

Demonstration of Network Service Header Based Service Function Chain Application with Function Allocation Model

Rui Kang, Fujun He, Takehiro Sato, and Eiji Oki
Graduate School of Informatics, Kyoto University, Kyoto, Japan

Abstract—A virtual network function allocation model to maximize continuous available time of service function chains was introduced in our previous work. The performance of this model needs to be evaluated on network devices. It is time-consuming and costly to deploy functions with real network devices. Existing simulation tools require powerful computation capability, which limits the usable cases. We implement a network service header based service function chain application which can be cooperated with the model. Demonstration validates that the application allocates functions by using the allocation from the model automatically and runs service function chains correctly.

I. INTRODUCTION

Service functions are widely deployed and used in many networks. Service function chain (SFC) is an ordered set of service functions, which directs the service traffic through these functions in order [1]. In network virtualization, virtual network functions (VNFs) are deployed on physical machines to act as these service functions.

Network service header (NSH) is a protocol described in [2]. NSH is the SFC encapsulation. It is designed to identify the service function path and the location of current function in an SFC. It can be easily implemented in devices.

A VNF allocation model is designed to obtain a possible allocation strategy for VNFs in SFCs. In [3], an allocation model to maximize SFC continuous available time was introduced. Its performance in the network needs to be evaluated with different parameters of network elements in addition to the mathematical results. It is not cost effective and convenient to perform the evaluation with real network devices. One possible solution is to simulate the whole process by software on a single computer. Computational capability required for this software should not be so high. The simulation can run at a low-end computer, such as a laptop. The result of the model is obtained by a mathematical programming approach or a heuristic algorithm. Both of them can run by writing a programming language such as Python. The functions in SFCs need to be allocated according to the computed result in a single application.

This paper reports an implementation of the NSH-based SFC application with the cooperation of the VNF allocation model in [3]. We implement an NSH-based SFC application on Ryu software-defined networking framework [4]. We implement the functions of classifier, service function forwarder, and SFC proxy [1] on switches by the modification of flow tables which is conducted by the application. We use the application to simulate the VNF allocation and the traffic through these

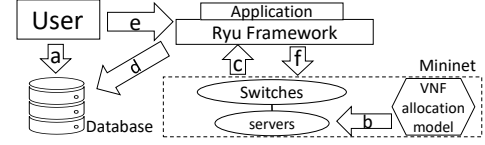


Fig. 1. Overall structure.

functions. The network devices are simulated in Mininet [5], which is connected to the application as a controller. It receives the registration message from allocated VNFs and instructs the actions of switches when they are necessary. We observe that the application allocates the VNFs to corresponding servers automatically and the path of each SFC is configured correctly.

II. IMPLEMENTATION

A. Overall structure

Fig. 1 overviews the structure of the reported application. A user registers the needed SFCs to database, as indicated by arrow *a*. The VNF allocation model with requested objective is triggered automatically or manually and sends the result to corresponding servers, as indicated by arrow *b*. The servers which receive messages from the allocation model change themselves to the corresponding functions and report their locations to the controller, as indicated by arrow *c*. When the controller receives these reports, it stores the information to database, as indicated by arrow *d*. When the user wants to start a service between two specific nodes, the description of the service is sent to the controller, as indicated by arrow *e*. The controller modifies the flow table in the switches when the service traffic goes through, as indicated by arrow *f*.

B. Database structure

There are three tables in the database, which are named *flow*, *service*, and *vnf*.

In table *flow*, a user defines the filter rules including Internet protocol (IP) and medium access control (MAC) addresses, ports of transmission control protocol (TCP) and user datagram protocol (UDP), type of protocols, and input port to decide if a flow belongs to a service whose identifier (ID) is “Service_ID”. A user uses “ID” to identify a flow and start or delete a flow. The application uses this value to match the detail of the service in table *service*.

Table *vnf* stores the register information from VNF. “ID” is the identification number for each VNF; “name” is a human-readable nickname for the VNF; “dpid” is the datapath ID of connected switch of the VNF; “in_port” is the port of

the switch, which connects to the VNF; “Mac_addr” is the Ethernet address of the VNF output port.

Table *service* describes the order of VNFs in each SFC given by users. Each VNF in SFC has a record. “Service_ID” shows the location of the VNF and is used in NSH.

C. Communication between each components

1) *Allocation results distribution and function register*: A program runs in each server for listening the configuration message from the allocation model. When the allocation results are computed, service ID, name, VNF ID, and other properties of each VNF are sent to the corresponding server. If the listening program receives the message, it chooses and starts a VNF according to the message and sends a register message to the controller through OpenFlow Packet_in message [6] which includes the information of the VNF.

2) *Interaction between controller and switches*: We use four tables in each switch named tables 0, 1, 2, and 3. In tables 0 and 1, we set a flow entry which lets all packages go to table 3 with priority 0. In table 2, we set a flow entry which lets all packages be sent to controller with priority 0. In table 3, we set a flow entry which lets all packages to be forwarded normally.

The OpenFlow connection is established when a switch is connected to controller. A new flow entry with priority 10 in table 0 is added to the new switch. In this flow entry, it matches the VNF register message with preset IP and MAC addresses. The action is to output the message to controller.

When a user starts a new service flow, a flow entry is configured to every switch. It matches the filter conditions of this flow and sends a message with the service ID, input port and the datapath ID to controller. It deletes all preset flow entries related to this flow.

When the controller receives the packages from this service, it looks up table *service* and finds the VNFs which belongs to this service by service ID. For the switch which receives the first frame of the flow, the controller asks the switch to encapsulate the Ethernet frame with NSH. In NSH, service path identification (SPI) is set to service ID, and service index (SI) is set to the length of SFC. An Ethernet frame whose destination is the address of the next VNF is encapsulated out of NSH. For the switch which is connected by the last VNF, it decapsulates NSH. For the other switches which are connected by VNFs, they decapsulate NSH and send packages to VNF. When they receive the packages from VNF, they encapsulate NSH with the same SPI and decrease SI by 1. An Ethernet frame whose destination is the address of the next VNF is encapsulated out of NSH.

III. DEMONSTRATION

We use Ryu 4.32 (added support for encap() and decap()), Mininet 2.3.0d6, Open vSwitch 2.12.0, Python 3.6.9, and Sqlite 2.8.17 for the demonstration. We connect the devices as shown in Fig. 2 in Mininet and enable the IP forwarding function for each service node. We run allocation model on node N1 simulated by Mininet. The objective of cooperated

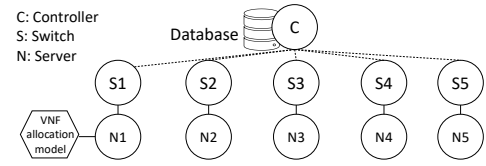


Fig. 2. Device connection diagram of demonstration.

id	name	dpid	in_port	mac_addr
Filter	Filter	Filter	Filter	Filter
1	forward1	1	1	00:00:00:00:00:01
2	forward2	2	1	00:00:00:00:00:02
3	forward3	4	1	00:00:00:00:00:04
4	forward4	5	1	00:00:00:00:00:05
5	forward5	3	1	00:00:00:00:00:03

Fig. 3. Register information in database.

```

20 22.4129/0120 10.0.0.4 10.0.0.5 UDP 1452 40000 - 44446 Len=1372
4
> Frame 28: 1452 bytes on wire (11616 bits), 1452 bytes captured (11616 bits) on interface 0
> Ethernet II, Src: 00:00:00:00:00:04 (00:00:00:00:00:04), Dst: 00:00:00:00:00:01 (00:00:00:00:00:01)
> User Datagram Protocol, Src Port: 40000, Dst Port: 44446
> Data (1372 bytes)

```

Fig. 4. Ethernet frame is encapsulated by NSH and another Ethernet frame.

```

mininet> h4 tracepath h5 -l 1400
1: 10.0.0.1 2.139ms
2: 10.0.0.2 1.422ms asymm 1
3: 10.0.0.5 2.177ms reached
Resume: pmtu 1400 hops 3 back 1

```

(a) Before allocation.

```

mininet> h4 tracepath h5 -l 1400
1: 10.0.0.5 1.685ms reached
Resume: pmtu 1400 hops 1 back 1

```

(b) After allocation.

Fig. 5. Result of tracepath.

allocation model is to maximize the continuous available time of SFCs [3]. We observe that the VNFs register themselves to database successfully as shown in Fig. 3. We use the tracepath command in Mininet to see the path of this flow. We observe that the data is successfully encapsulated by NSH as shown in Fig. 4. We observe that the path of a two-function chain is correctly configured from the result shown in Fig. 5(a) compared with the old path shown in Fig. 5(b).

ACKNOWLEDGMENT

This work was supported in part by JSPS KAKENHI, Japan, under Grant Number 18H03230.

REFERENCES

- [1] J. Halpern and C. Pignataro, “Service function chaining (SFC) architecture,” RFC 7665, Oct. 2015.
- [2] P. Quinn, U. Elzur, and C. Pignataro, “Network service header (NSH),” RFC 8300, Jan. 2018.
- [3] R. Kang, F. He, T. Sato, and E. Oki, “Virtual network function allocation to maximize continuous available time of service function chains,” in *2019 IEEE 8th International Conference on Cloud Networking (CloudNet)*, Nov 2019, pp. 1–6.
- [4] Ryu SDN Framework Community, “Ryu SDN framework,” <https://osrg.github.io/ryu/index.html>, accessed Jan. 21, 2020.
- [5] Mininet Team, “Mininet: An instant virtual network on your laptop (or other pc),” mininet.org/, accessed Jan. 21, 2020.
- [6] Open Networking Foundation, “OpenFlow switch specification version 1.3.0 (wire protocol 0x04),” mininet.org/, Jun. 2012, accessed Jan. 21, 2020.