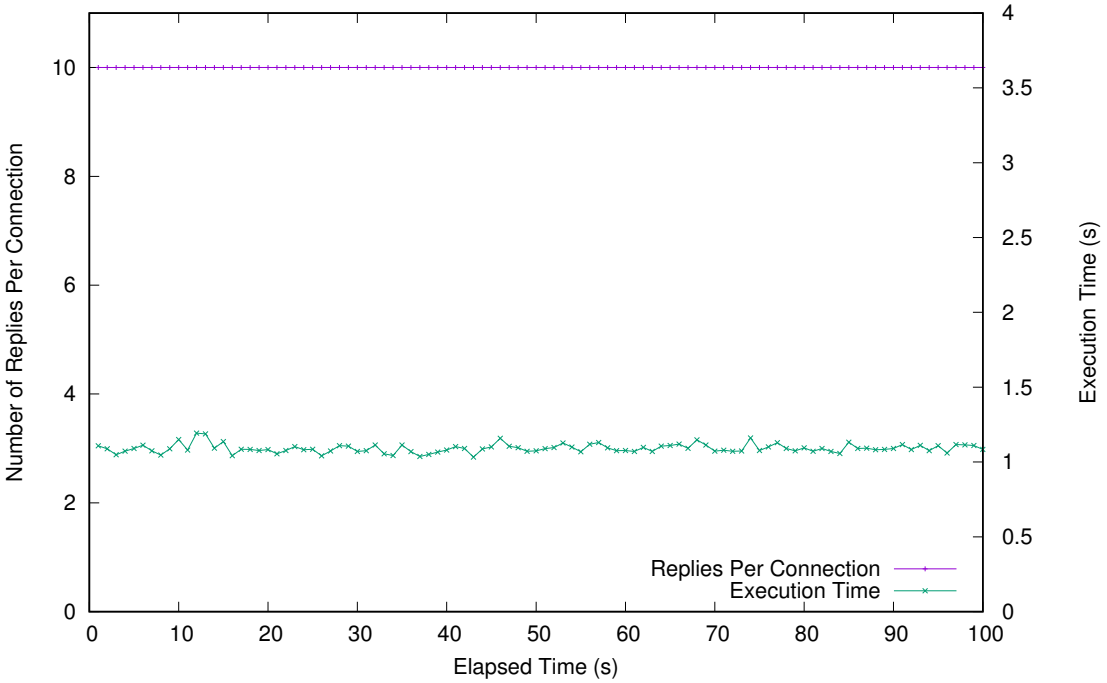
5.6.4 Waiting Time Between Sending a Flow-Mod Message and a Packet-Out Message in BRC

As described in Subsection 4.2.4, in implementation of BRC, we insert 20 ms wait between sending a Flow-Mod message and a Packet-Out message. We describe the detail from the results of the following experiment.

In the scenario shown in Subsection 5.1, we consider a flow is routed dynamically, but the path using VNF 2 is not selected. Using the flow described in the description in Subsection 5.4, we execute `httperf` every 1 second with the connection rate of 10. The amounts of CPU resources allocated to the VNFs in the path is 100%. Figure 31(a) shows the number of replies per connection and the execution time from the execution results of `httperf`. This result indicates that the



(a) `httperf` performance without wait insertion



(b) `httperf` performance with 20 ms wait insertion

Figure 31: Effect of wait insertion between Flow-Mod and Packet-Out messages

execution time is sometimes as long as 2 seconds or more even though sufficient CPU resources are allocated to the VNFs in the path. Investigation revealed that this is due to the delay in modifying the flow entry in switch.

In the environment of this experiment, when a new flow enters the system, a Packet-In message is sent from `br-int` of Compute0 to the BRC. The BRC sends a Flow-Mod message for setting an appropriate entry to the switch, and then sends the packet that triggered the Packet-In as a Packet-Out directly to the VNF. The VNF processes and sends the packet back to switch. Here, when the packet is sent back before the application of the setting of the new entry is completed in the switch, there is no entry to be applied at that time, and appropriate flow routing is not performed. The existence of a packet in which there is no entry to be applied is confirmed by experiments.

Therefore, we insert a certain wait between sending a Flow-Mod message and a Packet-Out message. The results of a same experiment performed when the wait time is set to 20 ms are shown in Figure 31(b). At this time, the execution time does not become extremely long, so this setting is applied to the following experiments.

Table 4: CPU usage of VNFs of the experiments in Subsection 5.6.3

Description	VNF 0 (Firewall)	VNF 1 (IPS)	result
No Limitations	100%	100%	Figure 28
Set Limitation to Firewall	22%	100%	Figure 29
Set Limitation to IPS	100%	22%	Figure 30

6 Evaluation Results and Discussions

In this section, we describe the details of the experimental evaluations, results, and discussions.

6.1 Scenario 1: Resource Allocation

6.1.1 Evaluation Scenario and Parameter Settings

We consider the scenario shown in Subsection 5.1 except that only VNF 0 and VNF 1 are used while VNF 2 is not used. Therefore, flows are statically routed between the client and the media server. The proposed system performs only dynamic resource allocation and does not conduct diffusion-related reactions.

Table 5 summarizes the settings of experiments. In Exp. A, Exp. B, and Exp. C, `httperf` is executed at every second for realizing a constant connection rate. These experiments confirm that the concentrations of chemical substances converge to an appropriate value for flow rate and that CPU resources are allocated to VNFs accordingly. We also assess that the flows are properly processed. In Exp. D, the connection rate changes dynamically at every 50 seconds for confirming the ability of the proposed method to follow fluctuations.

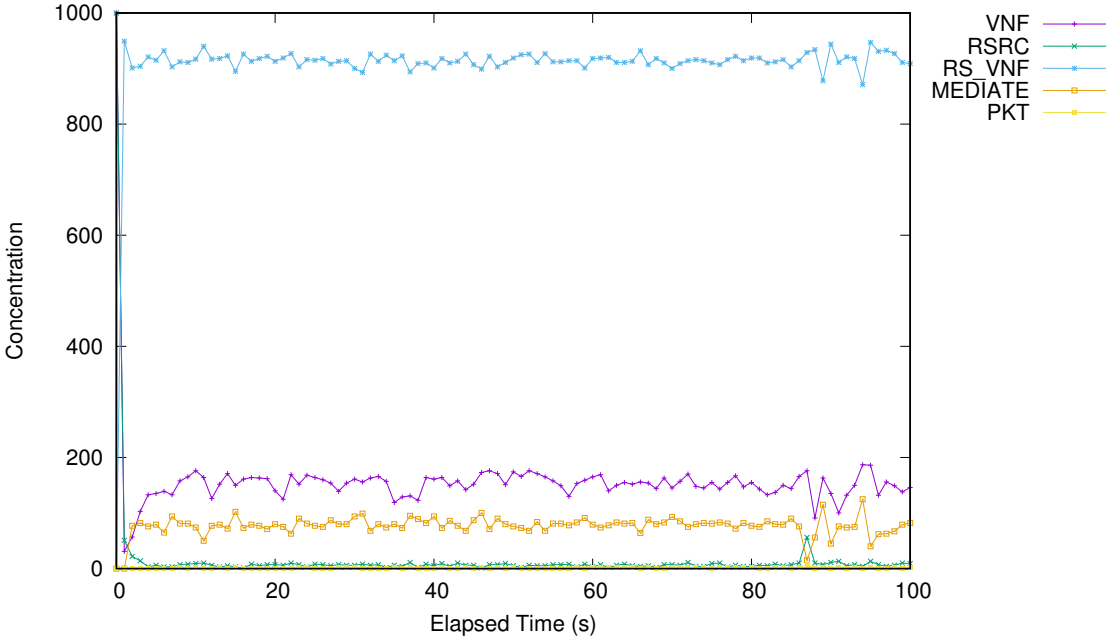
In tuple spaces for firewall and IPS servers, Reactions (15)–(18) are defined. Each reaction rate constant is set to 1.0. In addition, the initial values of concentrations are set to 1,000 for VNF, 1,000 for $RSRC$, and 0 for the others.

6.1.2 Experimental Results and Discussions

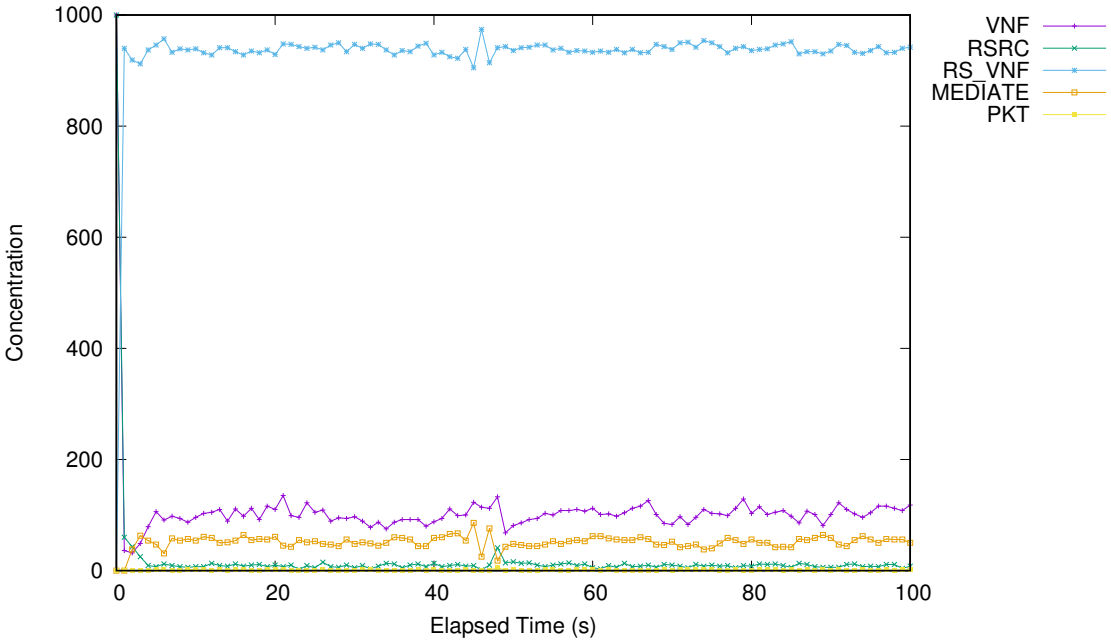
First, we show the results for Exp. A, Exp. B, and Exp. C, where the connection rates are constant. The references for the results to the figures are summarized in Table 6.

In Figures 32, 36, and 40, plotting transitions of the concentrations of chemical substances in each tuple space, the concentrations converge in around 5 seconds. Note that since the number of executed reactions is determined stochastically in the proposed method, the concentrations fluctuates even after the convergence.

In Figures 33, 37, and 41, plotting the changes of observed statistics, the values fluctuate significantly while `httperf` generates connections at a constant rate. This affects directly the concentrations of the chemical substances injected and updated in each tuple space. However, the

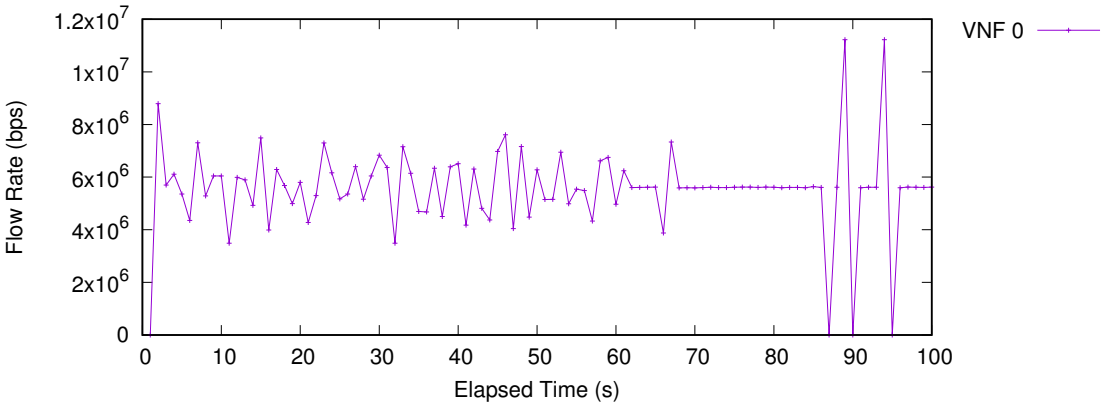


(a) BRTS 0

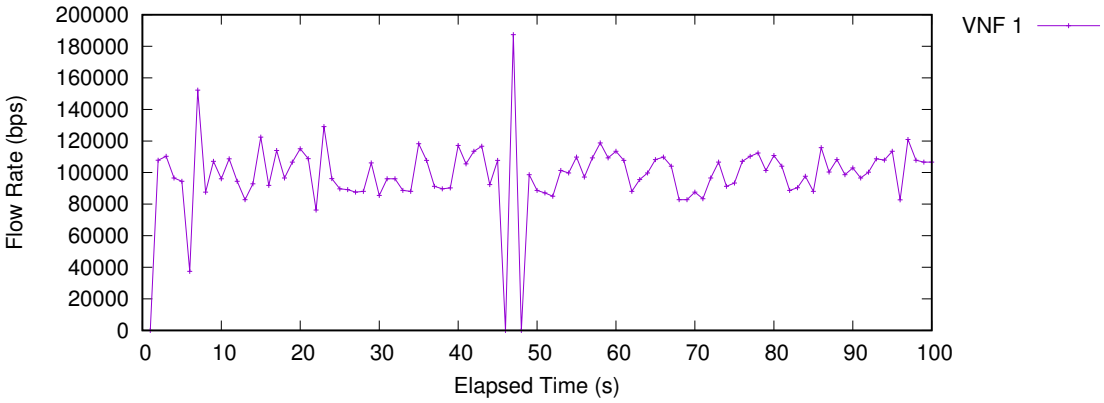


(b) BRTS 1

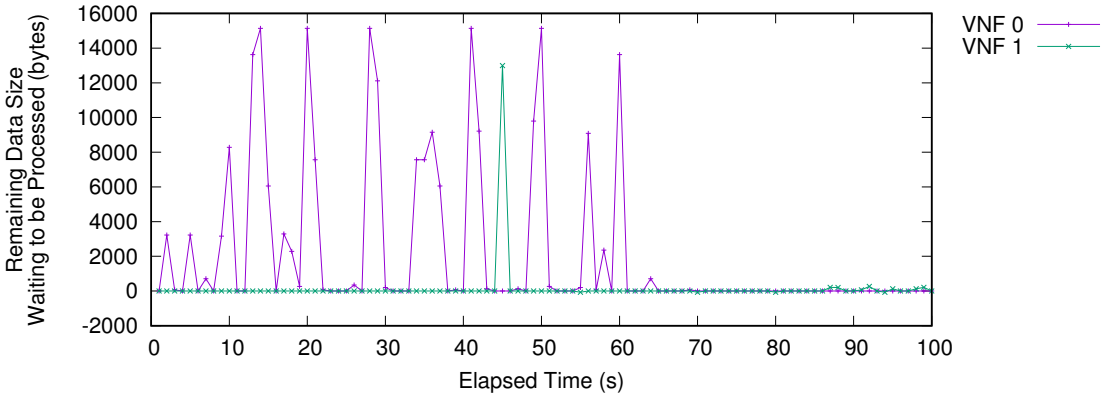
Figure 32: Concentrations of chemical substances for Exp. A



(a) Flow rate for VNF 0



(b) Flow rate for VNF 1



(c) Remaining data size waiting to be processed

Figure 33: Observed statistics for Exp. A

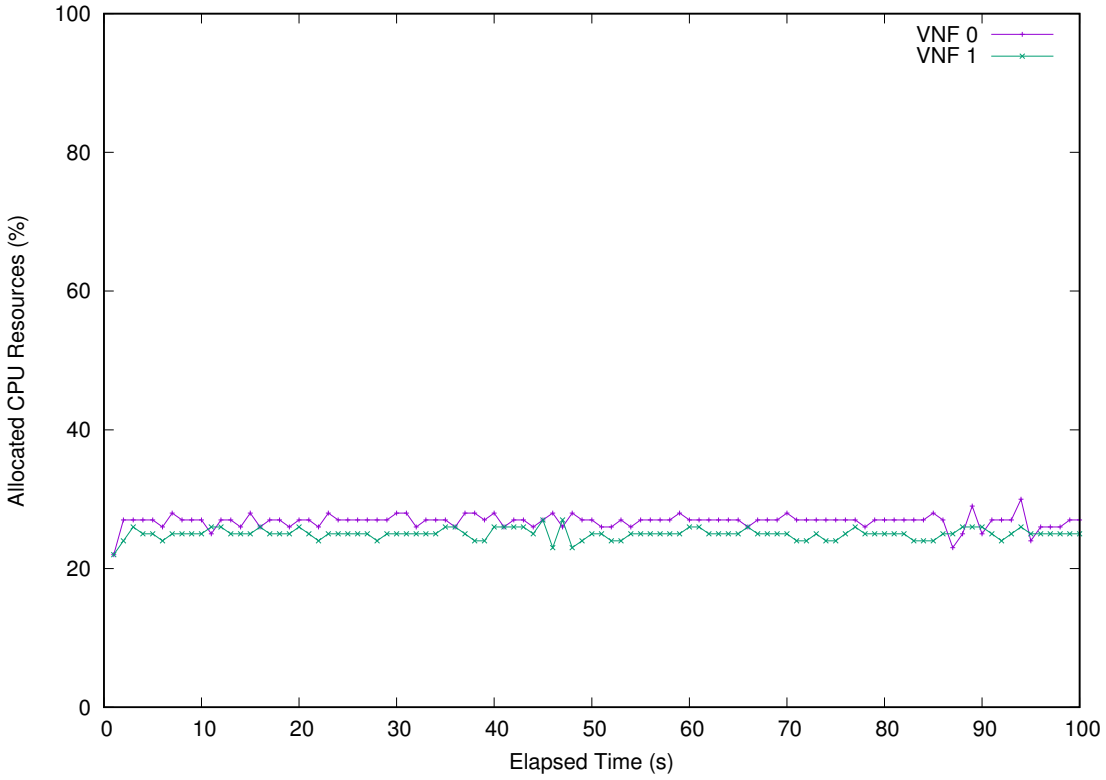


Figure 34: Allocated CPU resources for Exp. A

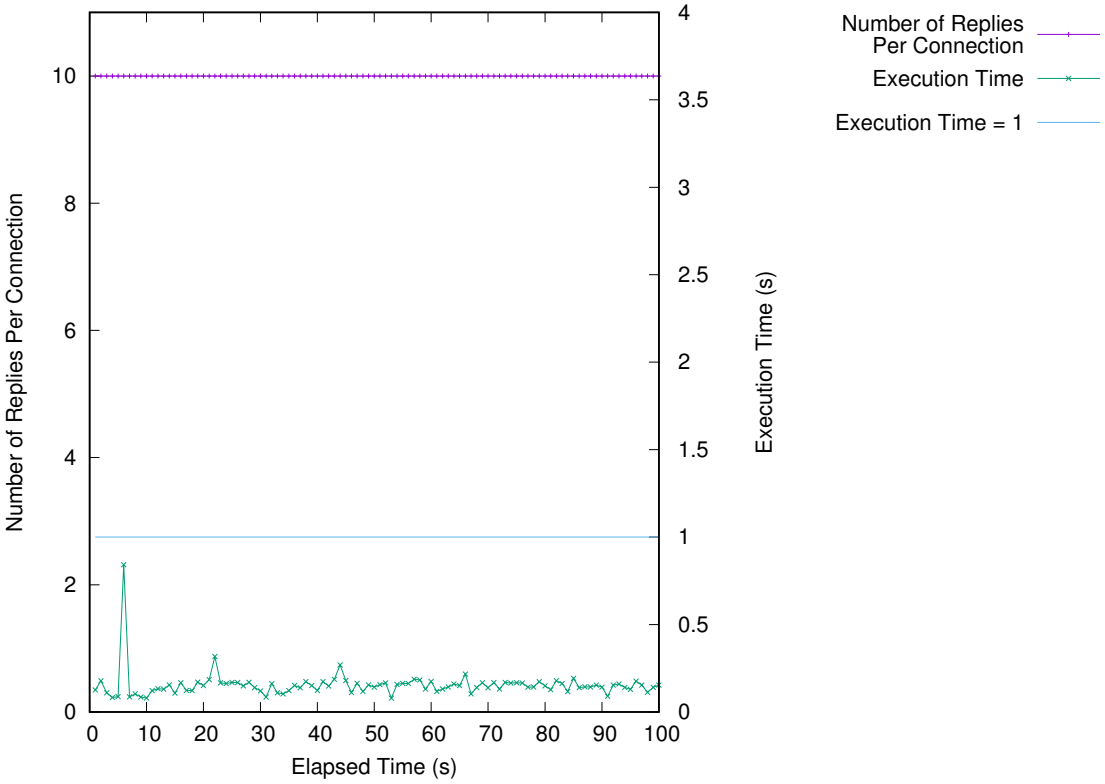
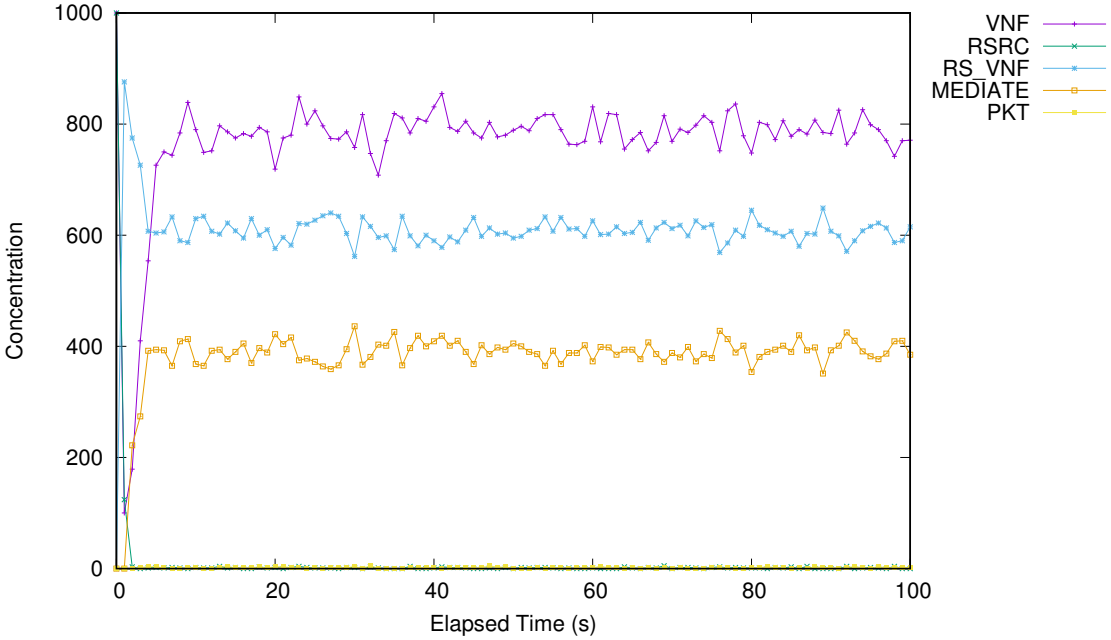
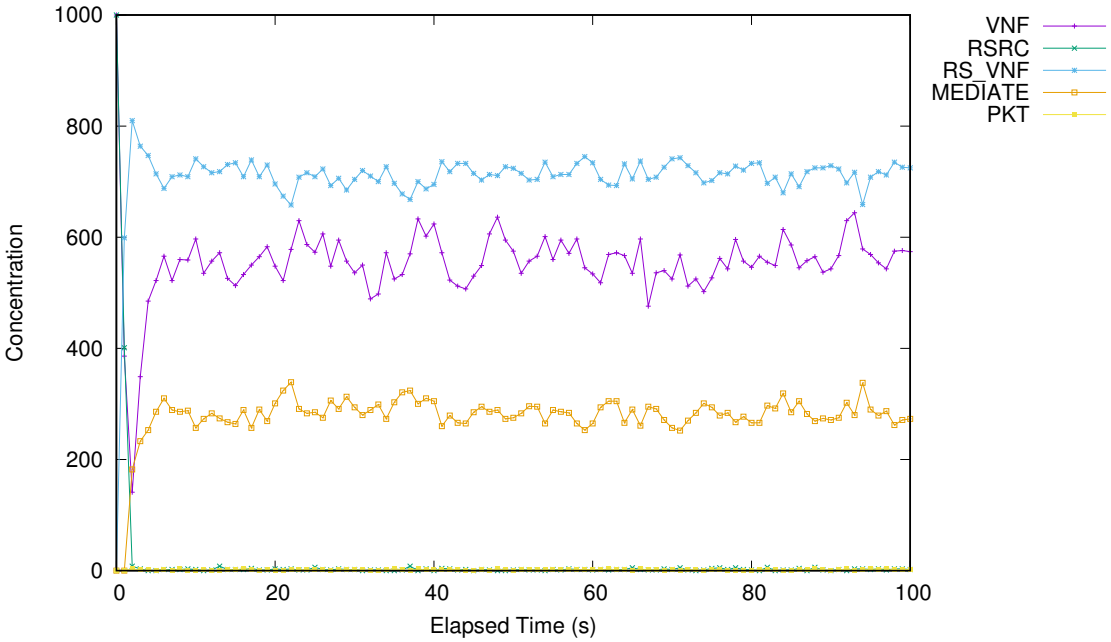


Figure 35: `httperf` performance for Exp. A

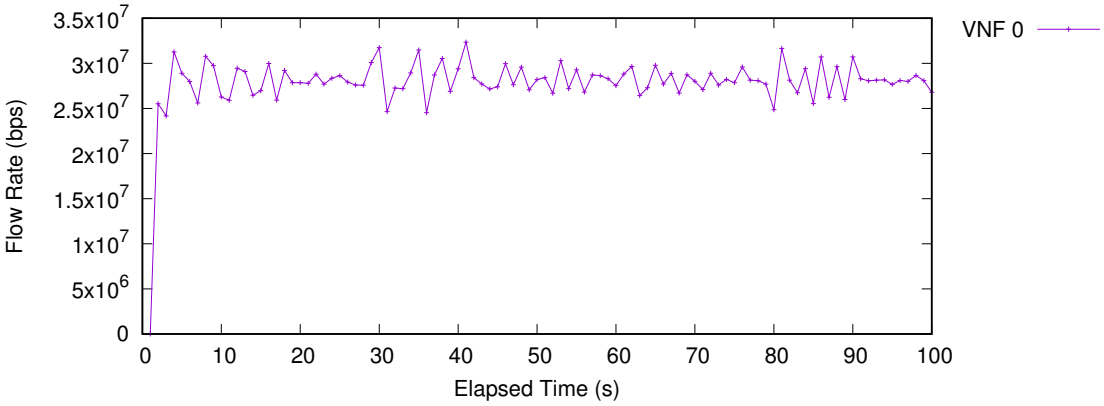


(a) BRTS 0

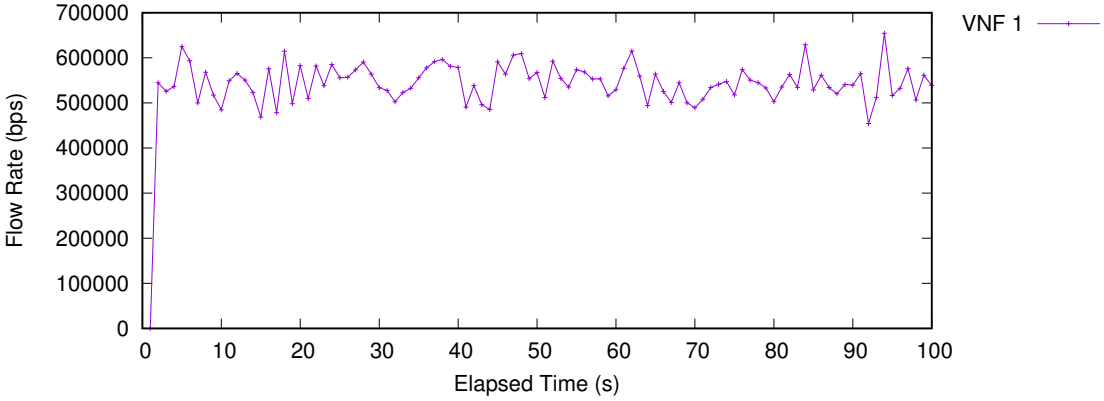


(b) BRTS 1

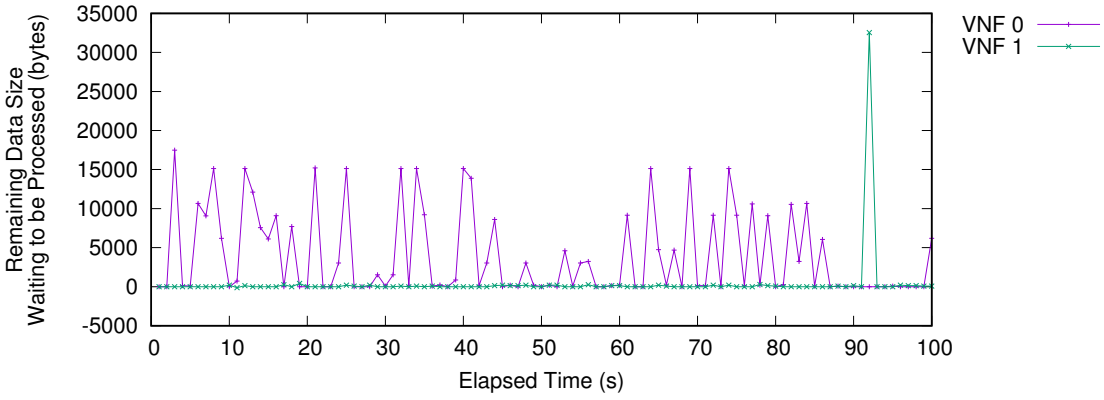
Figure 36: Concentrations of chemical substances for Exp. B



(a) Flow rate for VNF 0



(b) Flow rate for VNF 1



(c) Remaining data size waiting to be processed

Figure 37: Observed statistics for Exp. B

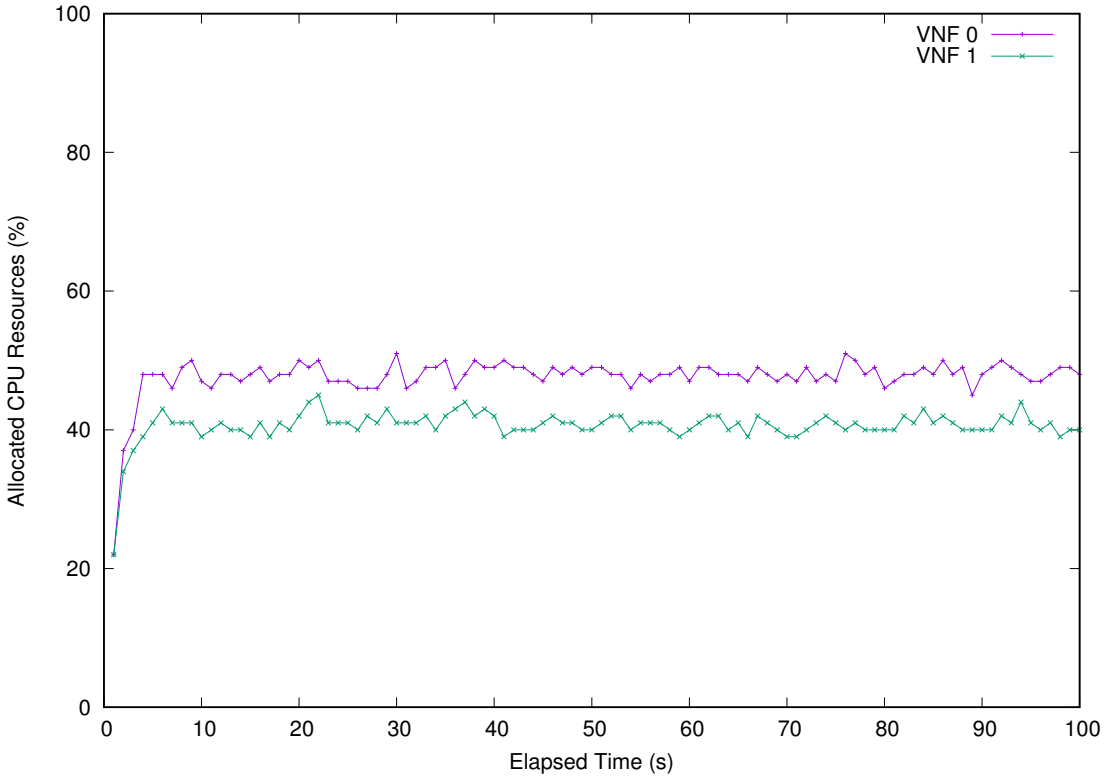


Figure 38: Allocated CPU resources for Exp. B

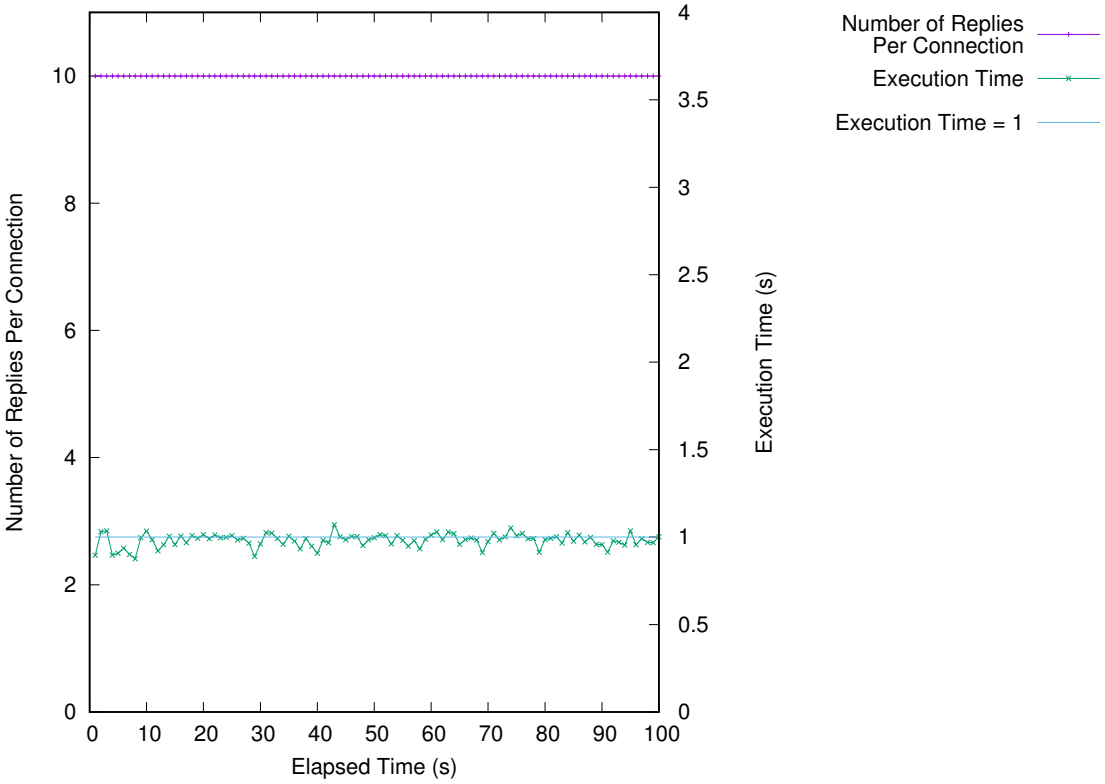
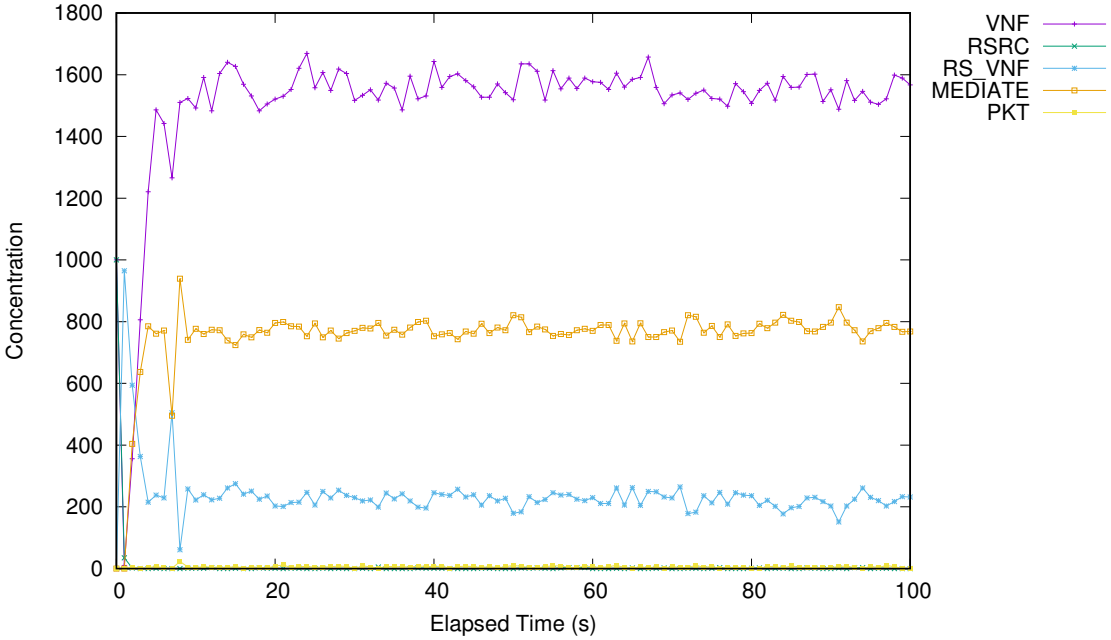
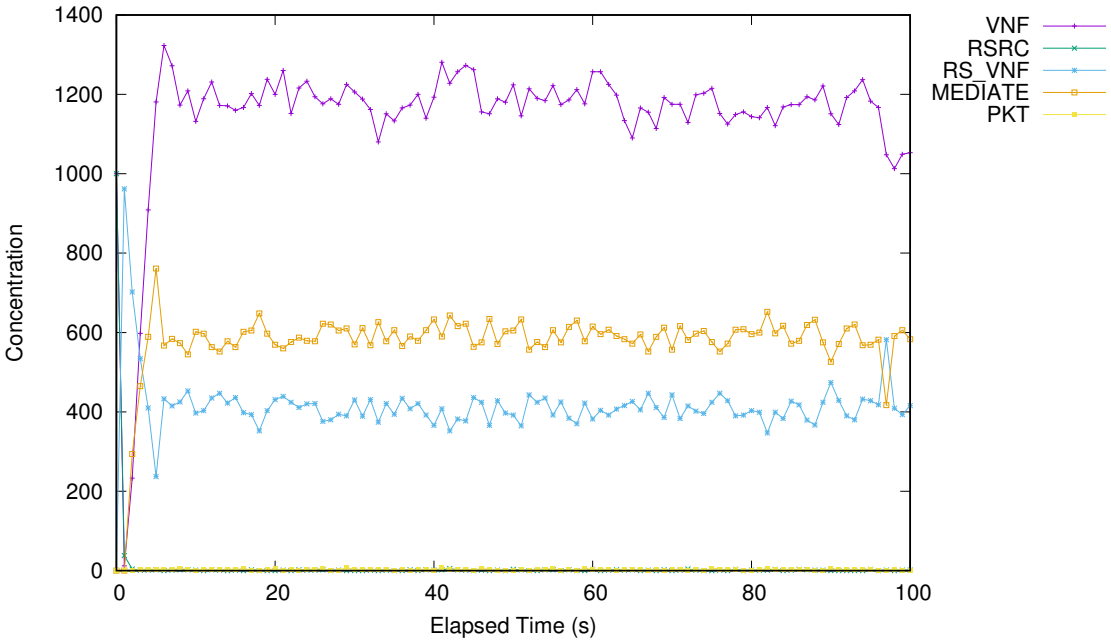


Figure 39: `httperf` performance for Exp. B

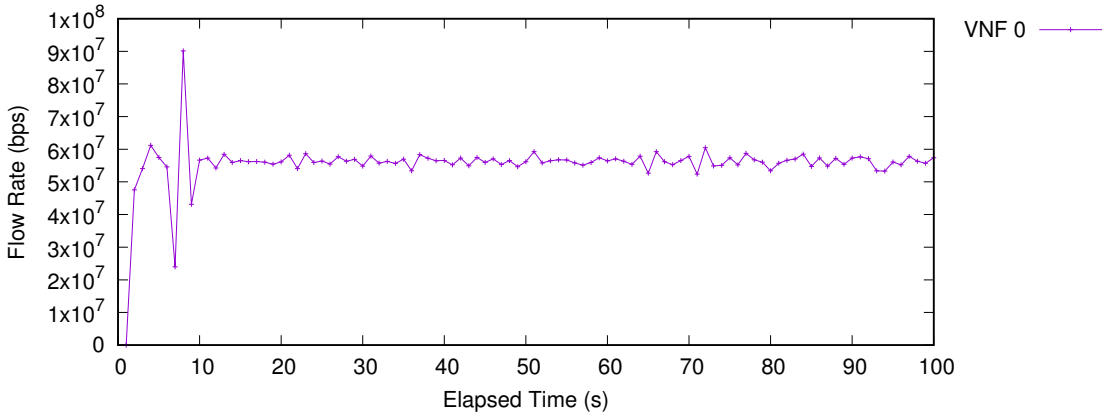


(a) BRTS 0

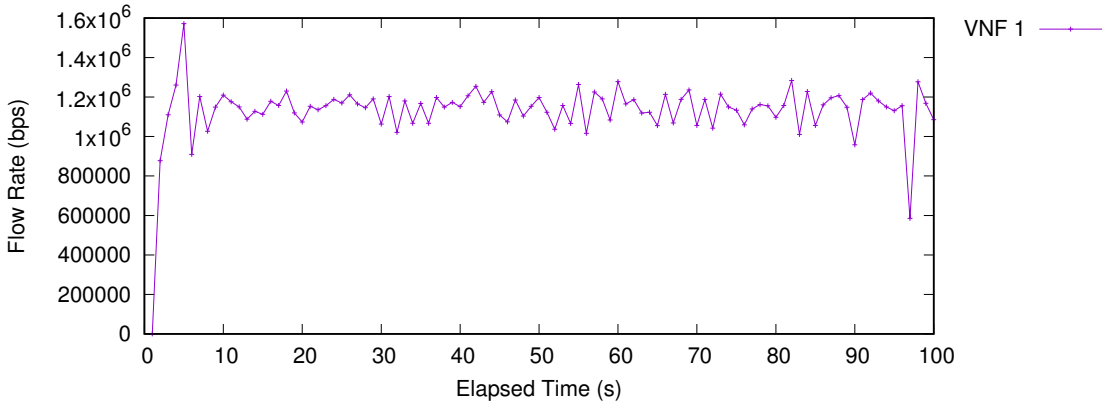


(b) BRTS 1

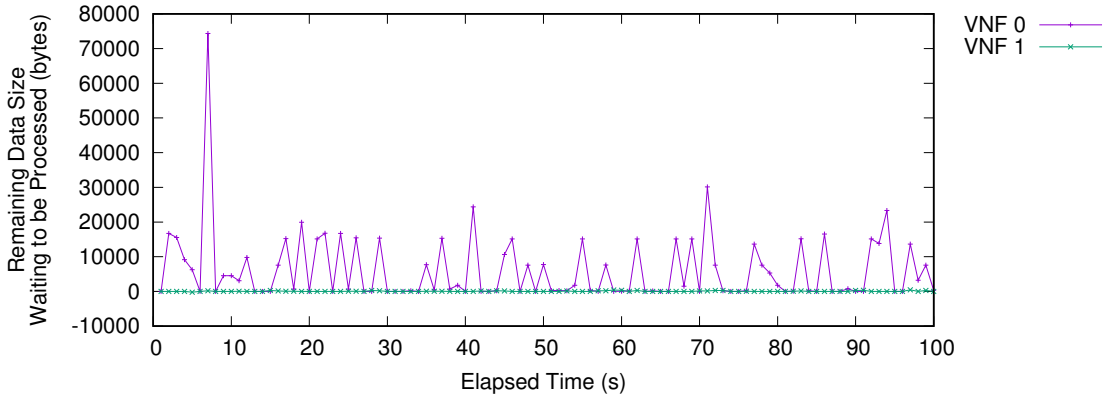
Figure 40: Concentrations of chemical substances for Exp. C



(a) Flow rate for VNF 0



(b) Flow rate for VNF 1



(c) Remaining data size waiting to be processed

Figure 41: Observed statistics for Exp. C

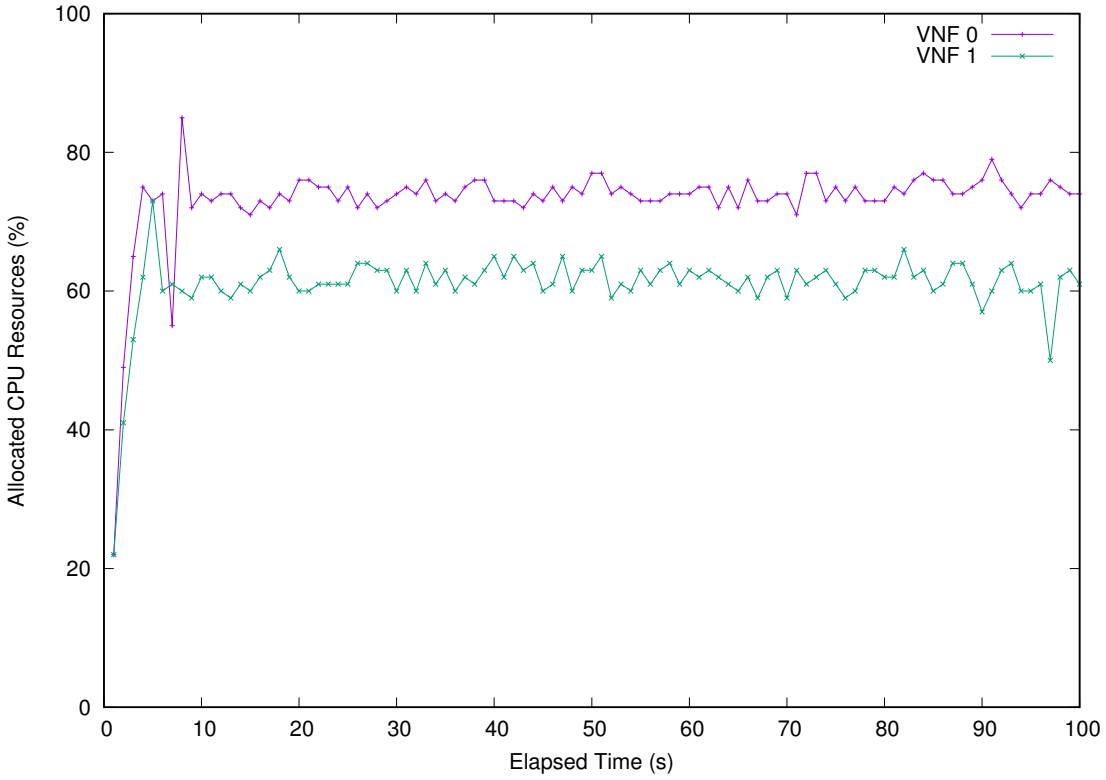


Figure 42: Allocated CPU resources for Exp. C

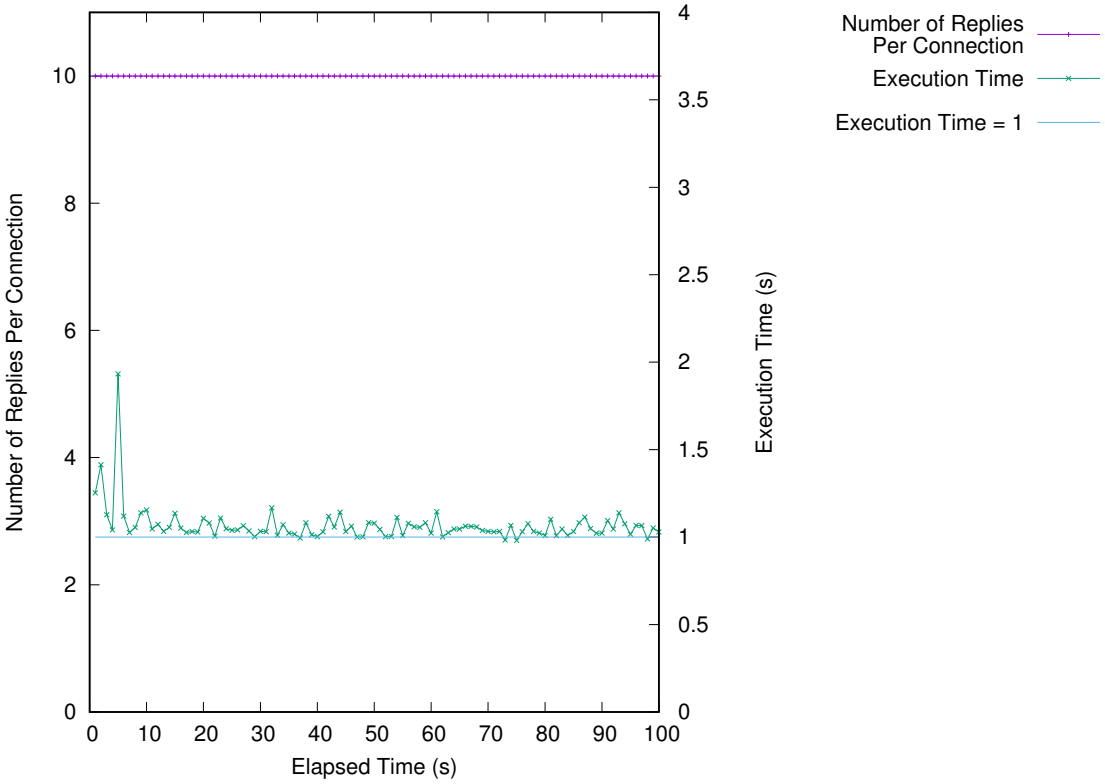


Figure 43: `httperf` performance for Exp. C

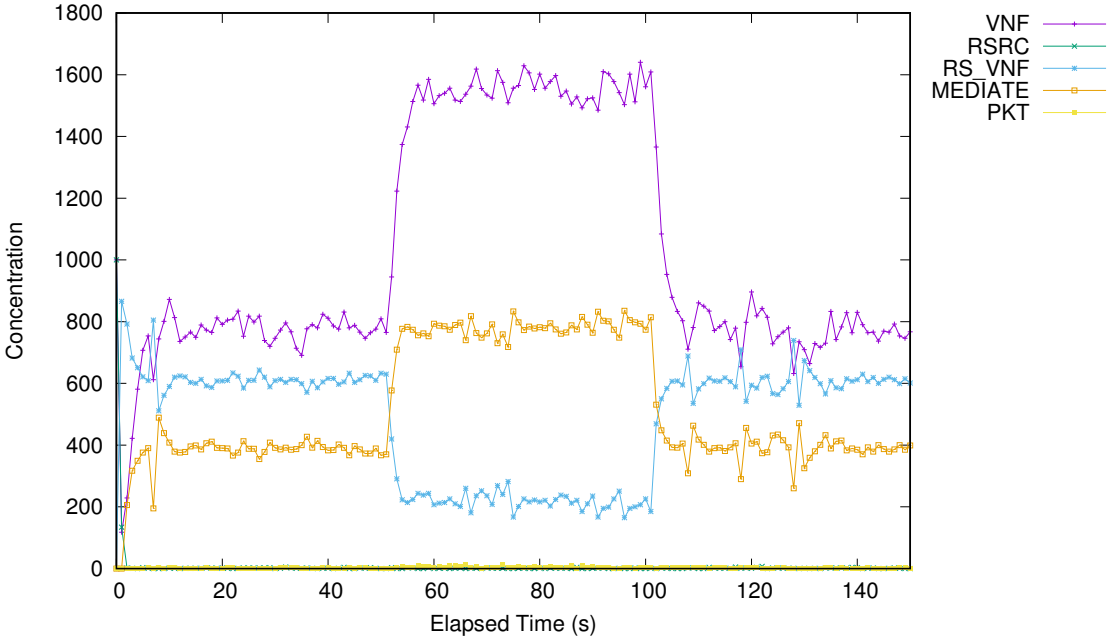
allocated resources to VNFs remains stable as shown in Figures 34, 38, and 42. Also, referring to the result for the remaining data size waiting to be processed in VNFs, it sometimes takes negative values. This is because of the difference in the update timing of the statistics in the switches. Note that such negative values do not affect the behavior of the proposed system. There are two possible solutions to this problem: extending the length of the execution time step and using a moving average of the observed values. Extending the length of the execution time step, the influence of one error in calculating the flow rate is reduced. As a result, the value is less up and down. However, the solution has nothing to do with the remaining data size waiting to be processed in the VNF, so its value does not change. Using a moving average of the observed values, it takes time for the actual flow information to be correctly recognized by the system. As a result, the tracking of the increase or decrease in traffic becomes slow. This behavior is the same when the length of the execution time step is extended. This observation error is not addressed, as it has no significant effect in this experiment.

In Figures 34, 38, and 42 for allocated CPU resources, it is observed that the concentrations calculated from the Equations (48) and (49) as the CPU resources for VNFs are almost correctly allocated after the concentrations have converged.

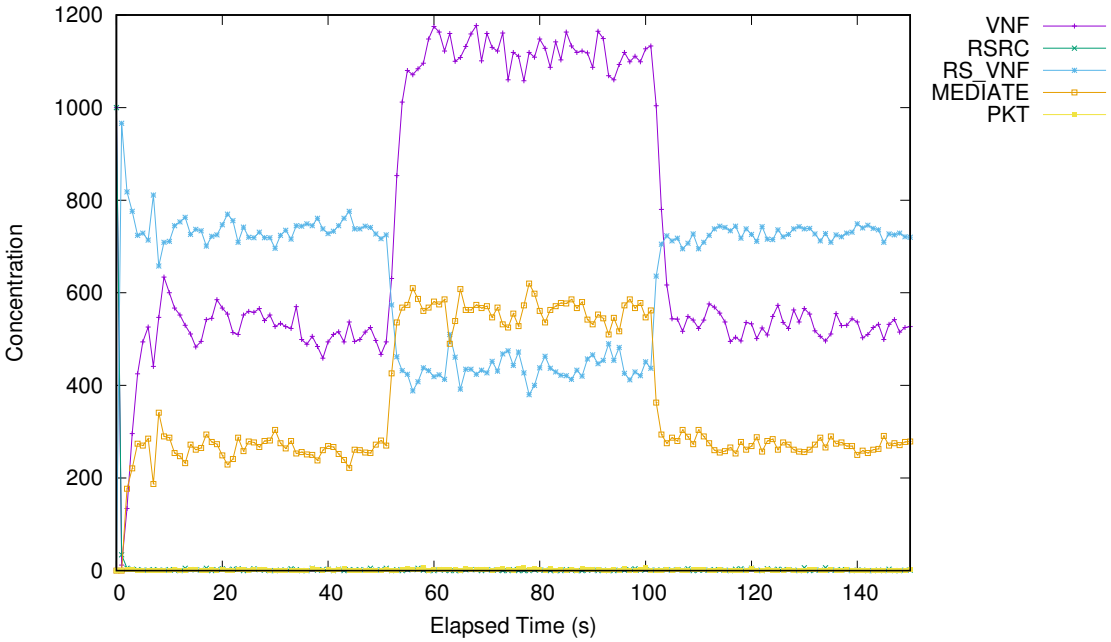
In Figures 35, 39, and 43, plotting `httperf` performance, all requests have been processed for all experiments. The execution time did not greatly exceed 1 second except for a few seconds after the start of the experiment, where CPU resources are insufficient for the flow rate.

We next discuss the results for Exp. D, in which the connection rate changes dynamically during the experiment. As shown in Figures 44 and 46, we can observe that the concentrations converge to appropriate values in short time after the changes in the flow rate. This results in the efficient allocation of CPU resources to VNFs. Figure 47 shows that all `httperf` requests were appropriately processed and the execution time did not significantly exceed 1 second.

From these results, we confirmed that the proposed method can dynamically and adaptively allocate CPU resources to the VNFs.

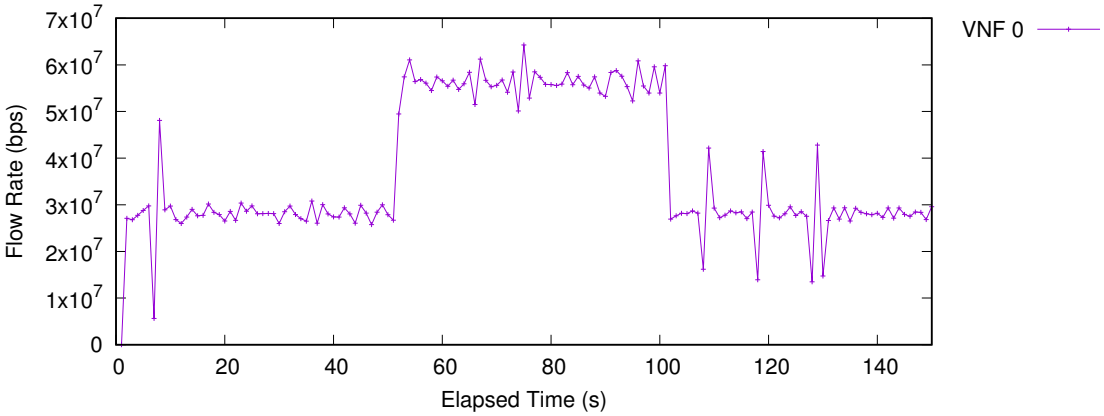


(a) BRTS 0

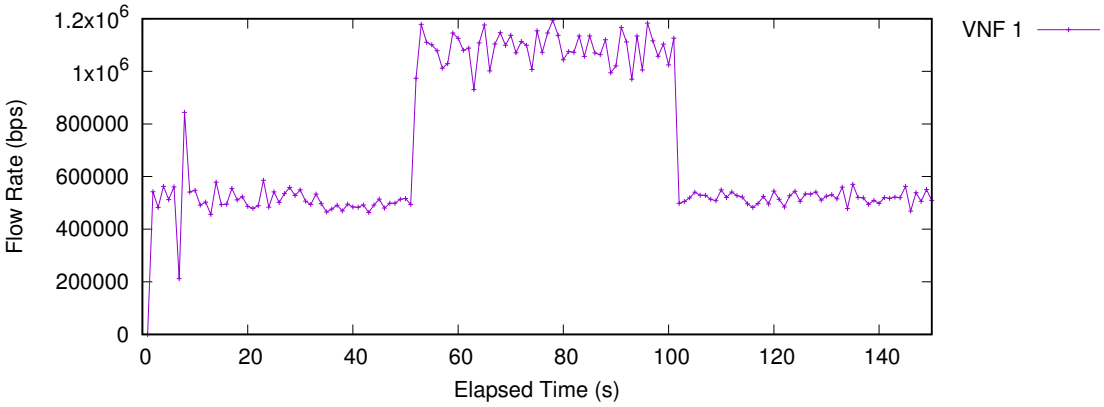


(b) BRTS 1

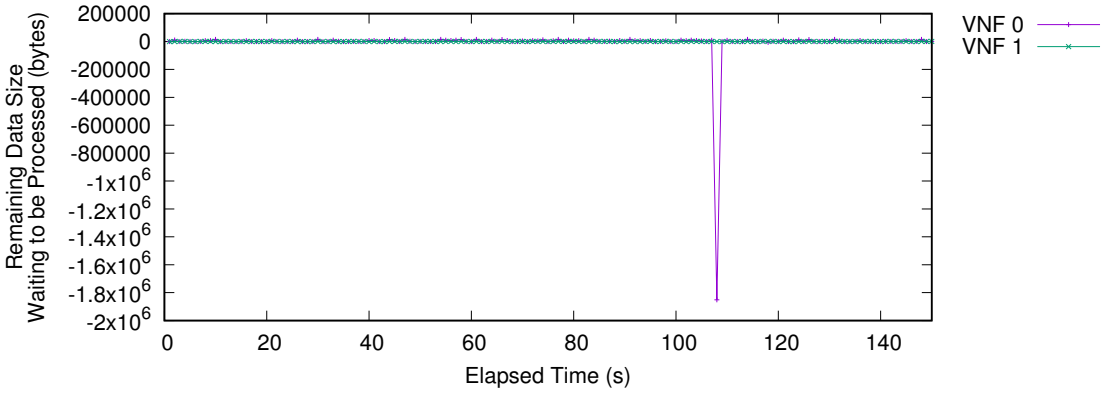
Figure 44: Concentrations of chemical substances for Exp. D



(a) Flow rate for VNF 0



(b) Flow rate for VNF 1



(c) Remaining data size waiting to be processed

Figure 45: Observed statistics for Exp. D

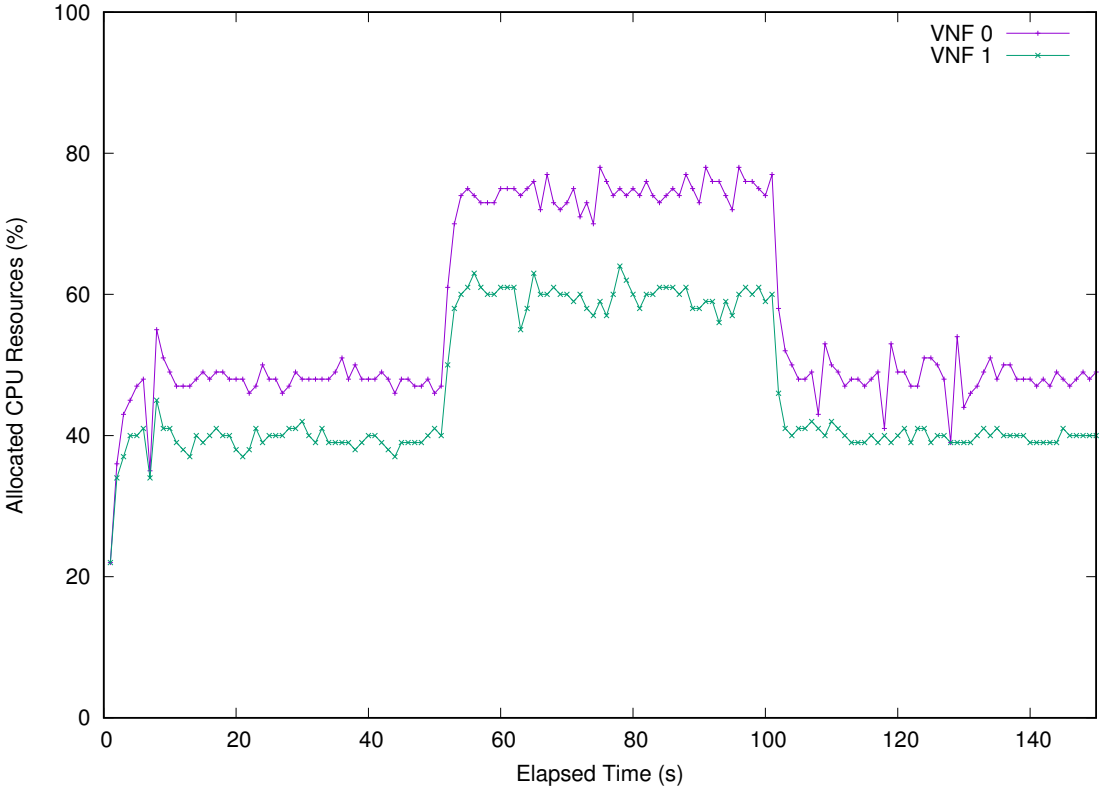


Figure 46: Allocated CPU resources for Exp. D

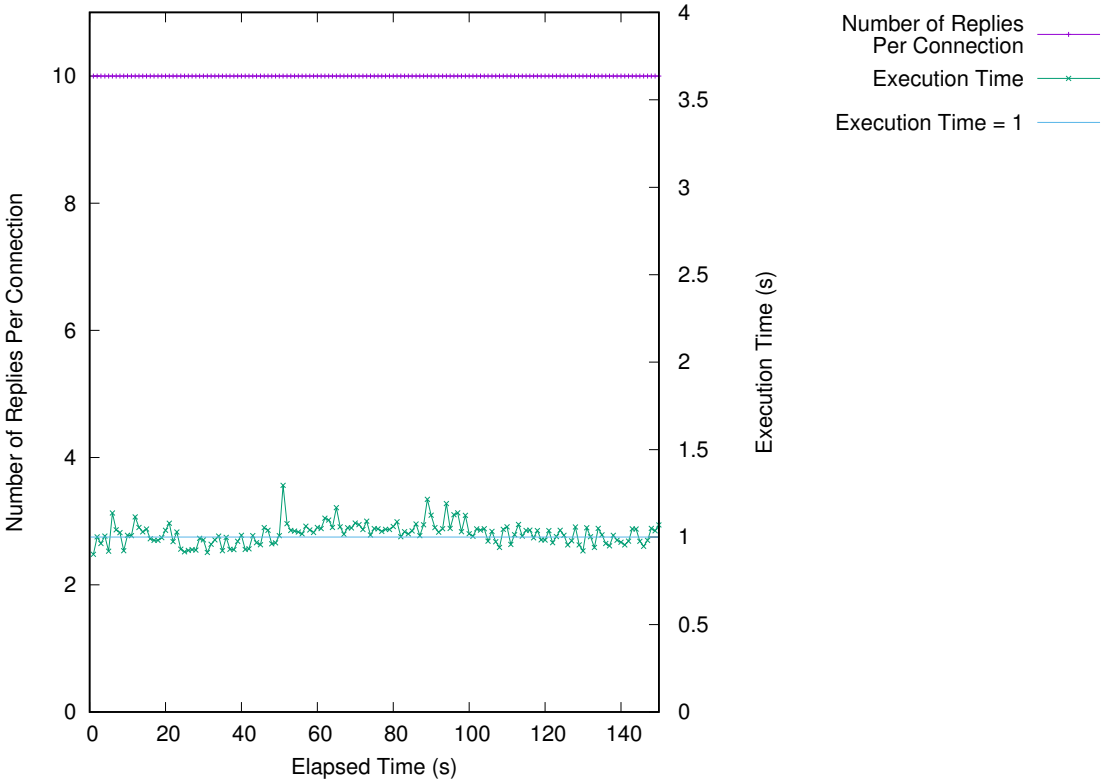


Figure 47: `httperf` performance for Exp. D

6.2 Scenario 2: Resource Allocation and Flow Routing

6.2.1 Evaluation Scenario and Parameter Settings

We consider the scenario shown in Subsection 5.1. Therefore, flows are dynamically routed between the client and the media server. The proposed system performs dynamic resource allocation and flow routing.

The experiment performed in this subsection is called Exp. E. In Exp. E, `httperf` is executed at every second for realizing the constant connection rate for 100 seconds.

For the first 50 seconds, VNF 2 is not available, and BRTS 2 is not connected to other tuple spaces. Therefore, VNF 1 processes all flows that IPS should be applied to. 50 seconds after the start of the experiment, VNF 2 becomes available, and BRTS 2 is connected to other tuple spaces. For simplicity in this implementation, Compute0 executes only one firewall, and Compute1 and Compute2 execute only one IPS. Additionally, BRTS 2 is connected to other tuple spaces from the beginning of the experiment, but does not exchange substances. It starts exchange of substances when 50 seconds have elapsed. The substances are exchanged only between BRTS 1 and BRTS 2. This experiment confirm that the stochastic flow routing based on the concentrations also works properly.

In BRTS 0 for firewall, Reactions (15)–(18) are defined. The initial values of the concentrations are set to 1,000 for VNF, 1,000 for $RSRC$, and 0 for the others. In BRTS 1 for IPS, Reactions (15)–(23) are defined. The initial values of the concentrations are set to 1,000 for VNF, 1,000 for $RSRC$, and 0 for the others. In BRTS 2 for IPS, Reactions (15)–(23) are defined. The initial values of the concentrations are set to 1,000 for VNF and 0 for the others. Additionally, each reaction rate constant is set to 0.2, for the reason described below.

6.2.2 Experimental Results and Discussion

We show the result for Exp. E. The references for the results to the figures are summarized in Table 7. Note that the results for VNF 2 and BRTS 2 except Figure 51 are plotted for the post-connection values only.

In Figure 48, convergence values are the same as Exp. C in the first 50 seconds, when VNF 2 is unavailable and BRTS 2 is not connected. Figure 51 shows that the SFP $\{\text{VNF } 0 \rightarrow \text{VNF } 1\}$ is selected for all flows from the client to the media server in this period. However, the concentrations

Table 5: Parameters for experiments in Subsection 6.1

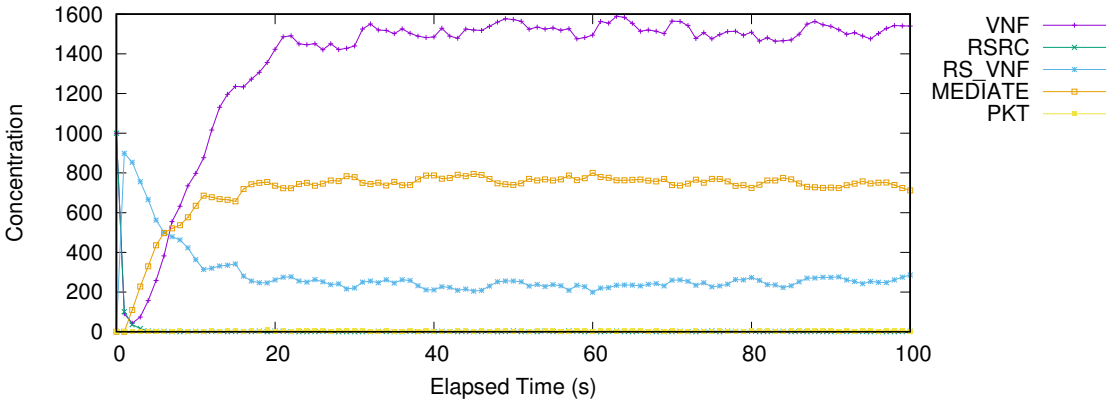
Exp. ID	Connection rate	Exp. duration (s)
A	1	100
B	5	100
C	10	100
D	5, 10, 5	150

Table 6: Reference to figures in Subsection 6.1

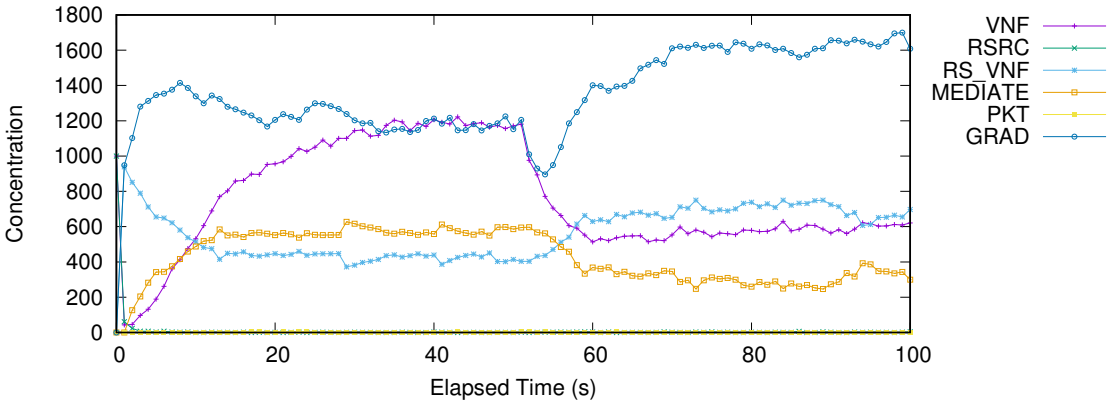
Exp. ID	Concentrations of Chemical Substances	Observed Statistics	Allocated CPU Resources	`httperf` Performance
A	Figure 32	Figure 33	Figure 34	Figure 35
B	Figure 36	Figure 37	Figure 38	Figure 39
C	Figure 40	Figure 41	Figure 42	Figure 43
D	Figure 44	Figure 45	Figure 46	Figure 47

Table 7: Figures depicting experimental results in Subsection 6.2

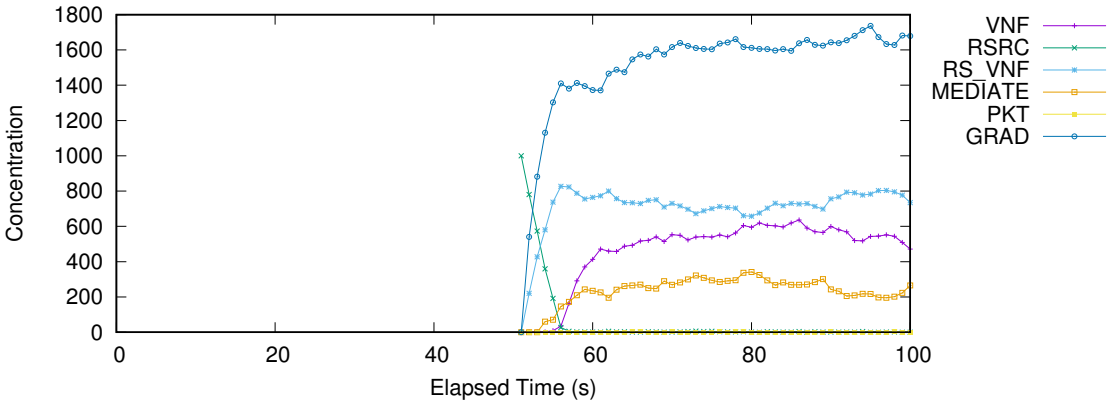
Exp. ID	Concentrations of Chemical Substances	Observed Statistics	Allocated CPU Resources	Selection of Path	`httperf` Performance
E	Figure 48	Figure 49	Figure 50	Figure 51	Figure 52



(a) BRTS 0

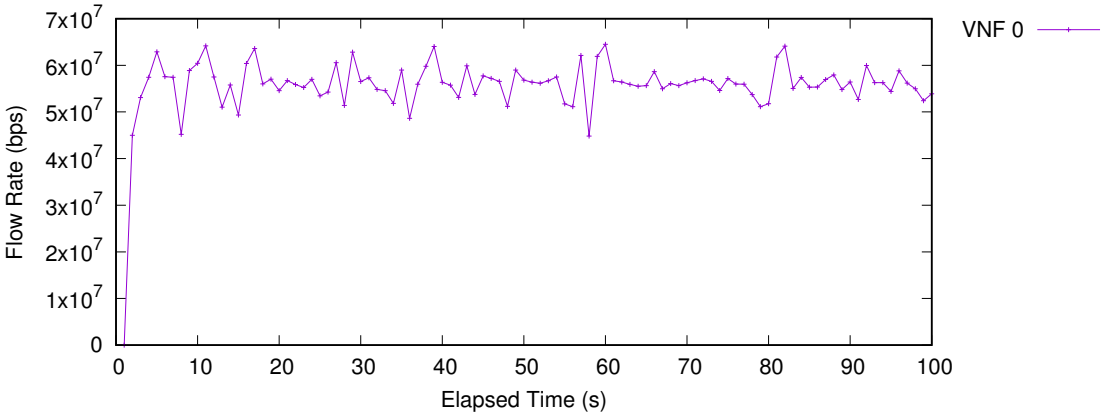


(b) BRTS 1

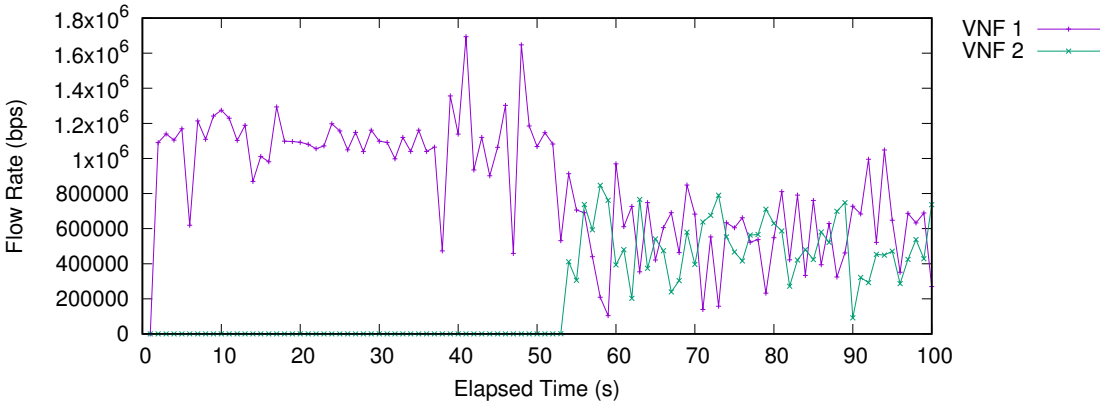


(c) BRTS 2

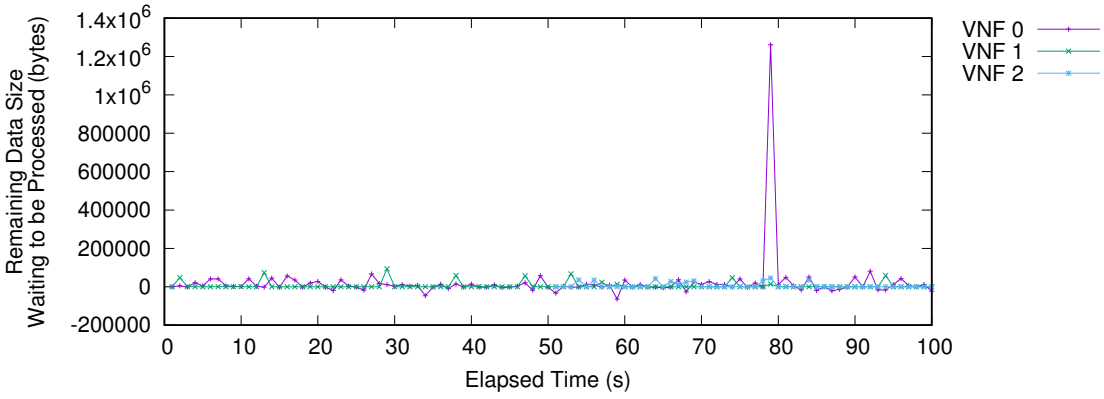
Figure 48: Concentrations of chemical substances for Exp. E



(a) Flow rate for VNF 0



(b) Flow rate for VNF 1 and VNF 2



(c) Remaining data size waiting to be processed

Figure 49: Observed statistics for Exp. E

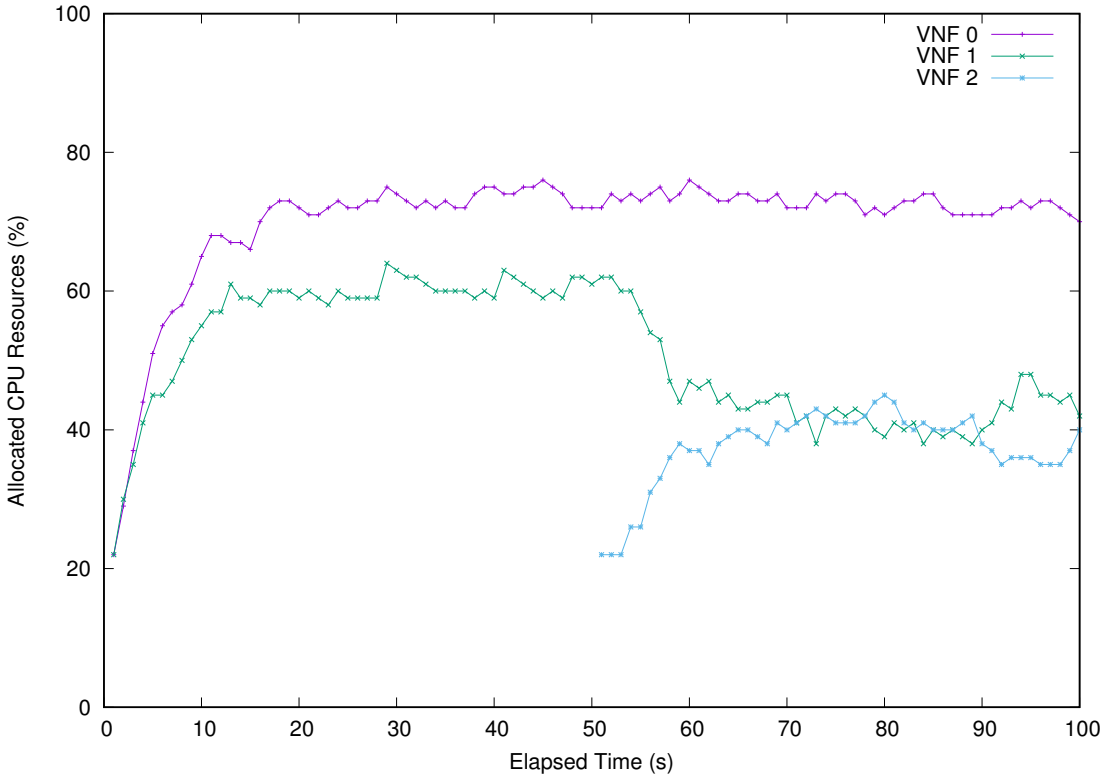


Figure 50: Allocated CPU resources for Exp. E

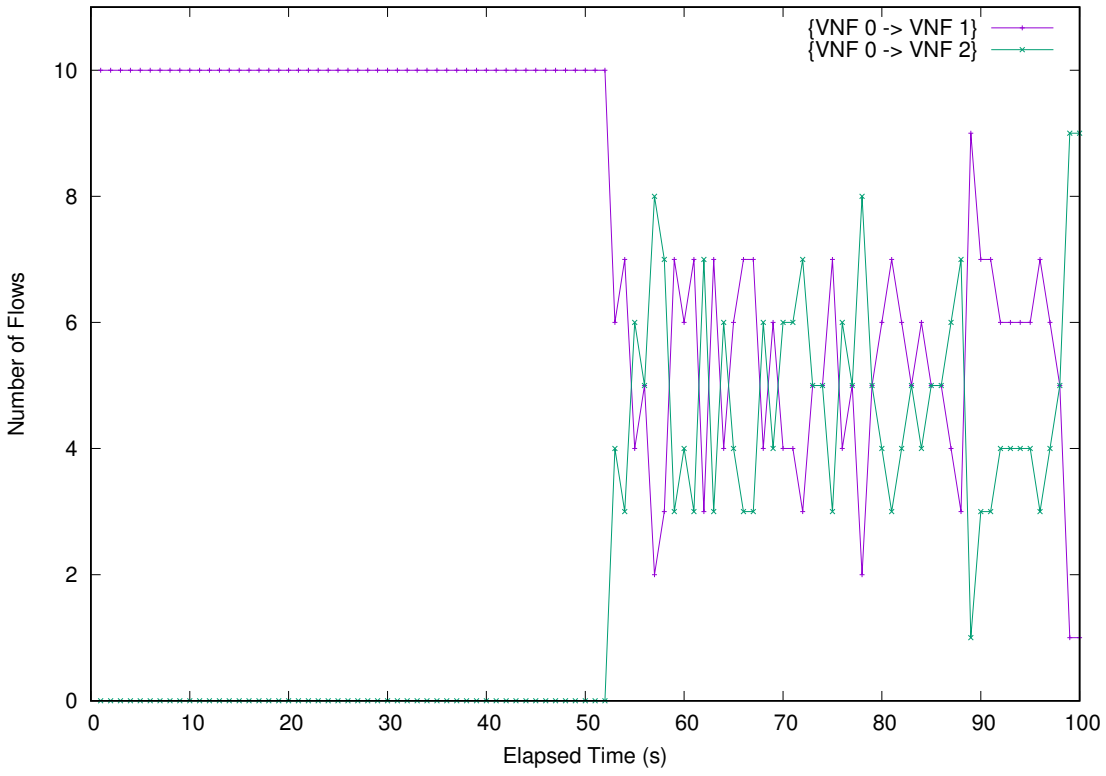


Figure 51: Number of flows to which each SFP is selected for flows from the client to the media server for Exp. E

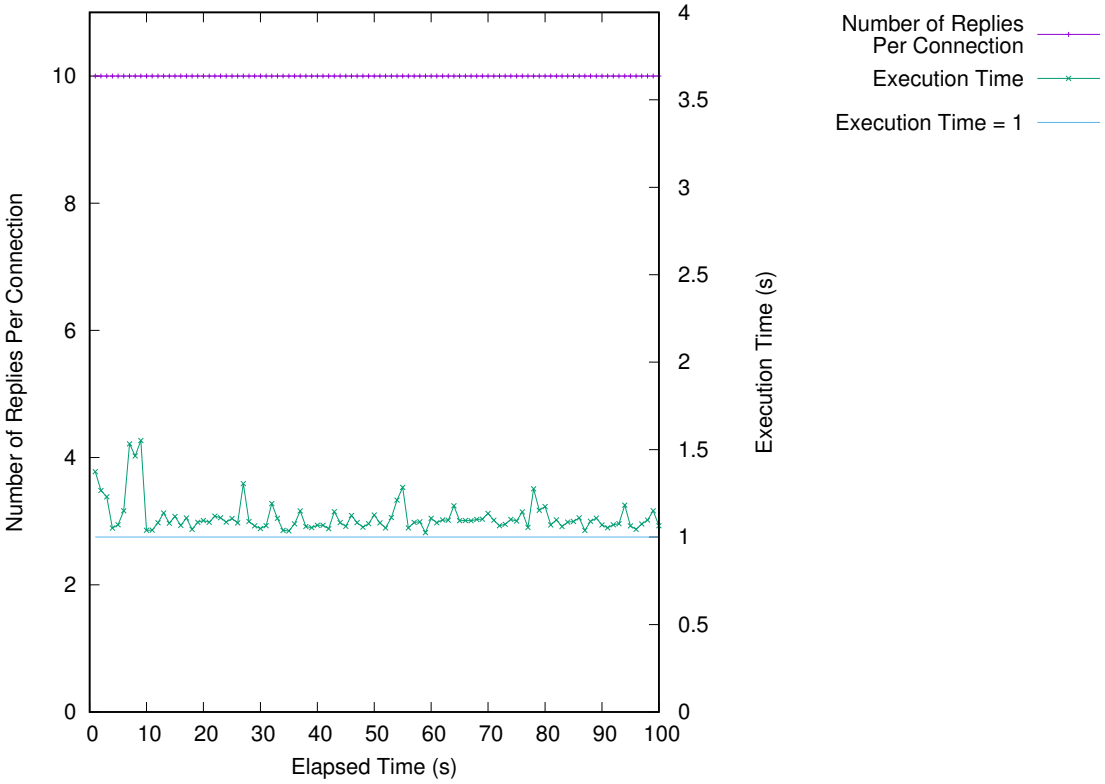


Figure 52: `httperf` performance for Exp. E

converge in around 20 seconds, while the concentrations converge in around 5 seconds in Exp. C, whose connection rate of `httperf` is the same. This is because each reaction rate constant is changed from 1.0 to 0.2, and the execution speed of each biochemical reaction is reduced. This change is made to solve the problem that would occur when performing this experiment with each reaction rate constant set to 1.0. With the reaction rate constants, the calculation in BRTS does not end within 1 second, which is the length of the execution time step. Then, the length of the execution time step cannot be maintained for the entire system. This problem occurs, for example, immediately after the start of the experiment, when the substance concentration takes a value far apart from the convergence value. Additionally, since many types of reactions have been defined, the total number of reactions is very large. Then, the recursive calculation process of BRTS described in Subsection 4.2.2 is executed a lot. As the result, execution time is very long. As a solution to this problem, in this experiment, each reaction rate constant is changed from 1.0 to 0.2. However, the reaction rate constant is originally a parameter to be determined according to the transient characteristics and the degree of overshoot. Therefore, an ideal solution is to improve the performance of the machine realizing BRTS so that the calculation is completed within an execution time step.

We next discuss the latter half of 50 seconds, when VNF 2 is available and BRTS 2 is connected.

In Figure 48, BRTS 0 has not changed since the first half. In BRTS 1, immediately after BRTS 2 is connected, the concentration of $GRAD$ decreases and then increases. This decrease is due to the diffusion of $GRAD$ to BRTS 2. This increase is due to the reaction toward the convergence value. In BRTS 2, each concentration go to the convergence value after connection to the system. The convergence values become the same as BRTS 1.

In Figure 51, we can observe that SFP $\{$VNF 0 $\rightarrow$ VNF 2$\}$ is selected for the flow from the client to the media server little after the connection of BRTS 2. As the experiment progresses, the SFP using VNF 1 and the SFP using VNF 2 are stochastically selected with the same probability. This indicates that the flow routing is properly performed according to the concentrations of $GRAD$ in each tuple space. As a result, the flow rate for each VNF also changes. It affects the concentrations in the Figure 48. In the setting of this experiment, each SFP is finally selected with the same probability, but it is also possible to change this control by changing the reaction rate constants [12]. For example, in cloud edge computing, it is possible to control to use only VNFs

executed in the edge below a certain flow rate, and to use VNFs executed in the cloud above that.

In Figure 50 for allocated CPU resources, it is observed that the resources calculated from the Equations (48) and (49) as the CPU resources for VNFs are almost correctly allocated after the concentrations have converged.

In Figure 52, plotting `httperf` performance, all requests have been processed for all experiments. The execution time did not greatly exceed 1 second except for a few seconds after the start of the experiment, where CPU resources are insufficient for the flow rate.

From these results, we confirmed that the proposed method can dynamically and adaptively route the flows.

7 Conclusion and Future Work

In this thesis, we implemented a construction method of service space in virtualized network system based on biochemically-inspired tuple space model as components on the NFV framework proposed by ETSI ISG for confirming the applicability and effectiveness of the proposed method in the actual NFV environment. In detail, we exploited an NFV environment using OPNFV and implemented the proposed NFV management system as a system of four main components. By extensive experimental evaluations, we showed that the proposed NFV management system worked properly with OPNFV for realizing video streaming services, in terms of adaptively allocating server resources to VNFs in accordance with the amount of traffic. It is also shown that the proposed system can dynamically distribute incoming flows to multiple VNFs for load balancing.

For future work, we plan to propose a method to automatically determine the parameters for VNFs required to convert between actual values and substance concentrations for avoiding preliminary experiments as in Subsection 5.6.2 in this thesis. It is also necessary to extend the proposed method to include more factors of the actual network environment, such as the effect of the propagation delays and the link bandwidths of the underlying networks.

Acknowledgments

My master's degree studies are supported by many people. I would like to thank them for helping. First, I would like to express my deepest gratitude to my supervisor, Professor Morito Matsuoka. He taught my attitude towards research and gave my support and useful comments in various situations. Additionally, I would like to show my greatest appreciation to Professor Masayuki Murata. He gave me insightful and incisive advice. Thanks to him, I was able to refine my research without losing sight of its direction. Furthermore, I would like to express the deepest appreciation to Professor Go Hasegawa. I was able to write this thesis because he taught me cordially how to do research. His advice was accurate and encouraging and he has supported my trial and error patiently. Also, I would like to appreciate to Assistant Professor Yuya Tarutani. He gave me beneficial comments about my research and life in the laboratory. Moreover, I would like to thank to students of Matsuoka Laboratory for their support of my research and laboratory life. Finally, I truly thank my friends and colleagues in Osaka University, for their great encouragement and support.

References

[1] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347–2376, Fourthquarter 2015.

[2] ETSI, "Network Functions Virtualisation - White Paper1." available at `https://portal.etsi.org/nfv/nfv_white_paper.pdf`.

[3] K. Ingham and S. Forrest, "A History and Survey of Network Firewalls," tech. rep., University of New Mexico, Jan. 2002.

[4] M. Finsterbusch, C. Richter, E. Rocha, J.-A. Muller, and K. Hanssgen, "A Survey of Payload-Based Traffic Classification Approaches," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 2, pp. 1135–1156, Oct. 2013.

[5] D. Wing, "Network Address Translation: Extending the Internet Address Space," *IEEE Internet Computing*, vol. 14, no. 4, pp. 66–70, June 2010.

[6] M. Olsson, S. Rommer, C. Mulligan, S. Sultana, and L. Frid, *SAE and the Evolved Packet Core: Driving the Mobile Broadband Revolution*. Academic Press, Aug. 2009.

[7] ETSI, "Network Functions Virtualisation - White Paper2." available at `https://portal.etsi.org/nfv/nfv_white_paper2.pdf`.

[8] I. Foster, C. Kesselman, C. Lee, B. Lindell, K. Nahrstedt, and A. Roy, "A Distributed Resource Management Architecture that Supports Advance Reservations and Co-allocation," in *Proceedings of 1999 Seventh International Workshop on Quality of Service*, pp. 27–36, May 1999.

[9] M. Viroli, M. Casadei, S. Montagna, and F. Zambonelli, "Spatial Coordination of Pervasive Services through Chemical-Inspired Tuple Spaces," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 6, no. 14, pp. 1–24, June 2011.

[10] G. Hasegawa and M. Murata, "Biochemically-inspired Method for Constructing Service Space in Virtualized Network System," in *Proceedings of ICIN 2016*, Mar. 2016.

[11] K. Sakata, “Adaptive and Autonomous Placement Method of Virtualized Network Functions based on Biochemical Reactions,” Master’s thesis, Osaka University, Feb. 2018.

[12] R. Kurokawa, “Biochemically-inspired, Adaptive, and Autonomous VNF Control for Service Function Chaining,” Master’s thesis, Osaka University, Feb. 2019.

[13] ETSI GS NFV 002, “Network Functions Virtualisation (NFV); Architectural Framework.” available at http://www.etsi.org/deliver/etsi_gs/nfv/001_099/002/01.01.01_60/gs_nfv002v010101p.pdf.

[14] ETSI GS NFV 005, “Network Functions Virtualisation (NFV); Ecosystem; Report on SDN Usage in NFV Architectural Framework.” available at https://www.etsi.org/deliver/etsi_gs/NFV-EVE/001_099/005/01.01.01_60/gs_nfv-eve005v010101p.pdf.

[15] “OPNFV: Home.” available at https://www.opnfv.org.

[16] “GitHub - httperf/httperf: The httperf HTTP load generator.” available at https://github.com/httperf/httperf.

[17] “Home - OpenDaylight.” available at https://www.opendaylight.org.

[18] “Network Service Header (NSH).” available at https://www.rfc-editor.org/rfc/pdfrfc/rfc8300.txt.pdf.

[19] R. N. Goldberg, Y. B. Tewari, D. Bell, and K. Fazio, “Thermodynamics of Enzyme-Catalyzed Reactions,” *Sience Direct*, vol. 20, no 16, pp. 2874–2877, Dec. 2004.

[20] L. Michaelis, M. L. Menten, K. A. Johnson, and R. Goody, “The Original Michaelis Constant: Translation of the 1913 Michaelis-Menten Paper,” *Biochemistry*, vol. 50, pp. 8264–8269, Sep. 2011.

[21] R. Mijumbi, J. Serrat, and J.-L. Gorricho, “Network Function Virtualization: State-of-the-art and Research Challenges,” *IEEE Communications Surveys & Tutorials Tutorials*, vol. 18, no. 1, pp. 236–262, Sep. 2015.

[22] R. Mijumbi, J. Serrat, J.-L. Gorricho, S. Latré, M. Charalambides, and D. Lopez, "Management and Orchestration Challenges in Network Function Virtualization," *IEEE Communications Magazine*, vol. 54, no. 1, pp. 98–105, Jan. 2016.

[23] "Home - OpenStack is open source software for creating private and public clouds." available at `https://www.openstack.org`.

[24] S. G. Soriga and M. Barbulescu, "A Comparison of the Performance and Scalability of Xen and KVM Hypervisors," in *Proceedings of 2013 RoEduNet International Conference 12th Edition: Networking in Education and Research*, Sept. 2013.

[25] "Open vSwitch." available at `http://www.openvswitch.org`.

[26] H. Li, Y. Cao, L. Petzold, and D. T. Gillespie, "Algorithms and Software for Stochastic Simulation of Biochemical Reacting Systems," *Biotechnology Progress*, vol. 24, no 1, pp. 56–61, Feb. 2008.

[27] C. V. Rao and A. Arkin, "Stochastic Chemical Kinetics and the Quasi-steady-state Assumption: Application to the Gillespie Algorithm," *Journal of Chemical Physics*, vol. 118, no 11, pp. 4999–5010, Aug. 2002.

[28] Y. Cao, D. T. Gillespie, and L. R. Petzold, "Avoiding Negative Populations in Explicit Poisson Tau-Leaping," *The Journal of Chemical Physics*, vol. 123, no 5, 2005.

[29] "CPU limit." available at `http://cpulimit.sourceforge.net`.

[30] "OpenFlow Switch Specification." available at `https://www.opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.3.1.pdf`.

[31] "Service Function Chaining (SFC) Architecture." available at `https://www.rfc-editor.org/rfc/pdfrfc/rfc7665.txt.pdf`.

[32] "NGINX — High Performance Load Balancer, Web Server, & Reverse Proxy." available at `https://www.nginx.com`.

[33] "netfilter/iptables project homepage - The netfilter.org project." available at `https://www.netfilter.org`.

[34] "Snort - Network Instruction Detection & Prevention System." available at https://www.snort.org.

[35] H. Hisamatsu, G. Hasegawa, and M. Murata, "Network Friendly Transmission Control for Progressive Download over TCP," *Journal of Communications*, vol. 7, no. 14, pp. 213–221, Mar. 2012.

[36] L. Cherkasova and R. Gardner, "Measuring CPU Overhead for I/O Processing in the Xen Virtual Machine Monitor," in *Proceedings of USENIX Annual Technical Conference*, pp. 387–390, Apr. 2005.

[37] Z. Meng, J. Bi, H. Wang, C. Sun, and H. Hu, "CoCo: Compact and Optimized Consolidation of Modularized Service Function Chains in NFV," in *Proceedings of 2018 IEEE International Conference on Communications (ICC)*, pp. 1–7, May 2018.