A Joint Network and Server Load Balancing Algorithm for Chaining Virtualized Network Functions

Minh-Tuan Thai[1], Ying-Dar Lin[1], Yuan-Cheng Lai[2]
[1]National Chiao Tung University, Hsinchu, Taiwan
[2]National Taiwan University of Science and Technology, Taipei, Taiwan
Email: tmtuan.eed03g@nctu.edu.tw, ydlin@cs.nctu.edu.tw, laiyc@cs.ntust.edu.tw

***Abstract*— Chaining virtualized network functions (VNF) is an effective practice to deploy network services in network operator's data centers. Two common concerns arise in such a deployment are network load balancing and server load balancing. In this study, motivated by the argument that such two concerns should be jointly addressed for efficiently chaining VNFs in a data center environment, we propose a 2-phase algorithm, Nearest First and Local-Global Transformation (NF-LGT), which concurrently supports network and service load balancing. The algorithm firstly constructs service chains by a greedy strategy which both considers network latency and server latency. Then a searching technique is applied to improve the solutions. We have implemented the algorithm using Software-defined networking (SDN)/OpenFlow concept. The experimental results indicate that, compared with a sequential approach, NF-LGT improves the system bandwidth utilization up to 45%.**

***Keywords*— Network function virtualization, software defined networking, service chaining, quality of services.**

I. INTRODUCTION

Network function virtualization (NFV) [1] concept was recently introduced as a new manner to deliver network services whereby virtualized network functions (VNFs) substitutes hardware-based network appliances. By leveraging virtualization and cloud technology, NFV enables network function deployment in network operator's data centers, with great flexibility in deployment and management. Therefore, customers can make use of the network services on pay-per-use and on-demand basic, hence avoiding the need to acquire, install and maintain special equipment.

Thanks to NFV paradigm, network service operators can supply multiple network services whereby each service is provided through a service chain [2], which is a pre-defined interconnection of VNFs. Flows (i.e., service requests) have to be steered across needed VNFs while skipping unnecessary ones. It should be noted that there may be precedence requirements among those VNFs. For example, packet inspection functions such as firewall, IDS, and IPS are required to perform before traffic optimization or protocol proxy functions, for instance, video transcoding, NAT and HTTP proxy/cache.

Fig. 1 provides an illustration of service chaining in a data center environment, where VNFs are deployed in virtual machines that run on physical machines. The physical machines are connected by Fat-Tree network topology [3] which is a common data center topology providing multi-path capacity. In order to accommodate service requests, network operators have to select VNFs among candidates to form service chains, then force the requests to travel through the VNFs in desired order.

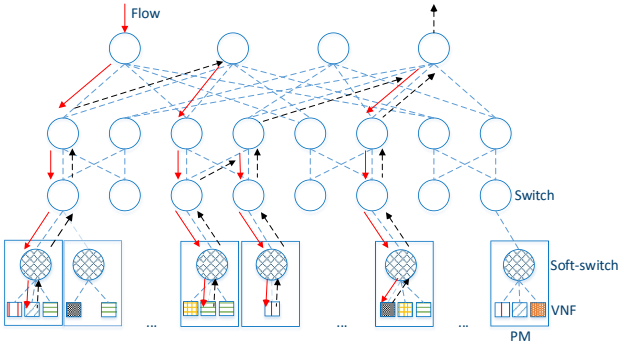


Fig. 1. Example of service chaining in a data center environment. The red solid lines represent downward traffic while black dash lines represent upward traffic.

Obviously, the number of possible chains is very large and increases exponentially with the number of required VNFs.

There are two important concerns, namely *server (i.e., VNF) load balancing,* and *network load balancing*, for instantiating service chains in a data center environment. The first concern is from the VNF multi-instance deploying model. In other words, one type of VNF has more than one instance which resides in different network locations. The second concern is from the multi-path capacity of data center network topology. That is, flows transmitting between VNFs can go through multiple paths. It is well-known fact that such concerns have to be tackled jointly for enabling an efficient NFV environment. To illustrate this point, one can consider a situation such that a flow may be assigned to overloaded VNFs, resulting in server congestion, if service chains are built with the goal only of network load balancing. Similarly, network congestion possibly happens if server load balancing is individually solved. A sequential approach can be applied for tackling these two concerns as well. That is, after balancing the load of VNFs, we balance the load of links. However, such an approach may just provide a suboptimal load balancing ability since it does not consider all possible combinations between VNFs and links when constructing service chains.

Both of server and network quality of services (QoS) issues have been intensively addressed in recent service chaining studies [4, 5, 6]. StEERING [4] proposes a greedy algorithm to solve service placement problem whose objective is to minimize network latency. However, the server QoS issue is ignored in this work. In contrast with StEERING, an online server load balancing problem, which is formulated in a linear program, is solved without the consideration of network QoS in SIMPLE

978-1-4799-6664-6/16/$31.00 ©2016 IEEE

[5]. The issues are jointly addressed by Stratos [6] whereby network traffic is split across network function instances according to their network latency. However, since the solution distributes packets of a flow among multi-paths, the packets may experience different latency which introduces a packet reordering problem, i.e. packet out-of-order arrivals, resulting in low throughput.

In this paper, aiming at providing both server load balancing and network load balancing for deploying service chains in a data center environment, we design an algorithm, called *Nearest First and Local-Global Transformation (NF-LGT)*, is composed of two phases. In the 1st phase, a service chain is constructed by a greedy strategy whereby the next VNF is selected among candidates according to its latency from the current location. The latency consists of network latency and server latency, which are calculated using the current packet arrival rate to links and VNFs. Consequently, network and server load balancing issues are jointly tackled in the algorithm. In the 2nd phase, a searching technique is applied to improve the solution of the 1st phase. The technique attempts to find a better service chain (i.e., smaller latency) by replacing selected VNF with another candidate and swapping the order of VNFs in service chains.

We have implemented our design as a module based on SDN/OpenFlow approach [7] decoupling control plane and data plane. The module, running atop an SDN controller, *(i)* monitors network topology, link and VNF status, *(ii)* computes service chains, and *(iii)* generates and inserts forwarding rules to OpenFlow switches which steer request traffic across VNFs.

Our proposed algorithm is evaluated using Mininet network emulator [8]. We compare the performance of NF-LGT to *Least Load First and Shortest Path First (LLF-SPF)* algorithm which sequentially solves server and network load balancing issues. The experimental results show that NF-LGT works efficiently. It can considerably improve the system bandwidth utilization while consume an acceptable amount of time.

The rest of this paper is organized as follows. In Section 2, we introduce the formal description of our service chaining problem. In Section 3, we elaborate our proposed algorithm, NF-LGT, and its implementation. Evaluation study and experimental results are presented in Section 4. Finally, Section 5 concludes this paper with a brief discussion on future work.

II. PROBLEM DESCRIPTION

This section first formally describes important terminologies in this study. Then, the problem statement is introduced. Fig. 2 provides an example of notation usage, and Table I summarizes notations used in our service chaining problem. It should be noted that we assume that the problem follows M/M/1 queuing model in this study.

A. Terminologies

1) Virtualized network function (VNF): In our problem, a VNF is responsible for handling a specific network function running in a virtual machine. Let $F = \{f_{i,j}, 0 \le j \le M_i\}$ present the set of VNFs of the system, where $f_{i,j}$ is jth instance of type-i network function and M_i is the number of instances of type-i network function. Each VNF $f_{i,j}$ has a latency $lt(f_{i,j})$ defined as

$$lt(f_{i,j}) = \frac{1}{C(f_{i,j}) - pa(f_{i,j})}, \quad (1)$$

where $C(f_{i,j})$ and $pa(f_{i,j})$ are the maximum bandwidth of $f_{i,j}$ and the current packet arrival rate to $f_{i,j}$, respectively.

2) Network topology: Let a directed graph $G = (F \cup S, EF \cup ES)$ denote the data center network topology where VNFs are deployed, with node set is $F \cup S$, and edge set is $EF \cup ES$. The set $S = \{s_k, 0 \le k \le |S|\}$ represents the switches of the network, where s_k is the kth switch. The set $EF = \{ef_{i,j}^k\}$ denotes the links between switches and VNFs, where $ef_{i,j}^k$ is the link between switch s_k and VNF $f_{i,j}$. The set $ES = \{es_k^{k'}\}$ denotes the links between switches, where $es_k^{k'}$ is the link between the switches s_k and $s_{k'}$. Links $ef_{i,j}^k$ and $es_k^{k'}$ have latency $lt(ef_{i,j}^k)$ and $lt(es_{k'}^k)$ respectively defined as

$$lt(ef_{i,j}^k) = \frac{1}{C(ef_{i,j}^k) - pa(ef_{i,j}^k)}, \text{ and } lt(es_{k'}^k) = \frac{1}{C(es_{k'}^k) - pa(es_{k'}^k)}, \quad (2)$$

where $C(ef_{i,j}^k)$ and $C(es_{k'}^k)$ are the maximum bandwidth, and $pa(ef_{i,j}^k)$ and $pa(es_{k'}^k)$ are the current packet arrival rate to $ef_{i,j}^k$ and $es_k^{k'}$.

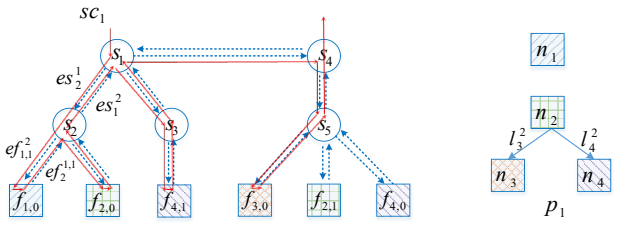
3) Request and Service policy: A network service request r_v is defined as a couple (b_v, p_v), where b_v is demanded bandwidth and $p_v =< N_v, L_v >$ is a directed acyclic graph representing the desired service policy for the request. The node set $N_v = \{n_i\}$ denotes network functions in which the request r_v must be processed. The link set $L_v = \{l_i^{i'}\}$ represents the precedence constraints among $n_i \in N_v$, where $l_i^{i'}$ is the precedence constraint from ith network function to i'th network function. In other words, the request r_v must be processed by type-i network function before type-i' network function.

B. Problem statement

Given a set of VNFs F, a network topology graph $G = (F \cup S, EF \cup ES)$, and a request $r_v\,(b_v, p_v)$. The objective of this work is to determining the service chain SC_v, which is an ordered set of links $ef_{i,j}^k \in EF, es_k^{k'} \in ES$ and VNFs $f_{i,j} \in F$ by which r_v need to go through such that its latency LT_v is minimized. The formula

$$LT_v = \sum_{ef_{i,j}^k,\ es_{k'}^k, f_{i,j} \in SC_v} \left(lt(ef_{i,j}^k) + lt(es_{k'}^k) + lt(f_{i,j})\right), \quad (3)$$

is used to calculate the latency of the service chain.



a) Network topology and service chain b) Service policy

Fig. 2. Example of notation usage.

TABLE I. NOTATIONS USED IN SERVICE CHAINING PROBLEM

Notations	*Meaning*
Virtualized network function	
$F = \{f_{i,j} \mid 0 \le j \le M_i\}$	The set of VNFs of the system, where $f_{i,j}$ is jth instance of type-i network function and M_i is the number of instances of type-i network function
$C(f_{i,j})$	The maximum bandwidth of $f_{i,j}$
$pa(f_{i,j})$	The current packet arrival rate to $f_{i,j}$
$lt(f_{i,j})$	The current latency of $f_{i,j}$
Network topology	
$S = \{s_k \mid 0 \le k \le \lvert S \rvert\}$	The set of $\lvert S \rvert$ switches in the system, where s_k is kth switch
$G < F \cup S, EF \cup ES >$	The network topology graph, where node set is $F \cup S$ and link set is $EF \cup ES$
$EF = \{ef_{i,j}^k\}$	The link set between VNFs and switches, where $ef_{i,j}^k$ is the link between s_k and $f_{i,j}$
$ES = \{es_{k'}^k\}$	The link set between switches, where $es_{k'}^k$ is the link between s_k and $s_{k'}$
$C(ef_{i,j}^k)$ and $C(es_{k'}^k)$	The maximum bandwidth of $ef_{i,j}^k$ and $es_{k'}^k$
$pa(ef_{i,j}^k)$ and $pa(es_{k'}^k)$	The current packet arrival rate to $ef_{i,j}^k$ and $es_{k'}^k$
$lt(ef_{i,j}^k)$ and $lt(es_{k'}^k)$	The current latency of $ef_{i,j}^k$ and $es_{k'}^k$
Request – Service policy	
$r_v(b_v, p_v)$	The vth network service request
b_v	The demanded bandwidth of r_v
$p_v < N_v, L_v >$	The directed acyclic graph represents the service policy required by vth request
$N_v = \{n_i\}$	The node set represents required network functions by vth request
$L_v = \{l_i^{i'}\}$	The link set represents the precedence constraints among network functions $n_i \in N_v$; where $l_i^{i'}$ is precedence constraint from type-i network function to type-i network function
SC_v	The service chain for r_v, which is ordered set of links $ef_{i,j}^k \in EF$, $es_k^{k'} \in ES$ and VNFs $f_{i,j} \in F$ by which r_v need to go through
LT_v	The latency of SC_v, $LT_v = \sum_{ef_{i,j}^k,\ es_{k'}^k, f_{i,j} \in SC_v} (lt(ef_{i,j}^k) + lt(es_{k'}^k) + lt(f_{i,j}))$

The objective of our service chaining problem is finding a service chain whose latency is smallest among candidates which can be solved by an exhaustive searching algorithm. However, such an approach is impractical due to the huge numbers of links and VNFs; and the time-sensitive requirement of NFV environment. As a result, an efficient approximation algorithm is needed.

III. NEAREST FIRST AND LOCAL-GLOBAL TRANSFORMATION

In this section, we describe our proposed algorithm for solving the service chaining problem defined in Section 2. We first give an overview of 2-phase service chaining algorithm with the Nearest First phase and the Local-Global Transformation phase. The details of our approach are then elaborated in the subsequent subsections. Finally, we introduce the implementation of our design using SDN/OpenFlow approach.

A. Overview

The key argument of this work is that we should jointly address server and network load balancing issues in chaining VNFs. This requirement is satisfied by our design, which concurrently considers the current load of link and VNF in building service chains. The latency of the service chains, calculated as in (3), includes link and VNF latencies which are calculated based on their current packet arrival rate using (1) and (2). By doing so, server load balancing and network load balancing are jointly supported by our algorithm.

To solve the problem, we introduce a 2-phase algorithm called *Nearest First and Local-Global Transformation (NF-LGT)*. In the *Nearest First* phase, we construct an initial service chain for a submitted request using a greedy strategy. In the strategy, we choose the VNF, whose latency from the current location is smallest, to be the next destination. We do this iteratively until all required network functions are composed to the service chain. Since the greedy strategy may yield a local optimal solution, a searching technique is applied in the *Local-Global Transformation* phase to improve the solution. The technique consists of two methods namely *Local Transformation* and *Global Transformation* which aim at finding a smaller latency service chain among feasible solutions. During Local Transformation, a new feasible service chain is constructed by replacing a current selected VNF with another same type candidate, while the order of two VNFs, which have no precedence constraints, are swapped to generate a new feasible chain in Global Transformation.

B. Nearest First phase

Algorithm 1 shows the pseudo code of the Nearest First phase which constructs a service chain SC_v by a greedy approach. At the first step, it initializes the ingress switch as the current location (line 2). Next, all available VNFs, which have no any precedent requirements, are discovered (line 4). After that, the smallest-latency path sp, which is from the current location to an available VNF, $f_{i,j} \in AF$, is selected among all feasible paths FP to construct the service chain SC_v (line 6 and 7). Finally, since network function n_i is already included to SC_v, we remove it from N_v and assign $f_{i,j}$ as current location (line 8 and 9). Such steps are repeated until all required network functions $n_i \in N_v$ are included in the service chain SC_v (line 3).

Algorithm 1 Nearest First (NF) phase

Input: $G < F \cup S, EF \cup ES >$, $r_v(b_v, p_v)$
Output: SC_v
1: **Begin**
2: $currentLocation \leftarrow Ingress\ switch$
3: **While** ($N_v \neq \emptyset$)
Find all available VNFs
4: $AF \leftarrow f_{i,j}, \forall f_{i,j} \in F \mid \nexists l_i^{i'} \in L_v$
Find all feasible paths
5: $FP \leftarrow feasiblePath(currentLocation, f_{i,j}, b_v)\ \forall f_{i,j} \in AF$
Find smallest latency path
6: $sp \leftarrow smallestLatency(FP)$
7: $SC_v \leftarrow SC_v \cup sp$
8: $N_v \leftarrow N_v - n_i$
9: $currentLocation \leftarrow f_{i,j}$
10: **End-While**
11: **End**

C. Local-Global Transformation phase

In this phase, we attempt to improve the service chain SC_v found in the Nearest First phase. In other words, we try to find a new service chain SC'_v such that $LT'_v < LT_v$. The pseudo code of the Local-Global Transformation phase is shown in Algorithm 2. At the first step, we assign SC_v to the current accepted solution SC'_v (line 2). After that, SC_v is transformed to construct new feasible service chains during Local and Global Transformation methods. In Local Transformation, a current selected VNF $f_{i,j} \in SC_v$ is substituted by a same type $f_{i,j'}$ to form a new feasible service chain (line 6 and 7). If the latency LT_v of the new chain SC_v is smaller than LT'_v of current solution SC'_v, it becomes SC'_v (line 8 and 9). In Global Transformation, we interchange the positions of VNFs $f_{i,j}$ and $f_{i',j}$, which are currently in SC_v, to form a new feasible chain (line 12 and 13). It should be noted that such VNFs have no any precedent requirements (line 3).

Algorithm 2 Local-Global Transformation (LGT) phase

Input: $G < F \cup S, EF \cup ES >, r_v(b_v, p_v), SC_v$
Output: SC'_v
1: **Begin**
2: $SC'_v \leftarrow SC_v$
3: **For each** pairs $(n_i, n_{i'}) \in N_v | \nexists\, l_i^{i'}, l_{i'}^{i} \in L_v$
Local transformation
4: **For each** $n_i \in N_v \mid M_i > 1$ **do**
5: Let $f_{i,j} \in SC_v$ be the current selected VNF for n_i
6: Replace $f_{i,j}$ in SC_v with a $f_{i,j'}, \forall f_{i,j'} \in n_i$
7: $SC_v \leftarrow update(SC_v)$
8: **If** $(LT_v < LT'_v)$ # find a better than SC_v
9: $SC'_v \leftarrow SC_v$
10: **End-If**
11: **End-For**
Global transformation
12: Swap the order of $f_{i,j}$ and $f_{i',j}$ in SC_v
13: $SC_v \leftarrow update(SC_v)$
14: **End-For**
15: **End**

D. Example run for NF-LGT

An example running for NF-LGT is illustrated in Fig. 3. Note that although there are multiple paths between two VNFs, only the smallest latency path is shown for simplicity.

In Fig. 3a, an initial service chain with the latency $LT_v = 15.0$ is constructed by the Nearest First phase. From Fig. 3b to Fig. 3d, we attempt to find a better solution by the Local-Global Transformation phase. To be more specific, an accepted solution with $LT_v = 14.0$ is found by applying the Local transformation method, i.e., replacing $f_{2,1}$ with $f_{2,0}$, in Fig. 3b. Unfortunately, the method generates a failed solution showed in Fig. 3c. The latency of the new service chain is 17.0, which is greater than the current value. As can be seen from Fig. 3d, the Global transformation method discovers the final solution with $LT_v = 13.0$ by swapping $f_{3,0}$ and $f_{4,1}$.

E. SDN/OpenFlow-based implementation

To validate our design, we have implemented the proposed algorithm NF-LGT as a module running atop of a SDN controller. As outlined in Fig. 4, the module communicates with the controller through provided Restful API. In more details, the

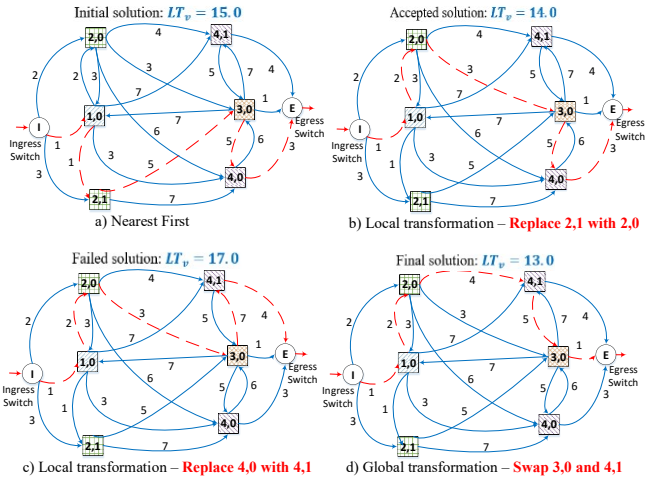


Fig. 3. An example run for NF-LGT. A blue solid line represents the smallest-latency path from a VNF to another while the red dash lines represent a feasible service chain.

Network Monitor component maintains current network topology, the status of links and VNFs by periodically sending query commands to the controller. The Service Chain Builder component, where NF-LGT algorithm is implemented, calculates service chains with network status and submitted requests as the inputs. Subsequently, corresponding flow entries are generated by the Forwarding Rule Generator component for such service chains. Then, the entries are sent to the controller via Restful API. The controller's service abstraction layer will, in turn, insert the flow entries to switches in data plane using OpenFlow plugin.

1) Link and VNF latency calculation: the NF-LGT algorithm constructs service chains using the current latency of links and VNFs. Unfortunately, it is non-trivial to measure such parameters directly both from switches or controller. In our implementation, we apply port statistic monitoring technique to estimate these parameters. That is, the Network Monitor component periodically queries the number of transmitted bytes from every switch port. Then, it calculates the increment from the last monitoring time and estimates the current packet arrival rate of every link and VNF. It is assumed that link and VNF maximum bandwidth is determined in advance, one can easily

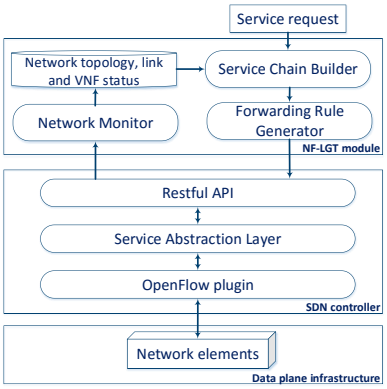


Fig. 4. SDN/OpenFlow-based implementation for NF-LGT.

compute the current latency of links and VNFs using (1) and (2) respectively.

2) Packet steering: In an NFV environment, network traffic needs to be steered through required VNFs in desired order. Since a packet may traverse a switch multiple times, and it should be treated differently each time. Hence, switches must know the status of a packet, i.e., where the packet is currently in its service chain, in order to determine processing actions for the packet. In this work, we propose *Status tag* for encoding which network functions have already been traversed in a service chain. As shown in Fig. 5, the *i*th bit in the tag represents the type-*i* network function. The tag with an initial value is added to a packet at ingress switches and removed at egress switches. At soft switches where VNFs are connected, if a packet has been processed by a VNF, which is easily determined by the arriving port of the packet, the corresponding bit is set to 1. VLAN tag, MPLS label or unused IP header fields could be borrowed to embed the tag to packets depending the support of OpenFlow switch.

IV. Performace Evaluation

This section presents the experimental evaluation where the effectiveness of proposed algorithm is verified.

A. Experimental setup

The performance of our NF-LGT algorithm is evaluated in a 4-ary FatTree network topology which is simulated by Mininet network emulator. We use Open vSwitch [9] to simulate a number of VNFs in the network, which is relevant since they are both packet processing nodes. The VNFs are randomly located over the network. OpenDayLight Helium-SR3 [10], whose monitoring cycle is set to 3 seconds, is deployed as the SDN controller.

Network service requests are UDP flows generated by iPerf [11]. In our experiments, we vary their offered load in the range [1%, 5%] of the overall link and VNF bandwidth. The inter-arrival time and duration of the flows are exponentially distributed with an adjustable mean value in order to control the number of concurrent flows in the system. By doing so, we can observe the performance of the proposed algorithm under different system loads. Each request is processed by a service policy whose the number of network functions is uniformly generated in the range [1, 5], and their precedence constraints are randomly created.

The performance of proposed algorithm is measured by *bandwidth utilization* criteria which is the ratio of achieved bandwidth to designed bandwidth of flows. The achieved bandwidth is real bandwidth which a flow can achieve when traverses through a service chain. The designed bandwidth is the bandwidth which iPerf client tool assigns for the flow. The ratio is in the range [0.0, 1.0], and the higher ratio is, the better throughput the system can get [12].

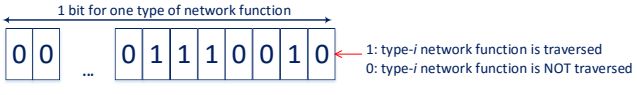


Fig. 5. Status tag for encoding packet status.

All the experiment results shown here are obtained by averaging the results of five simulation runs. Each run is terminated upon the successful completion of 1000 requests.

B. Result Analysis

1) Jointly vs. Sequentially: In this experiment, we compare NF, the 1st phase of our proposed algorithm, to LLF-SPF. In contrast to our design, LLF-SPF addresses server and network balancing issues sequentially. The algorithm constructs a service chain by firstly selecting the smallest load VNF among feasible candidates to be the next destination. Then, the path to the destination whose network latency is smallest is chosen to construct the service chain. The steps are repeated until all required network functions added to the service chain.

Fig. 6 shows that our design outperforms LLF-SPF at every traffic load value. For example, for the case where the number of concurrent flows is set to 200, the performance difference between NF and LLF-SPF is approximately 11%. This observed phenomenon can be explained by the chaining approach of LLF-SPF in which it chooses the smallest load VNF to be the next destination. However, it is possible that there are not any feasible paths to reach the VNF; as a result, network congestions would happen. On the contrary, NF considers concurrently VNF and link status for constructing service chains. Consequently, it can provide better performance than LLF-SPF.

The service chaining time of different algorithms is compared in Fig. 7. The experiment results show that NF and LLF-SPT consume almost the same amount of time for constructing service chains. Surprisingly, the service chain calculation time only makes up a small proportion of total chaining time. Take LLF-SPF algorithm, for example, the proportion is 18%.

2) 2 phases vs. 1 phase: In this investigation, we measure and observe how well LGT, the 2nd phase of our proposed algorithm, improves the solutions calculated by the 1st phase, NF.

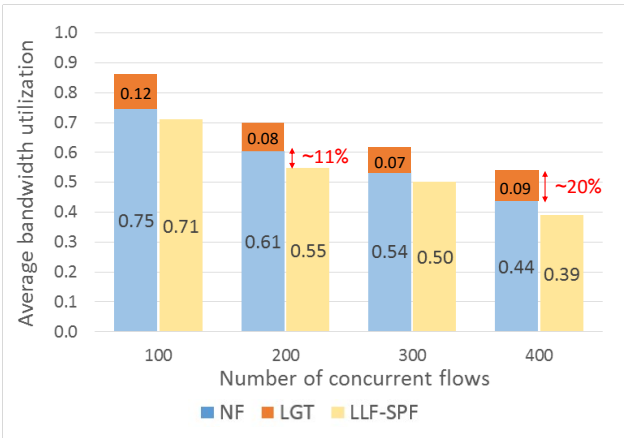


Fig. 6. Bandwidth utilization of different algorithms over traffic load.

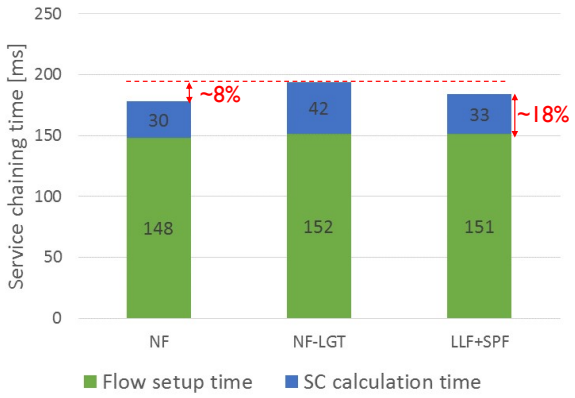


Fig. 7. Service chaining time of different algorithms.

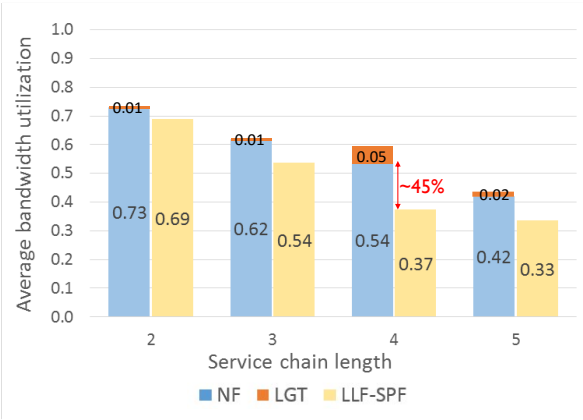


Fig. 8. Bandwidth utilization of different algorithms over service chain length.

The experiment results clearly indicate that the average bandwidth utilization is considerably improved by LGT phase. As shown in Fig. 6, the improvement varies between 20% and 12% at different numbers of concurrent flows. For the service chaining time, Fig. 7 presents that LGT phase increases the time about 8%. To sum up, it is worth applying the 2nd phase LGT in the proposed algorithm NF-LGT since the phase could noticeably enhance the algorithm while consume an acceptable amount of time.

3) The impact of service chain length: We conduct another experiment in which the number of concurrent flows is set 200, and the number of VNFs in service policies is varied in the range [2,5]. By doing so, the performance of different algorithms is investigated under various values of service chain length in this section.

As expected, Fig. 8 clearly demonstrates that the length of service chain significantly affects the performance of observed algorithms. Take NF for example, as the length increases from 2 to 5, the average bandwidth utilization decreases 0.73 to 0.42. The experiment results also indicate that our design is better LLF-SPF at any values of service chain length. Up to the length of 4, the longer length is, the better performance NF can get. At this value of service chain length, the performance difference between NF and LLF-SPF is 45% improvement in bandwidth utilization.

V. Conclusion

Motivated by the argument that network load balancing and server load balancing should be jointly addressed for efficiently enabling NFV in data center environment, we have presented a service chaining algorithm, NF-LGT. At the 1st phase of the algorithm, service chaining decisions are made by a greedy approach which considers concurrently server and network latencies. Then a searching technique is introduced to improve the decisions at the 2nd phase. We have implemented the algorithm using SDN/OpenFlow concept with three main functions such as network monitoring, service chain calculation, and forwarding rule generation.

The proposed algorithm has been experimentally evaluated under different scenarios by Mininet network emulator. The experimental results indicate that our algorithm outperforms the comparison algorithm which sequentially solves network and server load balancing issues. The results also show that it is worth applying the 2nd phase of our algorithm since it considerably improves the system throughput.

References

[1] Network Functions Virtualisation – Introductory White Paper. [Online]. Available at: https://portal.etsi.org/Portals/0/TBpages/NFV/Docs/NFV_White_Paper3.pdf

[2] W. John, K. Pentikousis, G. Agapiou, E. Jacob, M. Kind, A. Manzalini, F. Risso, D. Staessens, R. Steinert, and C. Meirosu, "Research Directions in Network Service Chaining," in *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*, 2013, pp. 1–7.

[3] M. Al-Fares, A. Loukissas, and A. Vahdat, "A Scalable, Commodity Data Center Network Architecture," in *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, New York, NY, USA, 2008, pp. 63–74

[4] Y. Zhang, N. Beheshti, L. Beliveau, G. Lefebvre, R. Manghirmalani, R. Mishra, R. Patneyt, M. Shirazipour, R. Subrahmaniam, C. Truchan, and M. Tatipamula, "StEERING: A software-defined networking for inline service chaining," in *2013 21st IEEE International Conference on Network Protocols (ICNP)*, 2013, pp. 1–10.

[5] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "SIMPLE-fying Middlebox Policy Enforcement Using SDN," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, New York, NY, USA, 2013, pp. 27–38.

[6] A. Gember, A. Krishnamurthy, S. S. John, R. Grandl, X. Gao, A. Anand, T. Benson, A. Akella, and V. Sekar, "Stratos: A Network-Aware Orchestration Layer for Middleboxes in the Cloud," May 2013.

[7] D. Kreutz, F. M. V. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[8] "Mininet". Available at: http://mininet.org/

[9] "Open vSwitch". Available at: http://openvswitch.org/

[10] "OpenDayLight". Available at: https://www.opendaylight.org/

[11] "iPerf". Available at: https://iperf.fr/

[12] Li, Yu, and Deng Pan. "OpenFlow based load balancing for Fat-Tree networks with multipath support." in *Proc. 12th IEEE International Conference on Communications (ICC'13)*, Budapest, Hungary, pp. 1-5. 2013.