Towards a Scalable, Flexible and High Performance NFV Execution Model

Peng Zheng*
Xi'an Jiaotong University
University of Minnesota
pzheng@umn.edu

Arvind Narayanan
University of Minnesota
arvind@cs.umn.edu

Zhi-Li Zhang
University of Minnesota
zhzhang@cs.umn.edu

ABSTRACT

Scaling the *software packet processing* capability of Network Function Virtualization (NFV) on multi-core commodity server to meet the high performance, *e.g.* 100 Gbps linespeed and beyond, is still a challenging task. We find that two existing service function chain (SFC) execution models often fail to scale up to the linespeed with increasing and diverse traffic. To this end, we present a novel hybrid execution model with the scalability, flexibility to attain high performance packet processing rate.

CCS CONCEPTS

• **General and reference → Performance**; • **Networks → Network performance analysis**; *Middle boxes / network appliances*;

KEYWORDS

NFV, multi-core, execution model, scalability, hybrid

ACM Reference Format:
Peng Zheng, Arvind Narayanan, and Zhi-Li Zhang. 2019. Towards a Scalable, Flexible and High Performance NFV Execution Model. In *The 15th International Conference on emerging Networking EXperiments and Technologies (CoNEXT '19 Companion), December 9–12, 2019, Orlando, FL, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3360468.3368181

1 INTRODUCTION

Network function virtualization (NFV) allows network operators to run (virtualized) network functions on commodity servers instead of dedicated hardware "middleboxes" such as firewalls, NATs and load balancers. This not only helps reduce the capital expenditure of carriers, but also provides them with the ability to dynamically scale out (or in) in accordance with traffic demands. Due to the slower *software* packet processing speed, scaling out via parallel packet processing on multi-core servers has been a major solution in design of software-based network functions [2]. Switches and servers with 100 Gbps ports and NICs are becoming cheaper and more commonplace. However, speeding up NFV packet processing pipeline to nearly the 100 Gbps linespeed and beyond on multi-core commodity servers is still a challenging task: while allocating more cores can improve the overall system throughput, the *uncore* resources such as the last-level cache (LLC), memory and I/O controllers shared among the cores becomes a bottle [3, 4]. In other words, NFV performance does not scale linearly with the number of cores. It is important to ensure most NFs and SFC operations are bound for the L1/L2 caches dedicated to individual cores so as to attain the 100 Gbps linespeed or higher performance [5].

Challenges. Two key factors make it a challenge to scale up NFV performance under multi-core server. First, most NFs are *stateful* which entails operations not only on packets, but also requires reads/writes to the NF state. The NF state competes with the packets to be processed for the L1/L2 cache resources. This problem is further compounded by the second factor: namely, a sequence of NFs – forming a so-called *service function chain* (SFC) – must often be performed on the packets to meet a network service policy. This not only increases the overall NF state cache resource requirements for SFC packet processing, but also poses the question how to execute NFs within an SFC.

Two existing execution models are employed in the literature [5]: the *pipeline* (PL) model allocates one core per NF and the packets are passed across the cores and processed by the NFs in a pipeline; the *run-to-completion* (RTC) model executes all the NFs in an SFC on a single core. These execution models have a strong impact on the *scaling* performance of SFC, due to the complicated interactions and resource contention among the NFs in an SFC and cores allocated to execute them. Both existing models have pros and cons and often fail to scale up to the linespeed. RTC avoids inter-core packet transfer penalty, but may suffer poorer cache locality and potentially longer per-packet processing latency, especially as the number of NFs increases. Note that we scale out SFC by allocating one core per the entire SFC under RTC model, as shown in Fig. 1 for an example 4-NF SFC, `ACL→NM→L4LB→L3FWD`, where the `L4LB` is the bottleneck NF in the SFC to be scaled. In comparison, PL staggers the per-packet processing latencies of different NFs in an SFC, but must pay the performance penalty of inter-core transfer latency: when switching from one core to another core, the packet being processed must be transferred via the shared L3 cache or worst case DRAM. In addition, due to more cores being used for processing each instance of an SFC, the number of cores available for scaling out is also reduced. On the other hand, PL provides better flexibility, for example, only scaling out the bottleneck NF `L4LB`, as shown in Fig. 2. This flexibility further decreases the negative interactions (*e.g.* the synchronization messages to ensure state consistency) for the `L4LB`'s upstream or downstream NFs instances running on different cores. For example, the scaling of `L4LB` using RTC model can introduce much more synchronization overhead to `NM` instances than using PL model, as marked blue in Fig. 1.

*The work was done while Peng Zheng was a visiting Ph.D student at the UMN.

CoNEXT '19 Companion, December 9–12, 2019, Orlando, FL, USA

ACM ISBN 978-1-4503-7006-6/19/12.
https://doi.org/10.1145/3360468.3368181

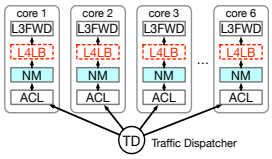

Figure 1: Scaling out using RTC

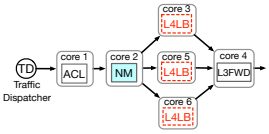

Figure 2: Flexible scaling out using PL

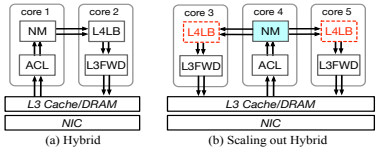

Figure 3: (a) An example of Hybrid model; (b) Scaling out using Hybrid

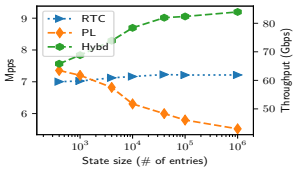

Figure 4: Scaling performance of three models

2 THE HYBRID APPROACH

To circumvent the limitations of existing SFC execution models, we present a *hybrid execution model* that combines the advantages of both RTC and PL models, while mitigating their limitations. By taking into account the NF states, the traffic affinities they induce as well as the complexity of individual NF operations, we partition a SFC into several sub-SFCs, where each sub-SFC is executed using the RTC model, and the sub-SFCs are processed in a pipeline fashion. Each sub-SFC is defined in such a manner to ensure that the the state needed by each NF in the sub-SFC for can be maintained in the L1/L2 caches most of the time to maximize the per-core packet throughput. By executing the NFs in each sub-SFC in a single core as in RTC, we minimize the inter-core transfer penalty. By dividing an SFC into multiple sub-SFCs, we can scale each sub-SFC independently to better account for the NF state requirements and traffic affinities while minimizing the negative interactions and synchronization overhead among them.

An example of Hybrid model. For the aforementioned 4-NF SFC, Fig. 3 (a) shows a simple hybrid execution model where we divide the SFC into two sub-SFCs: the ACL→ NM sub-SFC runs in a single core using RTC, and the L4LB→L3FWD sub-SFC runs in another core using RTC. Each SFC instance of this hybrid model will occupy 2 CPU cores. While scaling out, each sub-SFC in the Hybrid models can be independently scaled out to ensure flexibility. Fig. 3 (b) presents an example where the sub-SFC L4LB →L3FWD is scaled out. Comparing this hybrid model to the PL model, the Hybrid model saves one inter-core transfer, thus improving the overall throughput performance. Comparing with RTC executing the original 4-NF SFC on each core, the state memory requirement of the sub-SFC is smaller than that of the entire SFC (by an amount equivalent to the total state size of NM +ACL), thereby improving the cache locality for CPU cores due to the smaller state size; at the same time the hybrid model reduces the synchronization overhead between NM instances than RTC model. Next we evaluate our proposed Hybrid execution model and compare it with existing two execution models to better understand the scaling performance of SFC.

Preliminary Evaluation. Our testbed consists of two servers with Intel Xeon 8168 CPU @3.4GHz, each equipped with a dual-port Mellanox ConnectX-5 EN 100Gbps NIC. We have built a highly efficiency customized runtime system to manage SFCs and server resources. We implement the four NFs based on DPDK, which runs on one server. Another server, as traffic generator, runs TRex [1] to generate diversity workload with difference # of flows. The frame size is fixed to 1024 bytes.

We evaluate all three execution models using the same SFC aforementioned, and the same number of cores on the server under the same settings. Specifically, using 20 cores, we run 20 SFC instances for RTC and 5 SFC instances for PL; while for Hybrid model, we run 10 SFC instances as depicted in Fig. 3 (a). We set the state size (i.e. # of entries in the lookup table of each NF) from 400 to 1000k and we steer the same number of randomly generated flows for each instance in three models to measure the performance of different execution models. We use lock to ensure the correctness and consistency of state while scaling NF instances to multiple cores.

The performance of three execution models, under different workload, is shown in Fig. 4. Due to the space limitation, we briefly summarize our observations. With the increase of traffic diversity (the # of flows) and the required state size (# of entries), the Hybrid model shows the best scaling throughput and is significantly higher than the RTC and PL models. The PL model has the poorest performance due to the heavy inter-core transfer overhead. While minimal inter-core transfer under RTC, however, the state synchronization overhead becomes the bottleneck for scaling performance. The Hybrid model is able to tweak for a sweet point between PL and RTC by mitigating their drawbacks, thus leading to the best scaling performance for SFC running on multi-core commodity servers. To takeaway, through our initial evaluation and performance analysis, we demonstrate that the proposed hybrid execution model affords the network operators with the scalability and flexibility in dynamically provisioning the resources in a multi-core server to achieve the maximal system throughput.

3 CONCLUSION AND FUTURE WORK

Our work shows a novel hybrid execution model is promising for NFV, and provides a first step toward building a novel NFV execution model with high performance, scalability and flexibility. To fully develop such a framework, much work is still needed; these include mechanisms and algorithms for automatically profiling NFs, optimally partitioning SFCs for the hybrid model, dynamically distributing and balancing among (sub-)SFC instances in accordance with changing traffic dynamics and demands.

ACKNOWLEDGMENTS

The research is supported in part by US NSF under Grants CNS-1411636, CNS-1618339, CNS-1617729, CNS-1814322, and CNS-1836772. Peng Zheng gratefully acknowledges financial support from National Key R&D Program of China (2017YFB0801703), the NSFC (61672425) and China Scholarship Council.

REFERENCES

[1] 2019. Cisco T-Rex: Realistic traffic generator. https://trex-tgn.cisco.com
[2] Mihai Dobrescu et al. 2009. RouteBricks: Exploiting Parallelism to Scale Software Routers. In *ACM SOSP'09.*
[3] Mihai Dobrescu et al. 2012. Toward Predictable Performance in Software Packet-Processing Platforms. In *USENIX NSDI'12.*
[4] Amin Tootoonchian et al. 2018. ResQ: Enabling SLOs in Network Function Virtualization. In *USENIX NSDI'18.*
[5] Peng Zheng et al. 2019. A Closer Look at NFV Execution Models. In *ACM APNet'19.*