Journal of Network and Computer Applications 109 (2018) 1–10

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Towards load-balanced service chaining by Hash-based Traffic Steering on Softswitches

Minh-Tuan Thai [a,*], Ying-Dar Lin [b], Po-Ching Lin [c], Yuan-Cheng Lai [d]

[a] EECS International Graduate Program, National Chiao Tung University, No. 1001, Ta Hsueh Road, Hsinchu, 300, Taiwan
[b] Department of Computer Science, National Chiao Tung University, No. 1001, Ta Hsueh Road, Hsinchu, 300, Taiwan
[c] Department of Computer and Information Science, National Chung Cheng University, No. 168, Ta Hsueh Road, Chiayi, 62102, Taiwan
[d] Department of Information Management, National Taiwan University of Science and Technology, No. 43, Keelung Road, Taipei, 106, Taiwan

ARTICLE INFO

Keywords:
Load balancing
Network function virtualization
Service chaining
Software-defined networking

ABSTRACT

In this paper we study Hash-based Traffic Steering on Softswitches (HATS) which is a load balancing scheme for chaining virtualized network functions (VNFs), with the aim of mitigating the control and data plane overheads of existing methods. Our method uses a flow-hashing technique applied to softswitches to carry out server and network load balancing without triggering the controller. By exploiting the advantages of HATS, we then derive two algorithms, HATS with Flowcell-based Multipathing (HATS-Flowcell) and Dynamic Weight Adjustment for HATS (D-HATS), to address hash collision problems which downgrade system performance. The first algorithm divides an elephant flow into various equal-size flowcells, which are distributed over network paths as individual flows. The second algorithm periodically updates the hashing weights of VNFs and network paths according to their current load status. Our implementation demonstrates that HATS can be readily deployed on commodity network hardware. Furthermore, our experimental results show that D-HATS has roughly the same load balancing performance with Least Load First (LLF), a controller-based service chaining algorithm; while significantly reducing the number of flow entries and service chaining time by 54% and 93%.

1. Introduction

The concept of Network function virtualization (NFV) (Mijumbi et al., 2016) has the potential to enable flexible deployment, management, and provision of networking services in providers' data centers. With such a concept hardware-based appliances, i.e., middleboxes, are replaced with virtualized network functions (VNFs). Also, a composite network service can be provided via a service chain, which is a service policy defining a sequence of VNFs being applied to service requests, i.e., data flows (Quinn and Guichard, 2014). For example, a network security service may require data flows to pass through a firewall, followed by a content filter before ending up at an intrusion prevention system (IPS). To provide such services in a data center, network operators have to construct service paths, which are instances of service chains created using the overlay topology of VNFs. In other words, service paths define the interconnections of VNFs in an overlay network through which data flows have to be steered.

Service path instantiation, i.e., service chaining, raises two important concerns: VNF load balancing and network path load balancing (Thai et al., 2016). The first concern arises from the fact that there are multiple parallel instances of each VNF type for scale-out reasons. As a result, any proposed VNF load balancing scheme has to spread network traffic across those instances when service paths are being built. Furthermore, because of the multipath capacity of data center network topology, there should be a network load balancing solution for service path construction. These two issues have to be addressed to ensure operational efficiency and system performance.

A variety of research papers (Thai et al., 2016; Gember et al., 2013; Qazi et al., 2013; Carpio et al., 2017; Wang et al., 2017; Lin et al., 2016) have attempted to address the problems of load-balanced service chaining. Broadly speaking, almost all prior studies on service chaining are SDN-based solutions, in which the proposed load balancing algorithms are mostly implemented in the management and control planes. In other words, an SDN controller is responsible for computing service paths and then inserting forwarding rules to the data plane switches to steer flows across the required VNFs. Although these proffered solutions can fulfill the desired service requirements, they all have two critical drawbacks. The first is control plane overheads. This means the con-

* Corresponding author.
E-mail addresses: tmtuan.eed03g@nctu.edu.tw (M.-T. Thai), ydlin@cs.nctu.edu.tw (Y.-D. Lin), pclin@cs.ccu.edu.tw (P.-C. Lin), laiyc@cs.ntust.edu.tw (Y.-C. Lai).

https://doi.org/10.1016/j.jnca.2018.02.018
Received 17 October 2017; Received in revised form 24 January 2018; Accepted 27 February 2018
Available online 3 March 2018

troller needs to process a remarkable number of simultaneous packet-in events sent from the data plane. The second is data plane overheads, which are an enormous number of flow entries inserted into switches as each service path needs to be converted to a set of flow entries on all relevant switches. This also results in long flow setup times, which probably downgrades the system performance. According to Thai et al. (2016), such time is approximately 80% of the total service chaining time. To decrease control and data plane overheads, the authors in Lin et al. (2016) introduced a hash-based method to achieve load balancing among service nodes in an NFV environment. However, network load balancing issue was ignored in that work, and it required the support of an OpenFlow group table on hardware switches.

In this paper, we propose a load balancing system for chaining VNFs in a data center, called *Hash-based Traffic Steering on Softswitches (HATS)*. The aim of this work is to diminish control and data plane overheads in existing service chaining solutions. The main idea behind our design is that a centralized controller is employed to collect network topology and to pro-actively install the corresponding load balancing information to edge softswitches (software-based switches), which steer network traffic to different VNFs via multiple paths. VNF and path selections required 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 overheads. Furthermore, there are no special requirements for additional network hardware in this proposal.

HATS is derived from the popular hash-based load balancing scheme (Lin et al., 2016; Hopps, 2017), which selects VNFs and network paths among candidates to construct service paths based on the hash value of flows' header fields. Unfortunately, hash collisions cause a few elephant flows (long-lived and high bandwidth demands) to have the same hash value. These flows then travel to the same VNFs and network paths, even though the others may be lightly loaded. Such collisions may significantly degrade the system load balancing performance. To address this problem, we investigate two algorithms that are *HATS with Flowcell-based Multipathing* (HATS-Flowcell) and *Dynamic Weight Adjustment for HATS* (D-HATS). In the first algorithm, an elephant flow is split into various equal-size bursts of packets, or flowcells. The flowcells are then spread over network paths as individual flows. The second algorithm does not divide elephant flows to different sub-flows, but periodically updates the hashing weights of VNFs and network paths according to their current load status. Over-loaded VNFs and network paths tend to have smaller weights than under-loaded ones. In this way, the algorithm can address hash collision issues effectively.

We have developed our proposals using OpenDayLight (ODL) controller platform (Opendaylight, 2017) and Open vSwitch (Open vSwitch, 2017), whereby VNF and path selections are executed by the SELECT type of OpenFlow group table. The performance of HATS-Flowcell and D-HATS was evaluated using the Mininet network emulator (Mininet, 2017) with VNFs implemented by Click elements (Kohler et al., 2000). The experimental results showed that our proposed algorithms could effectively carry out VNF and network path load balancing while significantly decreasing the number of flow entries and service chaining time, compared to a controller-based service chaining algorithm.

Compared to an earlier conference version of this work (Thai et al., 2017), we add a new section which provides background knowledge on NFV and assesses existing load-balanced service chaining studies. Furthermore, an additional algorithm D-HATS, with its design, implementation, and experimental results, is added, with the aim of improving system load balancing performance. We also include more detailed analysis and comparisons between the proposed algorithms.

The rest of this paper is organized as follows. In Section 2, we review the background knowledge and related work of this study. Our load-balanced service chaining problem is described in Section 3, and in Section 4, we elaborate the proposed algorithms and their implementation. Evaluation and experimental results are presented in Section 5, and Section 6 concludes this paper.

2. Background and related work

We first give a brief overview of NFV and then review existing load balancing solutions for service chaining which are closely related to our work.

2.1. Background - network function virtualization

Telecommunication service providers have traditionally relied on proprietary appliances to offer network services. This expensive equipment has long product cycles, low service agility and cannot be flexibly deployed, which requires not only costly investments but also prevents network services from being upgrading. The concept of NFV has been proposed to address these issues by moving packet processing tasks from purpose-built hardware appliances to software-based functions. In other words, the functions are implemented on virtual machines by exploiting virtualization and cloud technology (Baker et al., 2015; Al-Dawsari et al., 2015). The virtual machines are programmed to play roles of network traffic processing functions such as firewall, IDS/IPS, NAT, video transcoding, and HTTP proxy/cache. In literature, these functions are referred as virtualized network functions (VNFs). By applying NFV, different VNFs can run on general-purpose servers and share underlying resources. Furthermore, the VNFs can be flexibly deployed at various times in different locations of a network such as federated data centers, core servers, and network edges. This flexibility can enhance system performance and provide dynamic scaling on service provisioning.

With the aim to develop standards for NFV, the European Telecommunication Standards Institute (ETSI) (ETSI - NFV, 2017) introduced an NFV architecture framework, which has been widely accepted by both academia and industry. There are two key components in the architecture, NFV infrastructure (NFVI) and an NFV management and orchestration platform (NFV-MANO). The NFVI includes both hardware and software resources that provide processing capacity for VNFs and connectivity among them. At a higher tier, the deployment and operation of these VNFs over the NFVI resources are managed by NFV-MANO.

Along with standardization activities, there has been a variety of attempts to provide NFV-enabled solutions. OPNFV (Open Platform for NFV (OPNFV), 2017) is an open source platform that attempts to establish an ecosystem for NFV. The platform allows users to integrate solutions from third-party projects such as ODL (Opendaylight, 2017), ONOS (Open Network Operating System, 2017), OpenStack (OpenStack, 2017), and Open vSwitch (Open vSwitch, 2017), in order to construct NFVI and NFV-MANO. By exploiting Xen-based virtualization (Barham et al., 2003) and network I/O optimization techniques, ClickOS (Martins et al., 2014) enables developers to implement various high-performance network functions using small-size, quick-booting, short-delay virtual machines. Similarly, the authors in Hwang et al. (2015) have developed a NetVM platform, which is built on top of KVM platform (Kivity et al., 2007) and Intel DPDK library (DPDK, 2017). The solution produces customizable data plane processing which can achieve near hardware performance.

2.2. Related work - load-balanced service chaining

By the concept of NFV, a composite network service can be provided through a service chain, which is a pre-defined sequence of VNFs being applied to data flows. To accommodate user requests for such services, providers need to carry out service chaining by selecting appropriate VNFs and network paths to construct service paths, in which data flows need to be steered across.

It is well-known that one should balance VNF and network path loads while performing service chaining to ensure efficient system

performance. Several studies have been carried out to address such requirements. A typical approach is to implement load balancing algorithms to a centralized SDN controller. In other words, the controller selects appropriate VNFs and network paths among suitable candidates to construct service paths according to the algorithms and then sets forwarding entries to the data plane. For example, SIMPLE (Qazi et al., 2013) introduced an online load balancing problem formulated as a linear programming problem whose objective is to minimize the maximum service functions load across the network. On the other hand, an offline optimization problem for multi-resource load balancing in NFV environment was proposed by Wang et al. (2017). The authors in Thai et al. (2016) developed a joint optimization algorithm aimed at balancing network and server load concurrently. Their experimental results showed that joint optimization outperforms a sequential algorithm. Stratos (Gember et al., 2013) also jointly balanced loads between service functions and network paths by splitting network traffic across service functions according to their network latency. However, the method distributes the packets of a flow to different instances of a stateful service function, which may cause incorrect processing results. Carpio et al. (2017) studied a VNF placement problem whose objective was to utilize the cost for all links in the network. Their results indicated that VNF placement schemes have a significant impact on network load balancing performance.

In all above proposed solutions, the controller must track the information of VNFs and network paths to select the appropriate ones for service chaining when receiving packet-in events from data plane. It is a challenging task because of the enormous numbers of service requests in a data center environment. Furthermore, the controller needs to convert service chaining rules to flow entries which are then inserted into relevant data plane switches. Such a large number of flow entries results in long service chaining time, which would dramatically downgrade system performance. To address the disadvantages of controller-based solutions, BHT (Lin et al., 2016) first constructed a tree of VNFs, and then used a flow-hashing technique to spread incoming traffic among the VNFs. By doing so, this approach was able to balance loads among VNFs in an NFV environment. Although the solution can achieve efficient load balancing performance, it requires the support of OpenFlow group table on hardware switches.

3. Problem description

This section first formally describes the terminology used in this study, and then states the problem to be addressed.

3.1. Terminology

3.1.1. 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 $N = \{n_i, 0 \le i < |N|\}$ be the set of $|N|$ network functions provided by the system. The VNFs of the system are denoted by a set $F = \{f_{i,j}, 0 \le j < M_i\}$, where $f_{i,j}$ and M_i are the j^{th} instance and the number of instances of n_i, respectively. We use $V(f_{i,j})$ to indicate the volume of network traffic arriving at VNF $f_{i,j}$.

3.1.2. Network topology

Let $S = \{s_k, 0 \le k < |S|\}$ be the set of $|S|$ softswitches in the network topology to which VNFs are connected. The softswitch-to-softswitch paths in the network are represented by a set $P = \{p^u_{k,k'}, 0 \le k, k' < |S|\}$, where $p^u_{k,k'}$ is the u^{th} 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

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

3.1.3. Service request

Let $r_v \in R$ be a service request arriving in the system, which demands a service chain $c_v \in C$ for its processing. The service chain c_v 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 network functions $n_i \in N'_v$.

3.1.4. VNF and path load balancing

We define the VNF load balancing criterion of the system as

$$VL = \max_{n_i \in N} \max_{f_{i,j} \in n_i} \left(\frac{V(f_{i,j})}{\sum_{f_{i,j} \in n_i} V(f_{i,j})} \right) * 100\%, \tag{2}$$

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

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

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

3.1.5. Control and data plane overhead

The overhead includes the number of flow entries in the system and the average service chaining time for service requests. The former is denoted by E, and the latter is denoted by T.

3.2. Problem statement

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

4. Hash-based load balanced Traffic Steering on Softswitches

In this section, we present our solution for solving the service chaining problem defined in Section 3. We first give the overview of HATS with its 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.

4.1. Approach overview

The architecture of HATS shown in Fig. 1 presents an overview of HATS where 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 provide the VNF and network path load balancing while reducing the control and data plane overhead in service chaining. This requirement is satisfied by our design, which consists the fundamental design decisions described below.

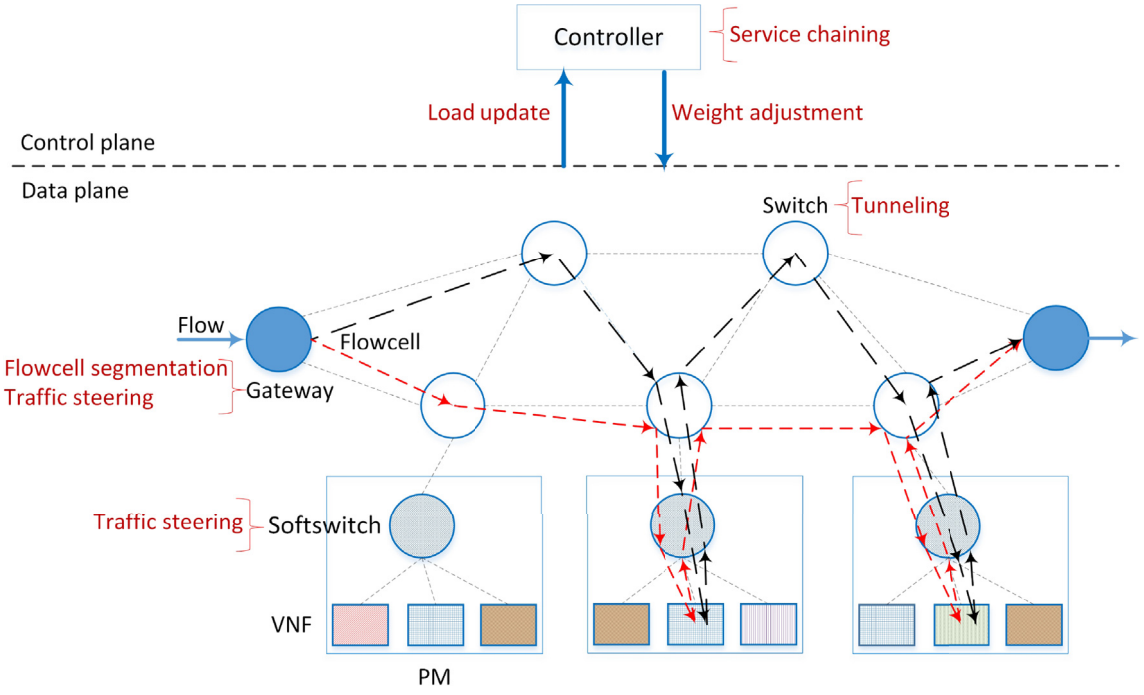


Fig. 1. The architecture of HATS.

4.1.1. Hash-based Traffic Steering on Softswitches

Most literature does not differentiate service chaining and traffic steering. Here we define service chaining as finding the appropriate VNFs and the associated order that must be applied to flows, and traffic steering as forwarding packets through VNFs. Chaining is thus to compute the service paths in the background periodically and store them in the flow tables which are looked up when steering packets.

Current studies (Thai et al., 2016; Gember et al., 2013; Qazi et al., 2013; Carpio et al., 2017; Wang et al., 2017) only implement load balancing intelligence in service chaining algorithms, while in HATS, load balancing resides in both service chaining and traffic steering procedures. The fundamental idea is that the controller does not construct service paths with specific VNFs and network paths, but pro-actively only inserts the information of possible candidates to data plane components such as softswitches and gateways. The components then select specific VNFs and network paths by hashing packets' header 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 markedly lowered. It should be noted that the gateways in this architecture are responsible for steering network traffic to the first VNFs of service paths.

4.1.2. 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 and network paths is kept in softswitches. After a packet is processed by a VNF, it will be redirected to a softswitch responsible for determining the next VNF by flow-hashing. Such steps are repeated until the packet travels through all required VNFs and returns to a gateway. Similarly, pre-configured softswitch-to-softswitch paths are inserted to softswitches, and network traffic is also split into these paths by flow-hashing. The hardware switches in the network are only responsible for tunneling network traffic from one softswitch to another.

Even though hash-based load balancing algorithm can also be accomplished in per-gateway and per-hop fashion, we decided to implement the algorithm in softswitches for three reasons. The first reason is that a per-gateway approach is inefficient because of the huge number of possible service paths in the network. While a per-hop approach performs poorly under asymmetric topologies (Zhou et al., 2014) and needs special requirements in hardware switches, such as flow-hashing and SDN support, which result in extra hardware cost and complicate systems management. The second reason is that softswitches reside at a suitable location for load balancing tasks, right above VNFs. They can easily modify packets without requiring any changes to VNFs or hardware switches. It is reasonable that the next destination of a packet and the path to reach that 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.

4.1.3. HATS with flowcell-based multipathing (HATS-Flowcell)

Hash-based load balancing has a severe drawback, which is the hash collisions of elephant flows. Because the flows share the same hash value, they travel to the same VNFs and network paths, which dramatically downgrades system load balancing performance. To address this, we follow the idea of flowcells (He et al., 2015), which are equal-size bursts of packets, by breaking an elephant flow at a gateway. The flowcells are then distributed over the paths as individual flows. Note that the flowcells of the same flow cannot be processed by different VNFs of a stateful network function. In other words, in this design the flowcells must travel through the same VNFs but via different paths.

The flowcell approach promises to provide better load balancing, but as it spreads packets of the same flow among multiple paths, the packets may experience different latencies, which introduces packet re-ordering. The problem must be carefully addressed to ensure the efficiency of any sub-flow load balancing schemes.

4.1.4. Dynamic Weight Adjustment for HATS (D-HATS)

Here we propose the Dynamic Weight Adjustment technique for our hash-based solution (D-HATS) with the aim of improving load balancing

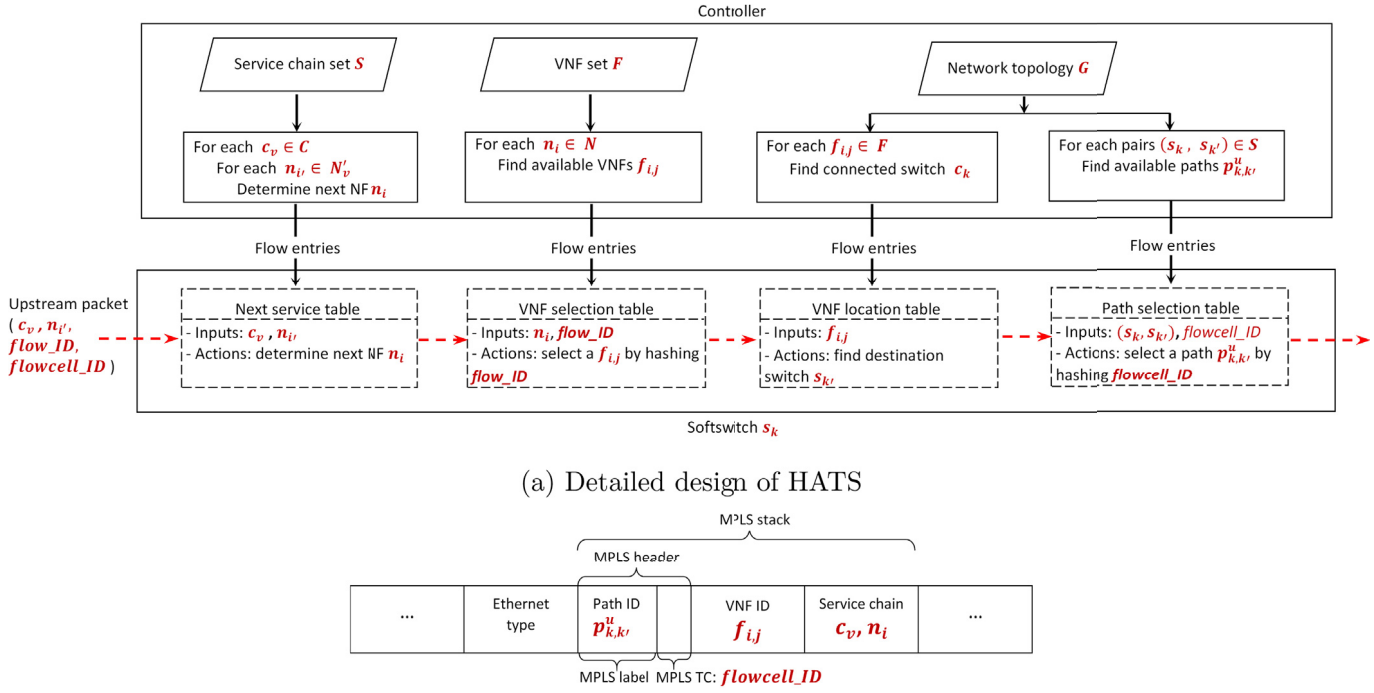


Fig. 2. Controller-based service chaining and hash-based traffic steering on an edge softswitch.

Table 1
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 $p^u_{k,k'}$

performance. Different from HATS-Flowcell, D-HATS avoids packet reordering issues by applying per-flow load balancing, i.e., the technique does not break a flow to flowcells.

To lower hashing collisions that cause the problems of unbalanced loads among VNFs and network paths, D-HATS exploits *weighted hash functions* to carry out VNF and network path load balancing. That is, data flows are distributed to VNFs and network paths by hash functions whose weights may be different. In addition, the weights are dynamically adjusted according to the current load status of the VNFs and network paths. Specifically, D-HATS measures the volume of network traffic arriving at a VNF $V(f_{i,j})$ and a path $V(p^u_{k,k'})$ at every time interval t. The algorithm then computes the hashing weights $wf_{i,j}$ of a VNF $f_{i,j}$, which is defined as

$$wf_{i,j} = \frac{\max_{f_{i,j} \in n_i} V(f_{i,j})}{V(f_{i,j})}, \tag{4}$$

Similarly, D-HATS also computes the hashing weight wp_u of a path $p^u_{k,k'}$, which is defined as

$$wp_u = \frac{\max_{p^u_{k,k'} \in P} V(p^u_{k,k'})}{V(p^u_{k,k'})}, \tag{5}$$

Eventually, D-HATS updates the weights of hash functions implemented in softswitches that are responsible for spreading flows to different VNFs and paths.

With D-HATS, over-loaded VNFs and network paths, which have smaller weights, are less likely to be selected than under-loaded ones during the period t. By doing so, the algorithm reduces the chance of hashing collisions for congested VNFs and network paths. Hence, it can perform load balancing more efficiently.

4.2. Detailed design of HATS

Fig. 2 shows the details of the HATS design, where a centralized controller is employed to collect the information of service chains C, VNFs F, and network topology G. The controller then generates traffic steering rules by inserting flow entries into the forwarding tables of 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 network 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 destination VNFs.

4.2.1. 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.

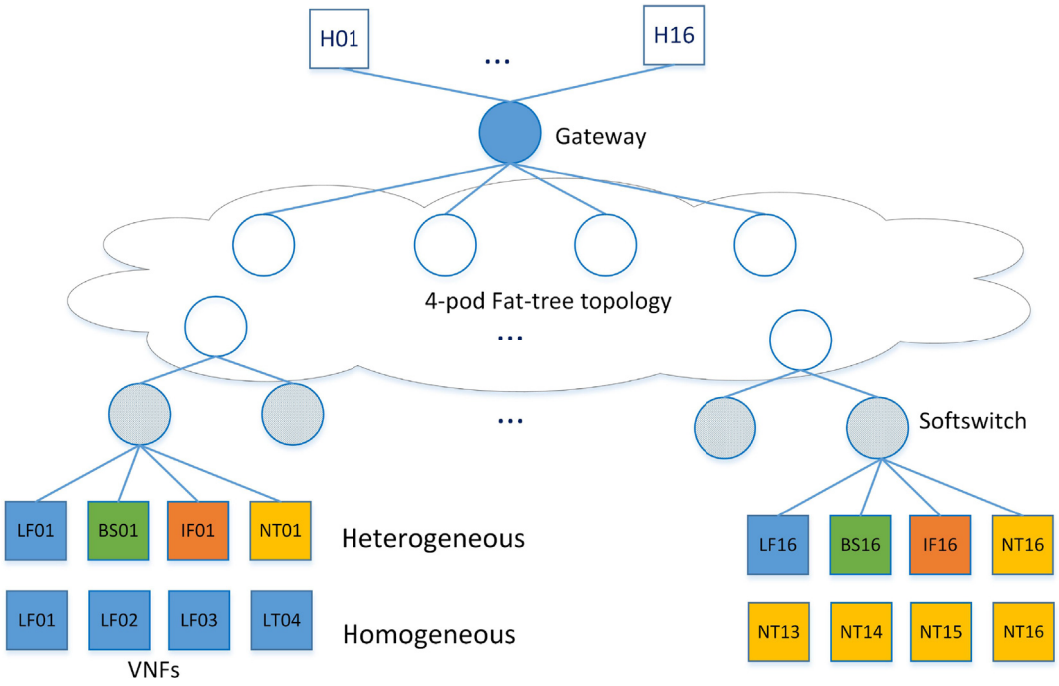


Fig. 3. Experimental setup.

Table 2
Load balancing algorithms compared in this paper.

Algorithm	Management/Control/ Data plane algorithm	Overhead	Packet re-order
HATS-Flowcell	Control and data plane	• N/A	Yes
LLF	Management and control plane	• VNF and path load tracking • Per-flow state	No
D-HATS	Control and data plane	• VNF and path load tracking • Hashing weight update	No

4.2.2. VNF selection table

The matching input is next network function n_i. This table, which contains the information of available VNFs belonging to the function n_i, will select a particular VNF $f_{i,j}$ by hashing the *flow_ID* of a packet. The identifier of $f_{i,j}$ is then encapsulated to the packet by adding an MPLS label. The *flow_ID* consists of the packet's header 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 environment.

4.2.3. 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 $s_{k'}$.

4.2.4. 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 candidates. The task is done by hashing packet's *flowcell_ID*, which consists of *flow_ID* and the MPLS TC field of a packet. 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 (Baker et al., 2015; Al-Dawsari et al., 2015).

At a high level, the controller pro-actively 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.

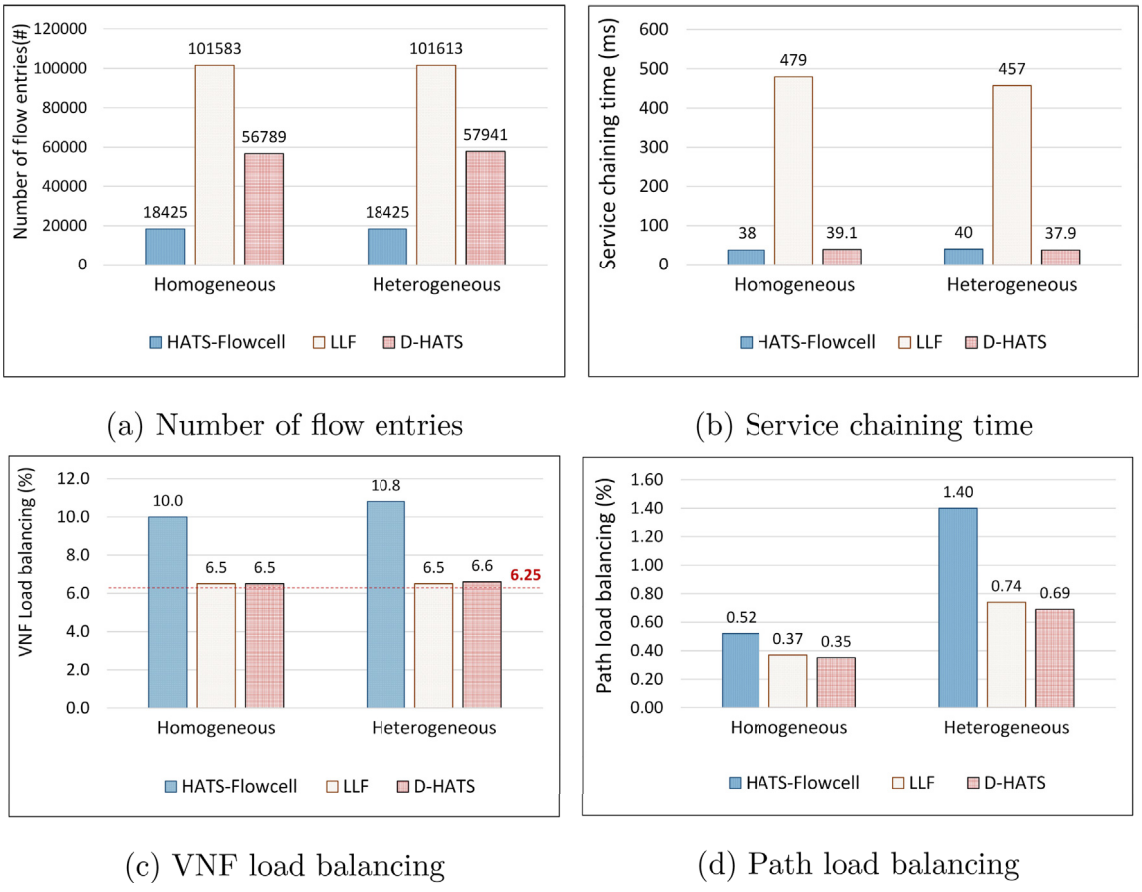


Fig. 4. VNF allocation: Homogeneous vs. Heterogeneous.

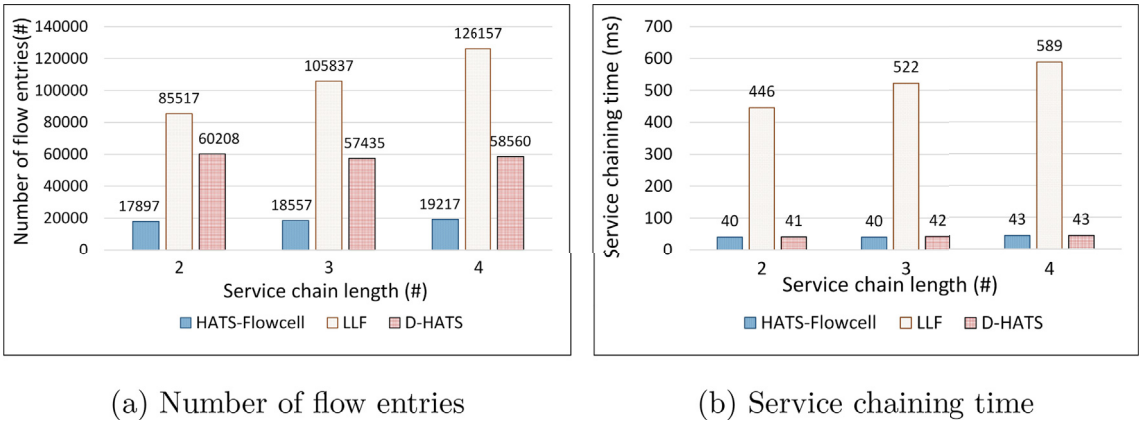


Fig. 5. The impact of service chain length.

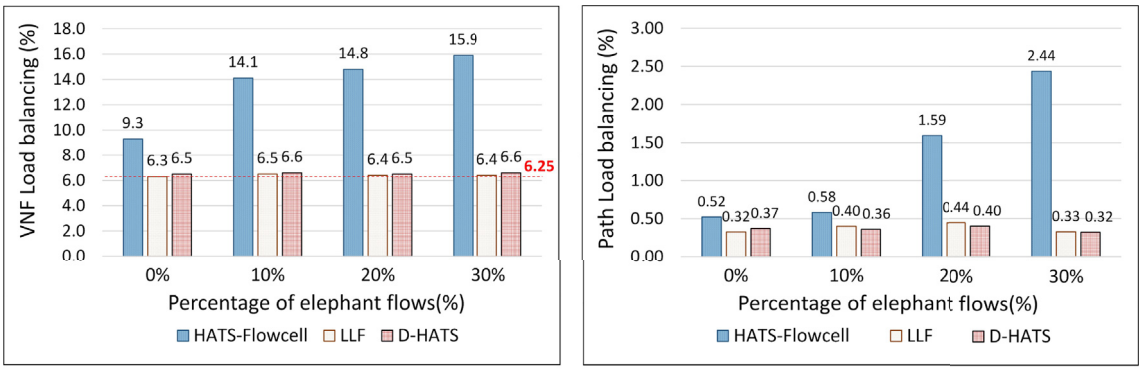
4.3. Implementation

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 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.

4.3.1. VNF and path selection by OpenFlow group table on open vSwitch

OpenFlow group table (OpenFlow, 2017) consists of multiple group entries which contain separate lists of actions referred to as OpenFlow buckets. The group types, which are *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 header fields, we leverage it to implement the hash-based traffic steering algorithm in HATS. Table 1 shows 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.



(a) VNF load balancing (b) Path load balancing

Fig. 6. The impact of elephant flows.

4.3.2. Elephant flow detection and flowcell segmentation

To split an elephant flow into two or more flowcells, HATs maintains a per-ow counter at the gateway. We install a forwarding rule that specifies all fields of *flow_ID* of the flow, and an associated counter of bytes increases 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 then resets the counter. Note that when a 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 approach is inapplicable to UDP traffic and is not supported by Open vSwitch.

4.3.3. VNF and path hashing weight adjustment

To dynamically update the hashing weights of VNFs and paths, we implement a module in the ODL controller which periodically queries the number of transmitted bytes from softswitch ports. After that, the controller measures the increment from the last monitoring time, and then estimates the volume of network traffic that arrives at every VNF and network path, i.e., $V(f_{i,j})$ and $V(p^u_{k,k'})$, respectively. Next, the controller computes the weights of VNFs $wf_{i,j}$ and paths wp_u using Equations (4) and (5). Eventually, the controller sends OpenFlow GroupMod messages to softswithes to modify the weights of corresponding hash functions in VNF and Path selection tables.

5. Performance evaluation

This section presents the experimental evaluation in which the effectiveness of our proposed algorithms is verified.

5.1. Experimental setup

Fig. 3 shows the experimental setup for evaluating the performance of our algorithms. Our experiments were conducted in a 4-pod fat-tree network topology simulated by Mininet network emulator. We used 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 were deployed in Mininet hosts, and there were 16 instances per VNF type. We implemented HATS in Open vSwitch 2.5.0, and ODL Beryllium-SR2 were deployed as the SDN controller in our experiments.

Network service requests are TCP flows generated by iPerf3 (iPerf, 2017). We created a new flow every 100 ms by randomly establishing a connection between pairs of hosts. There were 16 hosts, i.e., from Host 1 (H01) to Host 16 (H16), in our experiments. In this study, we follow a heavy-tailed flow size distribution from a data center (Kandula et al., 2009); most of the data bytes are from a small fraction of elephant flows. In our experiments, unless stated differently, 10% of flows were elephants whose durations were exponentially distributed with the mean of 10s, while others, i.e., mice flows, last for 1s. Each flow required a service chain whose number of network functions was uniformly generated in the range from 1 to 4.

We compared the performance of HATS-Flowcell and D-HATS to Least Load First (LLF) algorithm. In contrast to our algorithms, LLF mostly performs load balancing in the management and control plane. That is, the controller estimates the current load of VNFs and network paths by querying the number of transmitted bytes of switch ports in every 1s. Then, for each required network function in service chains, the algorithm selects the least loaded VNF and path among feasible candidates to construct service paths. Since HATS-Flowcell and D-HATS aim to lower control and plane overheads of existing controller-based service chaining solutions (Thai et al., 2016; Gember et al., 2013; Qazi et al., 2013; Carpio et al., 2017; Wang et al., 2017), it is appropriate to compare the algorithms to LLF, which shares common characteristics with these solutions.

Table 2 gives a summary of the three algorithms compared in this paper. The algorithms were compared and discussed based on four metrics, namely, the VNF load balancing criteria *SL*, the network path load balancing criteria *NL*, the total number of flow entries *E*, and the average service chaining time *T*. All the experimental results shown here were obtained by averaging the results of 5 simulation runs. Each run was terminated on successful completion of 10,000 data flows.

5.2. Result analysis

5.2.1. VNF allocation: homogeneous vs. heterogeneous

In this experiment we observed the performance of investigated algorithms under homogeneous and heterogeneous VNF allocation scenarios. As can be seen in Fig. 3, in a homogeneous scenario a physical machine consists of only one type of VNFs, whereas each machine consists of all four types of VNFs in a heterogeneous scenario.

Fig. 4a and b clearly show that HATS-Flowcell and D-HATS reduced control and data plane overheads significantly. Compared with LLF, HATS-Flowcell and D-HATS decreased the number of flow entries by about 82% and 43%, from 101.6 thousand to 18.4 thousand and 57.9 thousand, respectively. D-HATS generated a number of flow entries which are about three times more than those in HATS-Flowcell since it had to send flow entries to update the weights of hash functions in the data plane during experimental runs. In terms of service chaining time, the metric was reduced by over 92%, from 479 ms to about 40 ms by HATS-Flowcell and D-HATS algorithms because they both did not need to install per-flow matching entries to data plane switches as LLF

algorithm. The results were almost the same in homogeneous and heterogeneous scenarios. In other words, VNF allocation strategies do not affect control and data overheads.

The load balancing performance of compared algorithms is shown in Fig. 4c and d. The experimental results show that D-HATS has almost the same VNF and path load balancing performance with LLF. Although HATS-Flowcell can efficiently perform VNF and path load balancing, its performance is unfortunately not as good as those of LLF and D-HATS as a result of hash collisions. There is about 4.0% difference in the VNF load balancing criterion in both allocation scenarios. In terms of path load balancing, LLF and D-HATS are about 0.15% better than HATS-Flowcell in homogeneous and 0.7% better in heterogeneous scenarios. Fig. 4d also shows that the three algorithms have better path load balancing performance in a homogeneous scenario because there are always multiple paths from a type-i VNF to a type-i' VNF in the scenario, but not in the heterogeneous scenario.

5.2.2. The impact of service chain length

We conducted another experiment in which the number of required network functions in service chains varied in the range of (Quinn and Guichard, 2014; Gember et al., 2013) functions, and VNFs were homogeneously allocated. By doing so, the performance of different algorithms was 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 clearly do not impact on the two metrics.

As expected, the results demonstrate that the length of service chains significantly affects the performance of LLF algorithm, but not HATS-Flowcell and D-HATS. As can be seen in Fig. 5, the number of flow entries and the service chaining time increased linearly with the service chain length in the LLF algorithm. To be specific, the number of flow entries increased from 85.5 thousand to 126.2 thousand when the length was changed from 2 to 4. In other words, LLF requires about two more entries per flow when the service chain length increases by one network function. Meanwhile, the service chaining time increased from 446 ms to 589 ms in the service chain length range. On the other hand, control and data plane overheads are almost stable in HATS-Flowcell and D-HATS. If the service chain length was set to 4, the difference between D-HATS and LLF was roughly 54% and 93% in terms of the number of flow entries and the service chaining time, respectively.

5.2.3. The impact of elephant flows

To understand the influence of elephant flows on the performance of the three algorithms, we investigated this experiment in which the percentage that elephant flows varied from 0% to 30%, and VNF allocation was homogeneous.

Fig. 6 shows the performance of the three algorithms in terms of VNF and path load balancing metrics. The experimental results indicate that LLF and D-HATS were unaffected by the percentage of elephant flows, with a stable value of about 6.5% and 0.4% in the two metrics, respectively. In the case of HATS-Flowcell, unfortunately, the performance deteriorated when the percentage of elephant flows was increased from 0% to 30%. The path load balancing metric, for example; increased from 0.52% to 2.44%. To sum up, elephant flows have a high impact on HATS-Flowcell, but not on the LLF and D-HATS algorithms.

6. Conclusion

This paper has presented HATS which is an efficient load balancing system for chaining VNFs in a data center environment. Our method employs a centralized controller that collects network topology and then inserts the corresponding load balancing information to edge softswitches, which are responsible for splitting network traffic to different VNFs through multiple paths, using flow-hashing technique. We also derived two algorithms, HATS-Flowcell and D-HATS, to address hash collision issues, to ensure the efficient system performance. Compared to existing studies, our proposals do not require any special network hardware requirements, while significantly reducing control and data plane overheads. Furthermore, our proposed algorithms have roughly the same load balancing performance with LLF, a controller-based algorithm. Future work will focus on extending HATS to perform load balancing while supporting dynamic deployment and scaling of VNFs.

References

Al-Dawsari, B., Baker, T., England, D., 2015. Trusted energy-efficient cloud-based services brokerage platform. Int. J. Intell. Comput. Res. (IJICR). 6 (4), 630–639, https://doi.org/10.20533/ijicr.2042.4655.2015.0078, http://infonomics-society.org/wp-content/uploads/ijicr/published-papers/volume-6-2015/Trusted-Energy-Efficient-Cloud-Based-Services-brokerage-Platform.pdf.

Baker, T., Al-Dawsari, B., Tawfik, H., Reid, D., Ngoko, Y., 2015. GreeDi: an energy efficient routing algorithm for big data on cloud. Ad Hoc Netw. 35, 83–96, https://doi.org/10.1016/j.adhoc.2015.06.008, http://www.sciencedirect.com/science/article/pii/S1570870515001341.

Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A., 2003. Xen and the art of virtualization. In: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, SOSP '03. ACM, New York, NY, USA, pp. 164–177, http://doi.acm.org/10.1145/945445.945462.

Carpio, F., Dhahri, S., Jukan, A., 2017. VNF placement with replication for Load balancing in NFV networks. In: 2017 IEEE International Conference on Communications (ICC), pp. 1–6, https://doi.org/10.1109/ICC.2017.7996515.

DPDK Boosts Packet Processing, Performance, and Throughput. , 2017. https://www.intel.com/content/www/us/en/communications/data-plane-development-kit.html [Accessed 14 January 2017].

ETSI - NFV, 2017. http://www.etsi.org/technologies-clusters/technologies/nfv. (Accessed 12 October 2017).

Gember, A., Krishnamurthy, A., John, S.S., Grandl, R., Gao, X., Anand, A., Benson, T., Akella, A., Sekar, V., May 2013. Stratos: a Network-aware Orchestration Layer for Middleboxes in the Cloud. Tech. rep. http://arxiv-web3.library.cornell.edu/abs/1305.0209v1.

He, K., Rozner, E., Agarwal, K., Felter, W., Carter, J., Akella, A., 2015. Presto: edge-based load balancing for fast Datacenter networks. In: Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, SIGCOMM '15. ACM, New York, NY, USA, pp. 465–478, http://doi.acm.org/10.1145/2785956.2787507.

Hopps, C.E., 2017. Analysis of an Equal-cost Multi-path Algorithm. <chopps@merit.edu> https://tools.ietf.org/html/rfc2992 [Accessed 10 October 2017].

Hwang, J., Ramakrishnan, K.K., Wood, T., 2015. NetVM: high performance and flexible networking using virtualization on commodity platforms. IEEE Transac. Netw. Service Manag. 12 (1), 34–47, https://doi.org/10.1109/TNSM.2015.2401568.

iPerf - The TCP, UDP and SCTP Network Bandwidth Measurement Tool. , 2017. https://iperf.fr/ [Accessed 12 October 2017].

Kandula, S., Sengupta, S., Greenberg, A., Patel, P., Chaiken, R., 2009. The nature of data center traffic: measurements & analysis. In: Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference, IMC '09. ACM, New York, NY, USA, pp. 202–208, http://doi.acm.org/10.1145/1644893.1644918.

Kivity, A., Kamay, Y., Laor, D., Lublin, U., Liguori, A., 2007. KVM: the Linux virtual machine monitor. In: In Proceedings of the 2007 Ottawa Linux Symposium (OLS).

Kohler, E., Morris, R., Chen, B., Jannotti, J., Kaashoek, M.F., 2000. The Click modular router. ACM Trans. Comput. Syst. 18 (3), 263–297, http://doi.acm.org/10.1145/354871.354874.

Lin, P.C., Lin, Y.D., Wu, C.Y., Lai, Y.C., Kao, Y.C., 2016. Balanced service chaining in software-defined networks with network function virtualization. IEEE Comput. 49 (11), 68–76, https://doi.org/10.1109/MC.2016.349.

Martins, J., Ahmed, M., Raiciu, C., Olteanu, V., Honda, M., Bifulco, R., Huici, F., 2014. ClickOS and the art of network function virtualization. In: Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation, NSDI'14. USENIX Association, Berkeley, CA, USA, pp. 459–473, http://dl.acm.org/citation.cfm?id=2616448.2616491.

Mijumbi, R., Serrat, J., Gorricho, J.L., Bouten, N., Turck, F.D., Boutaba, R., 2016. Network function virtualization: state-of-the-art and research challenges. IEEE Commun. Surv. Tutorials 18 (1), 236–262, https://doi.org/10.1109/COMST.2015.2477041.

Mininet, 2017. An Instant Virtual Network on Your Laptop (Or Other PC) - Mininet. http://mininet.org/ [Accessed 12 October 2017].

Open Network Operating System, 2017. https://onosproject.org. (Accessed 12 October 2017).

Open Platform for NFV (OPNFV), 2017. https://www.opnfv.org/. (Accessed 14 January 2017).

Open vSwitch, 2017. http://openvswitch.org/. (Accessed 12 October 2017).

Opendaylight, 2017. https://www.opendaylight.org/. (Accessed 12 October 2017).

OpenFlow Switch Specification, 2017. https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf version 1.5.1 [Accessed 12 October 2017].

OpenStack - Open Source Software for Creating Private and Public Clouds. , 2017. https://www.openstack.org/ [Accessed 12 October 2017].

Qazi, Z.A., Tu, C.-C., Chiang, L., Miao, R., Sekar, V., Yu, M., 2013. SIMPLE-fying middlebox policy enforcement using SDN. In: Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM, SIGCOMM '13. ACM, New York, NY, USA, pp. 27–38, http://doi.acm.org/10.1145/2486001.2486022.

Quinn, P., Guichard, J., 2014. Service function chaining: creating a service plane via network service headers. IEEE Comput. 47 (11), 38–44, https://doi.org/10.1109/MC.2014.328.

Thai, M.T., Lin, Y.D., Lai, Y.C., 2016. A joint network and server load balancing algorithm for chaining virtualized network functions. In: 2016 IEEE International Conference on Communications (ICC), pp. 1–6, https://doi.org/10.1109/ICC.2016.7510712.

Thai, M.T., Lin, Y.D., Lin, P.C., Lai, Y.C., 2017. Hash-based load balanced traffic steering on softswitches for chaining virtualized network functions. In: 2017 IEEE International Conference on Communications (ICC), pp. 1–6, https://doi.org/10.1109/ICC.2017.7997162.

Wang, T., Xu, H., Liu, F., 2017. Multi-resource load balancing for virtual network functions. In: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), pp. 1322–1332, https://doi.org/10.1109/ICDCS.2017.233.

Zhou, J., Tewari, M., Zhu, M., Kabbani, A., Poutievski, L., Singh, A., Vahdat, A., 2014. WCMP: weighted cost multipathing for improved fairness in data centers. In: Proceedings of the Ninth European Conference on Computer Systems, EuroSys '14. ACM, New York, NY, USA, pp. 5:1–5:14, http://doi.acm.org/10.1145/2592798.2592803.

Minh-Tuan Thai received his M.S. degree in computer science from National Chiao Tung University (NCTU) - Taiwan in 2013. He currently is a Ph.D. candidate in Computer Science at EECS International Graduate Program, NCTU. Since 2014, he has been a graduate student researcher at High Speed Network Lab, NCTU and a part-time developer at Network Benchmarking Lab (NBL), NCTU. His research interests include software-defined networking, network function virtualization, and 5G mobile edge computing.

Ying-Dar Lin received the Ph.D. degree in computer science from the University of California at Los Angeles in 1993. He is a Distinguished Professor of Computer Science with National Chiao Tung University, Taiwan. He was a Visiting Scholar with Cisco System, San Jose, from 2007 to 2008. Since 2002, he has been the Founder and the Director of Network Benchmarking Laboratory, which reviews network products with real traffic and has been an approved test laboratory of the Open Networking Foundation (ONF) since 2014. He also cofounded L7 Networks Inc., in 2002, which was later acquired by D-Link Corp. He currently serves on the editorial boards of several IEEE journals and magazines. He published a book entitled Computer Networks: An Open Source Approach, with R.-H. Hwang and F. Baker (McGraw-Hill, 2011). His research interests include network security and wireless communications. His work on multihop cellular was the first along this line, and has been cited over 750 times and standardized into the IEEE 802.11s, the IEEE 802.15.5, the IEEE 802.16j, and 3GPP LTE-Advanced. He is an IEEE Distinguished Lecturer from 2014 to 2017 and an ONF Research Associate.

Po-Ching Lin received the B.S. degree in computer and information education from National Taiwan Normal University, Taipei, Taiwan, in 1995, and the M.S. and Ph.D. degrees in computer science from National Chiao Tung University, Hsinchu, Taiwan, in 2001 and 2008, respectively. He joined the faculty of the Department of Computer and Information Science, National Chung Cheng University (CCU), Chiayi, Taiwan, in August 2009. He is currently an Associate Professor. His research interests include network security, network traffic analysis, and performance evaluation of network systems.

Yuan-Cheng Lai received the Ph.D. degree in Computer Science from National Chiao Tung University in 1997. In August 2001, he joined the faculty of the Department of Information Management at National Taiwan University of Science and Technology where he had been a professor since February 2008. His research interests include wireless networks, network performance evaluation, network security, and Internet applications.