

Latency Measurement of Service Function Chaining on OpenStack Platform

Zuo Xiang*, Frank Gabriel*, Giang T. Nguyen*[†], and Frank H. P. Fitzek*

*Deutsche Telekom Chair of Communication Networks

[†]SFB-912 HAEC, Technische Universität Dresden, Germany

Email: {firstname.lastname}@tu-dresden.de

Abstract—Service Function Chaining allows the flexible and efficient deployment of network functions for different applications. With Network Function Virtualization the elements of the chain can be provisioned in virtual environments on any COTS hardware. This introduces the question of where to position the individual network functions within the virtualization environment. This problem of network function placement has been studied in theory as an optimization problem on a graph. However it is challenging to apply theoretical work on practical deployments. In this paper, we perform a measurement campaign to study the delay introduced by Service Function Chaining. We propose placement heuristics and evaluate the performance on OpenStack. With the proposed heuristics, the service delay can be reduced by more than 20%. We measured the overhead introduced by a network function implemented in user space. The processing delay in user space can be twice as much as the same function in kernel space. More interestingly, we identified a service interruption of more than 1 second after activation of the chain.

I. INTRODUCTION

Network Functions Virtualisation (NFV) [1] extends communications networks' capabilities and introduces new use cases. NFV resolves the bottleneck in legacy systems by removing the tight integration of middle-boxes running network Service Function (SF) from their hardware and leveraging advances in virtualisation technologies. Since practical use cases normally require a complete network service (NS) consisting of several SFs in a specific order, NFV has to be capable of forcing packets to traverse through them in the predefined order. That introduces the concept of Service Function Chaining (SFC).

One of the critical challenges in SFC provisioning is to decide where to launch SFIs to satisfy several objectives such as to increase resource utilization and to minimize SFC provision and service delays.

A substantial amount of research work in the literature has studied the optimization problem of efficiently placing SFIs to physical hosts [2], [3]. However, to the best of our knowledge, only a few of them were implemented in real-world cloud platforms. Subsequently, the practical applicability of the models to the current SOTA in orchestration and management of SFCs is uncertain. Furthermore, the lack of latency measurements on practical platforms is still an open issue.

In this paper, we perform a measurement campaign to understand the delays introduced by a SFC on an OpenStack cloud platform deployed on COTS hardware. we implement

heuristic algorithms to minimize latency and evaluate their efficiency. Our measurement results reveal the practical challenges in deploying Service Function Chaining with current technologies.

The rest of the paper is organized as follows. Section II discusses the state of the art in the literature. Section III introduces delay-aware Service Function Instances Placement and Chaining (SFIPC) heuristics. Section IV elaborates our measurement campaign, including the framework realization and the setup of our study. Afterwards, Section V describes and discusses measurement results. Finally, we conclude the paper and sketch our future work in Section VI.

II. RELATED WORK

A comprehensive introduction of general SFC frameworks and implementations is performed in [4]. These implementations are classified pursuant to utilized technologies (NFV and Software Defined Network (SDN)). B.Martini *et al.* [5] propose a service-oriented SDN controller to enable dynamic service chaining within the framework of service overlay networks. However, SFs are assumed as transparent and perform no packet modification. So the set-up duration and processing delay of SFs are not included in their results. StEERING is SFC framework proposed by Ying Zhang *et al.* [6] to provide flexible traffic routing through middleboxes based on SDN. A heuristic placement algorithm is also introduced to improve the SFC delay performance. The average latency of this greedy algorithm is evaluated and compared to a random deployment. João Soares *et al.* [7] presents the Cloud4NFV platform to orchestrate and manage SFs in a distributed telco cloud environment. The flexible composition of SFs is supported through SFC. However, performance measurements are not performed on their platform. A proof of concept is described to evaluate the benefits of the concept of SF-as-a-Service (SFaaS). However, this platform is relative complex for academic researches. For instance, the implementation of a new SFs placement algorithm on Cloud4NFV needs sufficient knowledge and experience of development on the OpenDaylight.

All in all, formulation of optimization problems in SFC related research needs to be simplified to facilitate the implementation and evaluation of delay-aware SF instance placement and chaining algorithms. Moreover, it is necessary to understand latencies of various phases in SFC instantiation

process to verify assumptions and to precisely pinpoint areas to improve SFC performance.

III. DELAY-AWARE SFIPC HEURISTICS

The problem addressed in this paper is the delay-aware Service Function Instances Placement and Chaining (SFIPC) problem: All SFIs of a logical SFC that are used to deploy VNFs need to be placed and then connected over a given physical infrastructure with various resource constraints including compute, network and storage, etc. The goal of delay-aware SFIPC solutions is to find a placement that minimizes both SFC rendering and service latency simultaneously. This section proposes three heuristic algorithms to solve this problem with low complexity.

When designing and optimizing heuristic greedy algorithms, the computing and networking in practical deployments should be considered. Thus, we make the following assumptions: *i)* Inter-node connections are more costly and time-consuming than intra-node connections. Packets of intra-node traffic can be forwarded directly by internal integration bridge. In comparison, packets of inter-node traffic need additional tunnel encapsulation to be transmitted through the underlying physical network. In order to reduce transmission delay, the number of inter-node connections should be minimized. *ii)* Instances on the same compute node have shared computing and networking resources including CPU, memory, storage and physical network interfaces. Allocating too many SFIs on the same compute node can cause it to overwork and significantly increase its processing delay.

Following heuristic approaches are designed and also implemented: *i)* Load Balancing (LB) strategy: is a iterative greedy algorithm that tries to minimize the processing time without considering the transmission delay. At each step of SFI placement, the compute node with minimal workload is always be selected. After all SFIs in the SFC have been allocated, they are chained with the same order of the placement. *ii)* Least Connection (LC) strategy: tries to minimize the transmission latency by recognizing the location of the service application and reducing inter-node connections at the same time. The physical node running the service application is firstly used to place SFIs. After selecting the node, as many SFIs as possible are placed to the selected node. When a node's resources are exhausted, the next node will be randomly selected for the placement. *iii)* Load Balancing with Least Connection (LBLC) strategy: is a optimized version of LB that take both transmission and processing latency into consideration. LBLC uses the same method as LB for SFIs placement. Instead of directly chaining allocated SFIs, a reordering is performed at first. This reordering is designed to minimize the connection between nodes. An example of using three algorithms to place four SFIs on two compute nodes is shown in Fig. 1.

IV. MEASUREMENT CAMPAIGN

A. Metrics

For a SF chain, the overall latency consist of three types i.e. rendering duration, gap duration and one-way delay as

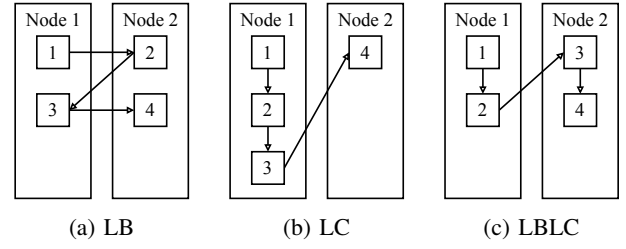


Fig. 1: An example of three implemented heuristic algorithms

depicted in Fig. 2. Those delays provide us a complete estimation of the SFC instantiation process, from preparing for a SF chain until the chain is actually in service: *i)* Rendering duration: characterizes the preparation duration to instantiate a service function chain. Since rendering process happens before SF chain starts working, this duration allows for better aligning expectation and precisely provisioning services. *ii)* Gap duration: captures the delay of the first packet passing through the chain to reach its destination. This gap duration exists because traffic policy and security rules applied only take effect when the first payload packet arrives. Even though this event happens only once for each newly created chain, it provides us a more accurate estimation on the delay of the chaining process. *iii)* One-Way Delay (OWD): This duration refers to the average delay of packets passing through the chain after it is completely ready, meaning after the rendering process completes and first packets pass the chain successfully.

B. Methodology

To measure the metrics defined in the previous section, we apply a combination of both passive and active measurement strategies. The passive strategy is used to measure the rendering duration which can be provided by a SFC manager. On the other hand, the active strategy is used to obtain the gap duration and one-way delay. We used UDP to send probing packets to have a full control of the sending process without being interfered by the connection establishment process as well as flow control and congestion control mechanisms of TCP. To synchronize the clocks of different machines, we set up a Network Time Protocol (NTP) server in the local network and synchronize all machines in our testbed's network.

For OWD measurements, the probing client marks packet ID and time stamp for each packet before sending it to the server. In order to measure the gap duration, the server must be able to distinguish between packets that have been processed with and without the SFC. Therefore, a SF that can modify UDP payload is used to replace original bytes and append timestamps of received packets.

C. Measurement setup

A research oriented SFC management and evaluation framework [8] is developed on OpenStack platform. It's built on top of OpenStack's official SFC extension [9]. Since OpenStack uses port as a connection point to attach a single instance to a network, this extension provides the port chain (PC)

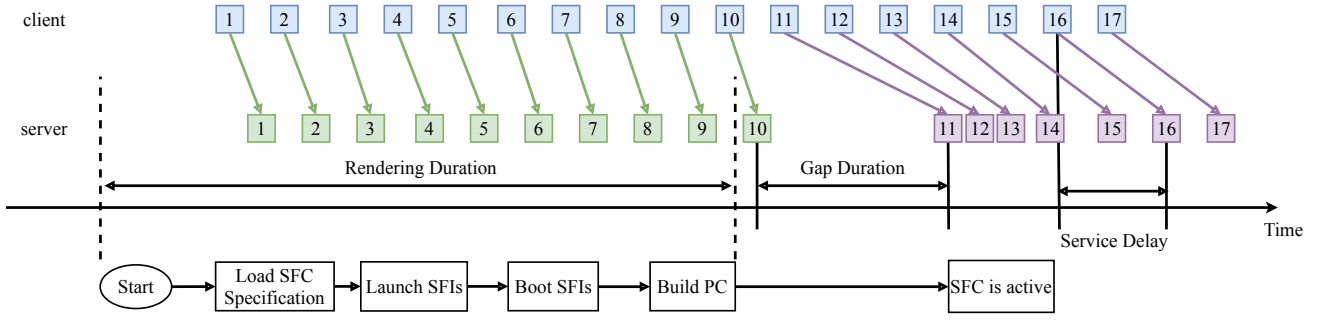


Fig. 2: SFC startup and service processes

resource for traffic redirection. SFIs are launched on several physical compute nodes. In our setup, SFIs only perform a few lightweight packet processing operations such as adding packet ID, time stamps and packet forwarding. We introduce only minimal functionalities because we would like to obtain measurement data that are independent from the complex packet processing of an actual SF. The packet forwarding functionality is performed in both kernel and user space. The Linux Kernel IP forwarding (or LKF) is chosen since it is one of the fastest SFs in the Linux kernel space. LKF is utilized to minimize the processing delay and as a baseline to evaluate the performance of other SF implementations. Additionally, we also implement the forwarding function in user space using Python scripts (or PyF). This provide us the more accurate estimation of the latency introduced by application. Finally, a probing interval of 4 ms and a UDP payload size of 512 bytes are used based on our estimation.

Our experiments were deployed on a physical testbed consisting of four COTS computers. All nodes were allocated with 4 CPU cores (Intel 4th Generation Core i5), 16 GB RAM and 128GB SSD. On top of the physical testbed, OpenStack Pike was installed on Ubuntu 16.04 operating system. In order to evaluate the impact of length of chain, $N(1 \leq N \leq 10)$ SFIs were created between the client and server topology introduced in IV-B. For each set of parameters, measurements were repeated 30 times to calculate the mean and 99.9% confidence intervals.

V. MEASUREMENT RESULTS

i) **Rendering delay:** Results of SFC rendering duration of different chain lengths are illustrated in Fig. 3a. Taking the chain with one SFI as an example, each bar contains three duration parts, namely the delay to start the SFI, boot the SFI and build the port chain (PC). While the SFI launching and booting duration are significantly dominant, the PC building duration accounts for a relative small proportion. Although the SFI launching and PC building durations show a linear growth trend, the booting duration remains almost unchanged and constitutes for roughly 100 seconds. It can be concluded that distributing SFIs over multiple compute nodes does not speed up the SFC rendering duration.

Gap time measurement results of three algorithms are depicted in Fig. 3b. Because calculated variances in results

are small enough, confidence intervals are not reported here for readability. According to the design of our SFC manager, traffic is redirected into the chain only after all resources show a fully active status. So this gap is expected to be relative small. However, the measurement results are obviously inconsistent with our expectations: The gap time is in the order of seconds. As shown in the figure, as the chain length grows, the gap duration of three algorithms as a whole presents an upward trend. For a SFC with 10 instances, the gap time of LB algorithm is more than 1 second. This means that the client will receive no packets for about one second. Taking the media streaming service as an example, the client has to pre-cache enough frames to avoid greatly reducing the quality of service. Moreover, different from rendering duration, the gap duration of three algorithms show obvious differences: Compared to the trend of fluctuations with LC algorithm, the LB and LBLC present a linear upward overall. While the LC algorithm always has the maximum gap duration when the chain length is smaller than 9, the LBLC has the best overall performance. For a chain of 10 SFIs, the gap duration of LBLC is less than 0.8 second compared to the duration about 1.1 second with LB. Therefore, the LBLC algorithm can reduce the gap duration by about 29% compared to the LB. It can be concluded that the LBLC algorithm can effectively accelerate the SFC initialization process.

Fig. 3c and 3d illustrate results of OWD measurements with Linux Kernel Forwarding (LKF) and Python Forwarding (PyF) used as SFs. As shown in the Fig. 3c, with the chain length ranging from 1 to 10, the OWDs of all three algorithms with LKF present an upward trend. Compared to the largest OWD presented by LB for all chain lengths, the LC algorithm has always the minimum delay. The results of LBLC are generally located between the LB and LC. The LC algorithm can reduce the service delay by about 20% compared to the LB for a chain with 10 SFIs. Since the LKF requires relative small processing time, the LC algorithm can significantly reduce the overall SFC transmission delay. Compared to LKF, Python forwarding is expected to require more processing time because it is implemented in user space and therefore calls for more computing resources. Therefore, the LBLC algorithm should have the best performance because it takes both transmission and processing delay into consideration. The measurement results shown in the 3d are obviously in line with

our expectations. Compared to LKF where LC has the minimal delay, LBLC presents the smallest delay when the chain length is bigger than 2. For a chain with 9 SFIs, LBLC has a minimal delay of 8.3 ms compared to LB 9.4 ms and LC 8.7 ms. The LBLC algorithm can reduce the service delay by nearly 10% compared to the LB strategy for a chain with 10 SFIs. From measurement results with PyF it can be concluded that the LBLC algorithm can provide a lower SFC service delay when both transmission and processing delay need to be considered.

VI. CONCLUSION

In this paper, we study the delays introduced by a SFC on OpenStack cloud platform. Heuristic algorithms are implemented for delay-aware placement problem. A measurement campaign is performed with the following observations: *i)* The one-way delay of packet passing a SFC can already be several milliseconds, even with minimum packet processing operations at kernel level. Whereas, being implemented in the user space, the same operation can double the delay. *ii)* With the proposed heuristics, the service delay of a packet can be reduced by more than 20%. *iii)* A service gap in the order of hundreds of milliseconds exists between the last packet without the service chain and the first packet passing through the chain. These observations reveal the practical challenges in deploying SFC with current technologies.

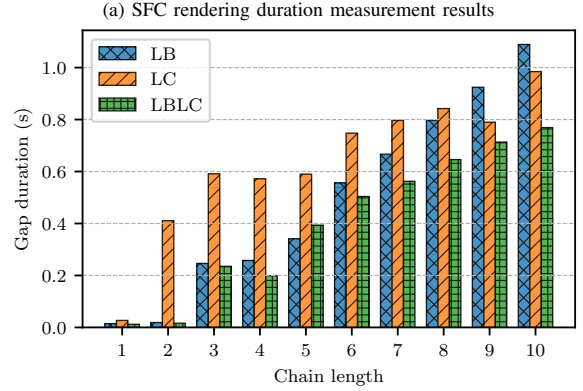
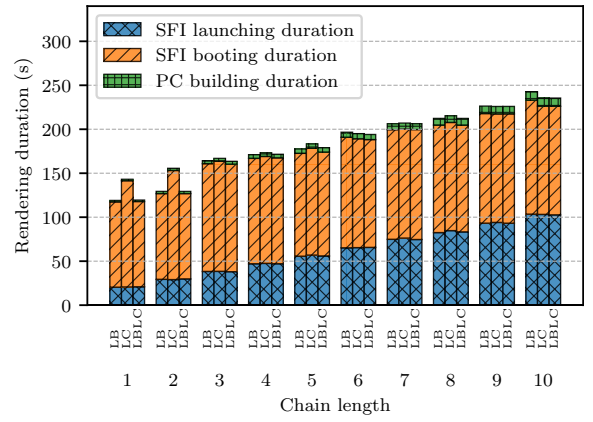
For future work, we plan to investigate the root cause of gap durations, extend our measurement study on a high-end physical testbed and an alternative software platform supporting SFC, such as a combination of container and SDN.

ACKNOWLEDGMENT

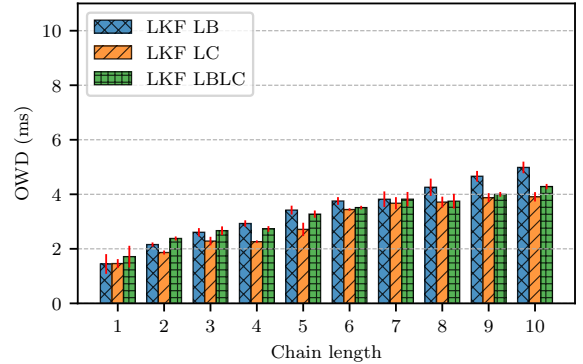
This work is supported in part by the German Research Foundation (DFG) within the Collaborative Research Center SFB 912 (HAEC).

REFERENCES

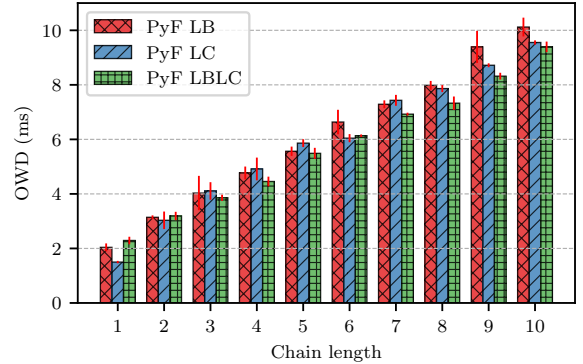
- [1] H. Hawilo, A. Shami, M. Mirahmadi, and R. Asal, "Nfv: state of the art, challenges, and implementation in next generation mobile networks (vepc)," *IEEE Network*, vol. 28, no. 6, pp. 18–26, 2014.
- [2] D. Bhamare, R. Jain, M. Samaka, and A. Erbad, "A survey on service function chaining," *Journal of Network and Computer Applications*, vol. 75, pp. 138–155, 2016.
- [3] J. G. Herrera and J. F. Botero, "Resource allocation in nfv: A comprehensive survey," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 518–532, 2016.
- [4] A. M. Medhat, T. Taleb, A. Elmangoush, G. A. Carella, S. Covaci, and T. Magedanz, "Service Function Chaining in Next Generation Networks : State of the Art and Research Challenges," pp. 2–9, 2016.
- [5] B. Martini, F. Paganelli, A. A. Mohammed, M. Gharbaoui, A. Sgambelluri, and P. Castoldi, "SDN controller for context-Aware data delivery in dynamic service chaining," *1st IEEE Conference on Network Software-Defined Infrastructures for Networks, Clouds, IoT and Services, NETSOFT 2015*, 2015.
- [6] Y. Zhang, N. Beheshti, L. Beliveau, G. Lefebvre, R. Manghirmalani, and R. Mishra, "StEERING : A Software-Defined Networking for Inline Service Chaining."
- [7] J. Soares, C. Gonçalves, B. Parreira, P. Tavares, J. Carapinha, and J. P. Barraca, "Toward a Telco Cloud Environment for Service Functions," no. February, pp. 98–106, 2015.
- [8] SFC-Ostack Source Code. [Accessed 2018-04-20]. [Online]. Available: {<https://github.com/stevetorenz/sfc-ostack>}
- [9] Service Function Chaining Extension for OpenStack Networking. [Accessed 2018-04-16]. [Online]. Available: {<https://docs.openstack.org/networking-sfc/latest/index.html>}



(b) SFC gap duration measurement results



(c) One-way delay when using Linux Kernel Forwarding



(d) One-way delay when using Python Forwarding

Fig. 3: Measurement results of different types of delays w.r.t. chain lengths.