Hash-based Load Balanced Traffic Steering on Softswitches for Chaining Virtualized Network Functions

Minh-Tuan Thai[1], Ying-Dar Lin[1], Po-Ching Lin[2], Yuan-Cheng Lai[3]
[1]National Chiao Tung University, Hsinchu, Taiwan
[2]National Chung Cheng University, Chiayi, Taiwan
[3]National Taiwan University of Science and Technology, Taipei, Taiwan
Email: tmtuan.eed03g@nctu.edu.tw, ydlin@cs.nctu.edu.tw, pclin@cs.ccu.edu.tw, laiyc@cs.ntust.edu.tw

***Abstract*— Prior load balancing solutions for chaining virtualized network functions cause significant control and data plane overheads and demand special requirements on network hardware. In this study, we present the design, implementation, and evaluation of Hash-based Traffic Steering on Softswitches (HATS), a load balancing mechanism that aims at mitigating such drawbacks. The method exploits flow hashing technique implemented on softswitches to perform server and network load balancing without triggering the control plane. We have implemented this design using OpenDayLight controller and Open vSwitch platform. The implementation demonstrates that HATS can be readily implemented with commodity network hardware. Furthermore, the experiment results confirm that HATS can reduce the number of flow entries and service chaining time up to 85% and 93%, respectively, when compared with Least Load First (LLF), a controller-based service chaining algorithm.**

Keywords*— *Network function virtualization, service chaining, load balancing, flow hashing, software defined networking.

I. INTRODUCTION

The introduction of network function virtualization (NFV) [1] is promising to enable flexible deployment, management, and provision of networking services in service providers' data centers whereby hardware-based middleboxes are replaced with virtualized network functions (VNFs). Thanks to NFV paradigm, a composite network service can be provided through a service chain, which is a service policy defining a sequence of VNFs being applied to service requests (i.e., flows) [2]. For example, a network security service may require flows to pass through a firewall, and then a content filter before ending at an intrusion prevention system (IPS). To provision such services in a data center environment, network operators have to construct service paths, which are instances of service chains created using the overlay topology between VNFs. In other words, service paths define the interconnections of VNFs in an overlay network that flows have to be steered across.

Service path instantiation (i.e., service chaining) raises two significant concerns, namely VNF load balancing and network path load balancing [3]. The first concern is from the fact that there are multiple parallel instances of each VNF type for scale-out reasons. As a result, a VNF load balancing scheme has to be proposed to spread network traffic across those instances when building service paths. Additionally, due to the multi-path capacity of data center network topology, there should be a network load balancing solution for service path construction. Such two concerns have to be carefully addressed to ensure operational efficiency and system performance.

A variety of research efforts [3,4,5,6] has been proposed to meet the objectives of service chaining problems. Broadly speaking, almost all prior studies on service chaining are SDN-based solutions whereby proposed algorithms are mostly implemented in the control plane. That is, there is an SDN controller which is responsible for calculating service paths and then inserting forwarding rules to the data plane switches to steer flows across the required VNFs. Although these studies can satisfy the desired service requirements, they all have two important drawbacks. The first is control plane overhead, which means the controller must keep track of the status of VNFs and network to make service chaining decisions. It is a challenging task due to the huge numbers of VNFs and network paths. Furthermore, the controller needs to handle a remarkable number of simultaneous packet-in events sent from the data plane. The second drawback is data plane overhead, which is the enormous number of flow entries inserted into switches since each service path needs to be converted to a set of flow entries on all the relevant switches. Additionally, the flow setup time is significantly long, which probably downgrades the system performance. According to [3], the time is approximately 80% of the total service chaining time. To lower the control and data plane overhead, the authors in [7] introduced a hash-based method for achieving load balancing among service nodes in an NFV environment. However, network load balancing issue is ignored in that work, and that solution requires the support of OpenFlow group table on hardware switches.

In this work, we propose a load balancing system for chaining VNFs in a data center, called *Ha*sh-based *T*raffic *S*teering on *S*oftswitches (HATS), with the aim of mitigating the control and data plane overhead in existing service chaining solutions. The main idea of our solution is that a centralized controller is employed to collect the network topology and proactively disseminate corresponding load balancing information to edge softswitches (software-based switches), which steer network traffic to VNFs through multiple paths. The VNF and path selection to construct service paths are made by flow matching and hashing at softswitches instead of triggering the controller. This solution not only supports VNF and network load balancing but also significantly decreases system overhead. Additionally, there are not any special requirements to network hardware in our solution.

We have implemented our design using the OpenDayLight (ODL) controller [8] and Open vSwitch [9], whereby VNF and path selection are executed by the SELECT type of OpenFlow group table. The performance of HATS is evaluated using Mininet network emulator [10] with VNFs implemented by the

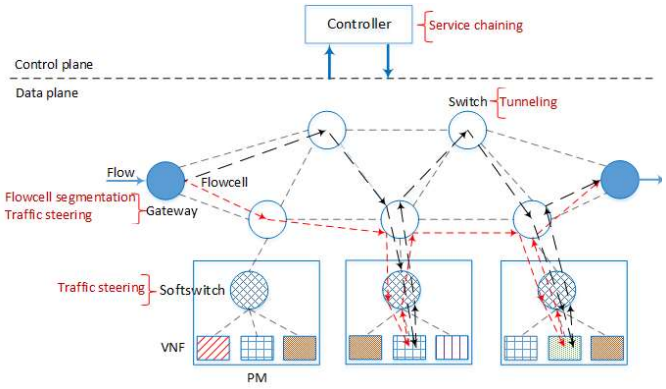


Fig. 1. The architecture of HATS.

Click elements [11]. The experiment results show that HATS can efficiently perform VNF and path load balancing while significantly lower the number of flow entries and service chaining time.

The rest of this paper is organized as follows. The service chaining problem is described formally in Section II. In Section III, we elaborate HATS and its implementation. Evaluation study and experimental results are presented in Section IV. Finally, Section V 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.

A. Terminologies

1) Virtualized network function (VNF): In service chaining, a VNF is a service node offering a network function responsible for a particular treatment of received packets. Let $N = \{n_i, 0 \leq i \leq |N|\}$ be the set of $|N|$ network functions provided by the system. The VNFs of the system are denoted by a set $F = \{f_{i,j}, 0 \leq j \leq M_i\}$, where $f_{i,j}$ and M_i are the jth instance and the number of instances of n_i. We use $V(f_{i,j})$ to indicate the volume of network traffic arrives at VNF $f_{i,j}$.

2) Network topology: Let $S = \{s_k, 0 \leq k \leq |S|\}$ be the set of $|S|$ softswitches in the network topology to which VNFs are connected. The softswitch-to-softswitch paths in the network is represented by a set $P = \{p^u_{k,k'}, 0 \leq k, k' \leq |S|\}$, where $p^u_{k,k'}$ is the uth path from softswitch s_k to softswitch $s_{k'}$. $V(p^u_{k,k'})$ denotes the volume of network traffic transmitted through the path $p^u_{k,k'}$. Further, let $L = \{l^k_{i,j}\}$ denote the locations of VNFs in the network, where the binary variable $l^k_{i,j} \in \{0,1\}$ indicates whether a VNF $f_{i,j} \in F$ is attached to a softswitch $s_k \in S$, where

$$l^k_{i,j} = \begin{cases} 1 & f_{i,j} \text{ is attached to } s_k, \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

3) Service request: Let $r_v(c_v) \in R$ be a service request arriving at the system, where c_v is the service chain for its processing. The service chain $c_v \in C$ is defined as a couple $(N'_v, \prec_v)$, where $N'_v \subseteq N$ is the set of network functions demanded by c_v and $\prec_v$ represents the sequential order of the functions $n_i \in N'_v$.

4) VNF and path load balancing: We define the VNF load balancing criterion of network function n_i as

$$VL_i = MAX_{f_{i,j} \in n_i}\left(\frac{V(f_{i,j})}{SUM_{f_{i,j} \in n_i}(V(f_{i,j}))}\right) * 100\%, \quad (2)$$

which is the maximum percentage of the volume of network traffic arriving at VNFs $f_{i,j} \in n_i$. Similarly, path load balancing criteria of the system is defined as

$$NL = MAX_{p^u_{k,k'} \in P}\left(\frac{V(p^u_{k,k'})}{SUM_{p^u_{k,k'} \in P}(V(p^u_{k,k'}))}\right) * 100\%, \quad (3)$$

which is the maximum percentage of the volume of network traffic transmitted through paths $p^u_{k,k'} \in P$.

5) Control and data plane overhead: Let E indicate the number of flow entries in the system and T be the average service chaining time for service requests.

B. Problem statement

Given a set of VNFs F, an overlay network topology $G(S, P, L)$, and a set of serivce requests R, the objective of this work is to design an efficient service chaining mechanism which not only provides server and network load balancing but also minimizes control and data plane overhead. In other words, we aim to minimize VL_i, NL and lower E, T at the same time when making service chaining decisions.

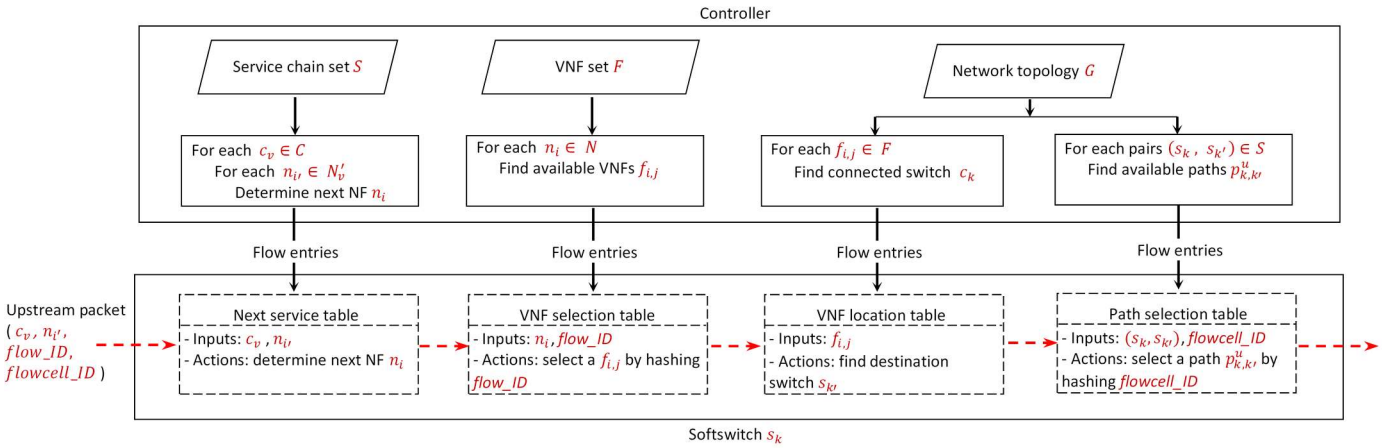
III. Hash-based Load Balanced Traffic Steering on Softswitches

In this section, we present the design of HATS for solving the service chaining problem defined in Section II. We first give the overview of this approach with the fundamental design ideas. The details are then elaborated in the subsequent subsections. Finally, we introduce the implementation of HATS using the ODL controller and Open vSwitch platform.

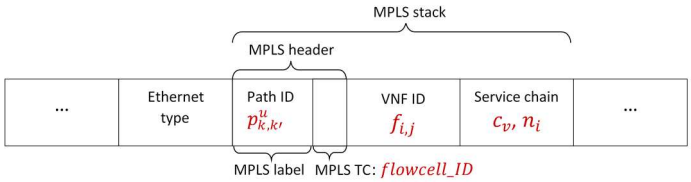
A. Approach Overview

Fig. 1 presents the overview of HATS wherein VNFs are deployed in virtual machines running on physical servers. The VNFs are connected to a data center network using softswitches. To accommodate service requests arriving at a gateway, the system has to make service chaining decisions that force the flows to travel through the VNFs in the desired order. We aim to address the VNF and network path load balancing issues while reducing the control and data plane overhead in service chaining. This requirement is satisfied by our design which consists of following fundamental design decisions:

1) Hash-based traffic steering on softswitches: Most literature does not differentiate service chaining and traffic steering. Here we define service chaining as finding the required network functions and the associated order that must apply to flows, and traffic steering as forwarding packets through VNFs. Chaining is thus to compute the service paths in



a) Detailed design of HATS



b) Packet format

Fig. 2. Controller-based service chaining and hash-based traffic steering on an edge softswitch.

the background periodically and store them in the flow tables which are looked up when steering packets.

Existing studies [3,4,5,6] only implement load balancing intelligence in service chaining algorithms, while in HATS, it resides in both service chaining and traffic steering procedures. The key idea is that the controller does not construct service paths with specific VNFs and paths, but just proactively disseminates possible candidates to the data plane components such as softswitches and gateways, which will select specific VNFs and paths by hash functions during traffic steering operations instead of triggering the controller. By doing so, HATS not only spreads flows to multiple VNFs and paths for load balancing but also reduces the number of packet-in events sent to the controller. Furthermore, HATS has no need of per-flow matching at switches; therefore, the number of flow entries and the waiting time for flow entry setup are remarkably lowered. It should be noted that the gateways in this architecture are responsible for steering network traffic to the first VNFs of service paths.

2) Per-softswitch load balancing: The hash-based VNF and network load balancing can be done at per-gateway, per-softswitch, and per-hop levels. In this study, we choose the second option in which the information of available VNFs is kept in softswitches. After a packet is processed by a VNF, it will be redirected to a softswitch responsible for determining next destination (i.e., a VNF) of the packet by flow hashing. Such steps are repeated until the packet travels through all required network functions and returns to a gateway. Similarly, pre-configured softswitch-to-softswitch paths are disseminated in softswitches, and network traffic is split onto these paths by flow hashing as well. The hardware switches in the network are only responsible for tunneling network traffic from a softswitch to another.

Even though hash-based load balancing can also be accomplished in per-gateway and per-hop fashions, we decide to implement it in softswitches for three reasons. The first reason is that per-gateway approach is inefficient due to the huge number of possible service paths in the network, while per-hop approach performs poorly under asymmetric topologies [12] and needs special requirements in hardware switches such as flow hashing and SDN support, which demand extra hardware cost and complicate system management. The second is that softswitches reside at a suitable location (i.e., right above VNFs) for load balancing tasks. They can easily modify packets without requiring any changes to VNFs or hardware switches. It is reasonable that next destination of a packet and the path to reach the destination are determined at a softswitch. Redirecting the packet to a gateway will raise scalability issues. Last but not least, softswitch platforms such as Open vSwitch have important functionalities like SDN-enable and OpenFlow group table, which are necessary to implement our design.

3) Flowcell-based multipathing

Hash-based multipathing has a major drawback, which is hash collision causing a few elephant (high bandwidth demand and long-lived) flows to share the same path, even though the other paths may be lightly loaded. This problem may significantly decrease system performance. To combat it, we follow the idea of flowcells [13], which are equal-size bursts of packets by breaking an elephant flow at a gateway. Then the flowcells are distributed over the paths like individual flows.

TABLE I. VNF AND PATH SELECTION TABLEs IMPLEMENTATION

	A group entry	A bucket	Hashed fields	Actions
VNF selection group table	- A network function n_i	- A VNF $f_{i,j} \in n_i$	- *flow_ID* and (c_v, n_i)	- Push MPLS label with VNF ID $f_{i,j}$
Path selection group table	- A pair of softswitches s_k and $s_{k'}$	- A path $p^u_{k,k'}$	- *flowcell_ID* and $f_{i,j}$	- Push MPLS label with path ID $p^u_{k,k'}$

The flowcell approach is promising to provide better load balancing, but since it spreads packets of the same flow among multipath, the packets may experience different latency, which introduces packet reordering. The problem must be carefully addressed to ensure the efficiency of any sub-flow load balancing schemes.

Note that the flowcells of the same flow cannot be processed by different VNFs of a stateful network function. In other words, the flowcells must travel through the same VNFs but via different paths in this design.

B. Detailed Design of HATS

Fig. 2 shows the details of the HATS design; wherein a centralized controller is employed to collect the information of service chains C, VNFs F, and network topology G. Then the controller generates traffic steering rules by inserting flow entries into the forwarding tables of a softswitch s_k, which involves *Next service table*, *VNF selection table*, *VNF location table* and *Path selection table*. As an upstream packet arrives at s_k, it is processed by the pipelined tables. Based on the packet information, such as the desired service chain c_v, the current visited network function $n_{i'}$, its *flow_ID* and *flowcell_ID*, the tables select next VNF $f_{i,j}$ and a path $p^u_{k,k'}$ to reach the VNF in a load balancing manner without triggering the controller.

The matching inputs and associated actions of the tables are shown in Fig. 2. Note that the tables are responsible for processing upstream packets. For downstream ones, the softswitch just decapsulates the packets and forwards them to the destinations, i.e., VNFs.

1) Next service table: The matching inputs are the required service chain c_v and the current visited network function $n_{i'}$. The action is to determine next network function n_i for incoming packets.

2) VNF selection table: Next required network function n_i is the macthing input. This table, which contains the information about available VNFs belonging to the funciton n_i, will select a particular VNF $f_{i,j}$ by hashing the *flow_ID* of a packet. The identifer of $f_{i,j}$ is then encapluted into the packet by adding an MPLS label. The *flow_ID* consists of the packet's fields such as etherType, src/dst MAC, src/dst IP and src/dst UDP/TCP ports, which identify the flow that the packet belongs to. The controller periodically maintains the VNFs set F, and then updates this table. Note that VNFs are dynamically created and destroyed over time in the NFV enviroment.

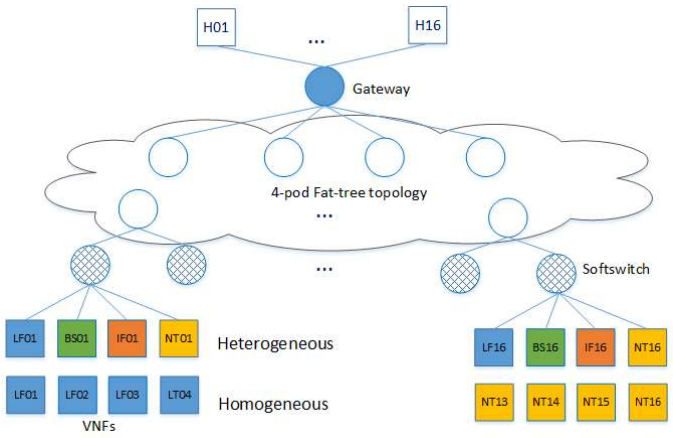


Fig. 3. Experimental setup.

3) VNF location table: After determining the next VNF $f_{i,j}$, packets are forwarded to this table, which is responsible for finding the connected softswitch $s_{k'}$ of VNF $f_{i,j}$. Obviously, the matching input here is $f_{i,j}$, and the associated action is to determine $s_{k'}$.

4) Path selection table: Since the VNF location table has determined the destination switch $s_{k'}$, it will select a specified path $p^u_{k,k'}$ from the current switch s_k to the destination switch $s_{k'}$ among the candidates. The task is done by hashing the packet's *flowcell_ID*, which consists of *flow_ID* and the MPLS TC field of a paket. In this study, we borrow the MPLS TC field to identify the flowcell to which the packet belongs. Similar to the VNF selection table, the identifier of the path $p^u_{k,k'}$ is also added to the packet using an MPLS label for routing purposes.

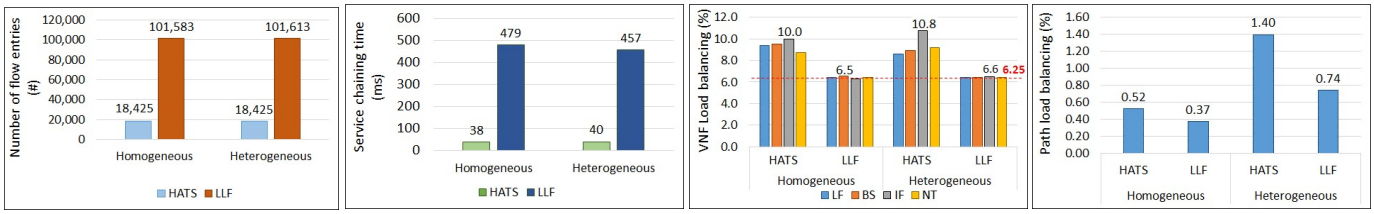
At a high level, the controller proactively computes all softswitch-to-softswitch paths $p^u_{k,k'} \in P$. Every path $p^u_{k,k'}$ is assigned a unique forwarding identifier. Once the paths are set up, the controller inserts the relevant forwarding rules into hardware switches to set up routing tunnels and installs the path information into the path selection tables in softswitches.

C. Implementation

To validate our design, we have implemented HATS using the ODL controller and Open vSwitch. A module is implemented on the controller to collect network topology and VNF information by periodically sending query commands to the data plane components via an ODL OpenFlow plugin. Subsequently, the corresponding flow entries are generated and then inserted into the switches.

1) VNF and path selection by OpenFlow Group table on Open vSwitch: OpenFlow group table [14] consists of multiple group entries which contain separate lists of actions referred to as OpenFlow buckets. The group types such as *ALL*, *INDIRECT*, *FAST FAILOVER* and *SELECT* determine how to apply buckets to incoming packets. For the group type SELECT, only one bucket is chosen to process packets using a switch-computed selection mechanism.

Since the selection mechanism of the group type SELECT on Open vSwitch is to hash a packet's fields, we leverage it to



a) Number of flow entries b) Service chaining time c) VNF load balancing d) Path load balancing

Fig. 4. VNF allocation: Homogeneous vs. Heterogeneous.

implement the hash-based traffic steering algorithm in HATS. Table I displays how VNF and Path selection tables are installed using Open vSwitch group tables. Note that we need to use two Open vSwitches to implement a softswitch s_k in HATS because Open vSwitch demands that the group table must be the last table in pipeline processing.

2) Elephant flow detection and Flowcell segmentation: To split an elephant flow into two or more flowcells, HATs maintains a per-flow counter at the gateway. We install a forwarding rule that exactly specifies all fields of the flow's *flow_ID* and an associated counter of bytes increased whenever a packet of the flow arrives at the gateway. A script periodically checks whether the counter exceeds the flowcell size. If yes, it increases the value of MPLS TC field, which is the flowcell identifier of incoming packets and resets the counter. Note that when flowcell identifier exceeds the maximum value of MPLS TC field, it is reset to zero. An alternative technique for flowcell segmentation is monitoring the TCP sequence numbers of packets in the same flow. However, this technique is inapplicable to UDP traffic and not supported by Open vSwitch.

IV. Performance Evaluation

This section presents the experimental evaluation in which the effectiveness of HATS is verified.

A. Experimental setup

Fig. 3 shows the experimental setup to evaluate the performance of HATS. Our experiments are conducted in a 4-pod fat-tree network topology simulated by Mininet network emulator. We use Click elements to implement four types of VNFs, namely, L2 packet forwarding (LF), bandwidth shaper (BS), IP packet filter (IF) and network address translation-NAT (NT). The VNFs are deployed in Mininet hosts, and there are 16 instances per VNF type. We implement HATS in Open vSwitch 2.5.0, and ODL Beryllium-SR2 is deployed as the SDN controller in our experiments.

Network service requests are TCP flows generated by iPerf3 [15]. We create a new flow every 100ms by randomly establishing a connection between pairs of hosts among 16 hosts, i.e., Host 1 (H01) ~ Host 16 (H16). In this study, we follow heavy-tailed flow size distribution from a data center [16]. That is, most of the data are from a small fraction of elephant flows. In our experiments, 10% of flows are elephants whose durations are exponentially distributed with the mean of 10s, while mice flows last for 1s. Each flow requires a service chain whose number of network functions is uniformly generated in the range [1, 4].

We compare the performance of HATS to Least Load First (LLF) algorithm. In contrast to our design, LLF performs load balancing in the control plane. That is, the controller estimates the current load of VNFs and paths by querying the number of transmitted bytes of switch ports in every 1s. Then, for each required network function in the service chains, the algorithm selects the least loaded VNF and path among feasible candidates to construct service paths. The two algorithms are compared and discussed based on four metrics, namely, the VNF load balancing criteria SL_i, the network path load balancing criteria NL, the total number of flow entries E, and the average service chaining time T. Note that we do not compare HATS to existing service chaining solutions [3,4,5,6,7] since they do not share the same objectives with our method or demand special requirements to implement.

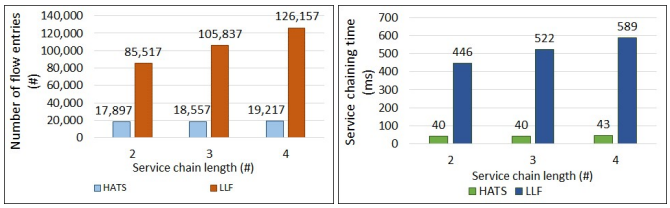
All the experiment results shown here are obtained by averaging the results of 5 simulation runs. Each run is terminated upon successful completion of 10,000 flows.

B. Result analysis

1) VNF allocation: Homogeneous vs. Heterogeneous: In this experiment, we observe the performance of investigated algorithms under two VNF allocation scenarios, namely homogeneous and heterogeneous. As can be seen in Fig. 3, a physical machine consists of only one type of VNFs in a homogeneous scenario, whereas each machine consists of all four types of VNFs in a heterogeneous scenario.

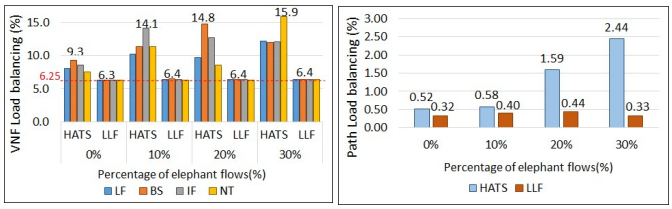
Figs. 4a and 4b clearly show that HATS reduces control and data plane overhead significantly. Compared with LLF, HATS can decrease the number of flow entries by about 82%, from 101.6 thousand to 18.4 thousand. Besides, the service chaining time is also reduced by over 92%, from 479ms to 38ms. The results are almost the same in homogeneous and heterogeneous scenarios. In other words, VNF allocation strategies do not affect control and data overhead.

The performance of VNF and path load balancing of HATS and LLF are compared in Figs. 4c and 4d. The experiment results indicate that HATS can efficiently perform VNF and path load balancing, but not as well as LLF due to hash collisions. There is a ~4.0% difference between two algorithms in the VNF load balancing criterion in both allocation scenarios. In term of path load balancing, LFF is about 0.15% and 0.66% better than HATS in homogeneous and heterogeneous scenarios. Fig. 4d also displays that both algorithms have better path load balancing performance in a homogeneous scenario. It can be explained by the fact that there are always multiple paths from a



a) Number of flow entries b) Service chaining time

Fig. 5. The impact of service chain length.



a) VNF load balancing b) Path load balancing

Fig. 6. The impact of elephant flows.

type-i VNF to a type-i' VNF in the scenario, but not in a heterogeneous scenario.

2) The impact of service chain length: We conduct another experiment in which the number of required network functions in service chains varies in the range [2,4], and VNFs are homogeneously allocated. By doing so, the performance of both algorithms is investigated with various service chain lengths. Note that we do not show the results of VNF and path load balancing in this experiment since the service chain lengths do not clearly impact on such two metrics.

As expected, the experiment results demonstrate that the length of service chains significantly affects the performance of LLF algorithm, but not HATS. As can be seen in Fig. 5, the number of flow entries and the service chaining time are linearly increased with the service chain length in the LLF algorithm. To be specific, the number of flow entries is increased from 85.5 thousand to 126.2 thousand when the length is changed from 2 to 4. In other words, LLF requires about 2 more entries per flow when the service chain length increases by one network function. Meanwhile, the service chaining time is increased from 446ms to 589ms in the same range of service chain length. On the other hand, the control and data plane overheads are almost stable in HATS. In the case in which the service chain length is set to 4, the performance difference between HATS and LLF is roughly 85% and 93% in terms of the number of flow entries and the service chaining time.

3) The impact of elephant flows: To understand the influence of elephant flows to the performance of HATS and LLF algorithms, we continue conducting this investigation in which the percentage of elephant flows varies in the range [0%,30%] and the VNF allocation is also homogeneous.

Fig. 6 displays the performance of HATS and LLF algorithms in terms of VNF and path load balancing metrics. The results indicate that LLF is unaffected by the percentage of elephant flows, i.e., its performance is stable with the value of about 6.4% and 0.4% in the two metrics. In the case of HATS, unfortunately, the performance is getting worse when the percentage of elephant flows is increased from 0% to 30%. Take path load balancing metric, for example; it increases from 0.52% to 2.44%. To sum up, the elephant flows have high impact in HATS algorithm, but not LLF algorithm.

V. CONCLUSION

This paper presents HATS which is an efficient load balancing mechanism for chaining VNFs in a data center. Our method employs a centralized controller to collect network topology and then disseminates corresponding load balancing information to edge softswitches responsible for splitting network traffic to VNFs through multiple paths using flow hashing technique. Compared to previous solutions, HATS does not require any special requirements to network hardware while significantly reduces control and data plane overheads. Our future works will focus on reducing the impact of elephant flows to load balancing performance, and handling packet reordering.

REFERENCES

[1] R. Mijumbi, J. Serrat, J. L. Gorricho, N. Bouten, F. D. Turck and R. Boutaba, "Network Function Virtualization: State-of-the-Art and Research Challenges," IEEE Communications Surveys Tutorials, vol. 18, no. 1, pp. 236–262, First Quarter 2016.

[2] P. Quinn and J. Guichard, "Service Function Chaining: Creating a Service Plane via Network Service Headers," IEEE Computer, vol. 47, no. 11, pp. 38–44, Nov. 2014.

[3] M.-T. Thai, Y.-D. Lin and Y.-C. Lai, "A joint network and server load balancing algorithm for chaining virtualized network functions," IEEE ICC, May 2016.

[4] G. Cheng, H. Chen, H. Hu, Z. Wang and J. Lan, "Enabling network function combination via service chain instantiation," Computer Networks, vol. 92, Part 2, pp. 396–407, Dec. 2015.

[5] T. Li, H. Zhou and H. Luo, "A new method for providing network services: Service function chain," Optical Switching and Networking.

[6] M. Arumaithurai, J. Chen, E. Monticelli, X. Fu and K. K. Ramakrishnan, "Exploiting ICN for Flexible Management of Software-defined Networks," in Proceedings of the 1st International Conference on Information-centric Networking, New York, NY, USA, 2014, pp. 107–116.

[7] P. C. Lin, Y. D. Lin, C. Y. Wu, Y. C. Lai, and Y. C. Kao, "Balanced Service Chaining in Software-Defined Networks with Network Function Virtualization," Computer, vol. 49, no. 11, pp. 68–76, Nov. 2016.

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

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

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

[11] E. Kohler, R. Morris, B. Chen, J. Jannotti and M. F. Kaashoek, "The Click Modular Router," ACM Trans. Comput. Syst., vol. 18, no. 3, pp. 263–297, Aug. 2000.

[12] J. Zhou, M. Tewari, M. Zhu, A. Kabbani, L. Poutievski, A. Singh and A. Vahdat, "WCMP: Weighted Cost Multipathing for Improved Fairness in Data Centers," in Proceedings of the Ninth European Conference on Computer Systems, New York, NY, USA, 2014, p. 5:1–5:14.

[13] K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter and A. Akella, "Presto: Edge-based Load Balancing for Fast Datacenter Networks," in Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, New York, NY, USA, 2015, pp. 465–478.

[14] "OpenFlow Switch Specification version 1.5.0". Available at: https://www.opennetworking.org/

[15] "iPerf3". Available at: https://iperf.fr/

[16] S. Kandula, S. Sengupta, A. Greenberg, P. Patel and R. Chaiken, "The Nature of Data Center Traffic: Measurements & Analysis," in Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference, New York, NY, 2009, pp. 202–208.