`Octans`: Optimal Placement of Service Function Chains in Many-Core Systems

Heng Yu, Zhilong Zheng, Junxian Shen, Congcong Miao, Chen Sun, Hongxin Hu, *Member, IEEE*, Jun Bi, *Senior Member, IEEE*, Jianping Wu, *Fellow, IEEE*, and Jilong Wang, *Member, IEEE*

Abstract—Network Function Virtualization (NFV) offers service delivery flexibility and reduces overall costs by running service function chains (SFCs) on commodity servers with many cores. Existing solutions for placing SFCs in one server treat all CPU cores as equal and allocate isolated CPU cores to network functions (NFs). However, advanced servers often adopt Non-Uniform Memory Access (NUMA) architecture to improve the scalability of many-core systems. CPU cores are grouped into nodes, incurring performance degradation due to cross-node memory access and intra-node resource contention. Our evaluation shows that randomly selecting cores to place NFs in an SFC could suffer from 39.2 percent lower throughput comparing to an optimal placement solution. In this article, we propose `Octans`, an NFV orchestrator to achieve maximum aggregate throughput of all SFCs in many-core systems. `Octans` first formulates the optimization problem as a Non-Linear Integer Programming (NLIP) Model. Then we identify the key factor for problem solving as evaluating the throughput drop of an NF caused by other NFs in the same SFC or different SFCs, i.e., *performance drop index*, and propose a formal and accurate prediction model based on system level performance metrics. Finally, we propose two online algorithms to quickly find near-optimal placement solutions for one-time and incremental deployment. Extensive evaluation on a prototype implementation shows that `Octans` significantly improves the aggregate throughput comparing to two state-of-the-art placement solutions by 27.1~45.2 percent for one-time deployment and by 20.9~38.1 percent for incremental deployment, with very low prediction errors. Moreover, `Octans` could quickly find a near-optimal placement solution with tiny optimality gap.

Index Terms—Many-core system, network function virtualization, service function chain, optimal placement

1 Introduction

Network Function Virtualization (NFV) was recently introduced to address the limitations of traditional middleboxes. NFV runs network functions (NFs) on commodity servers with general-purpose processors such as Intel x86 to improve service delivery flexibility and reduce overall costs. In NFV, packets are usually processed by a sequence of NFs, which form a *service function chain (SFC)*.

Nowadays, to provide high throughput and simplify equipment management in data center, commodity servers used in NFV are often high-performance and high-density with *multiple CPU cores* [1], [2], which we refer to as *many-core systems*. One such server has the capability of accommodating an entire SFC or even multiple SFCs [3], [4]. In this situation, current solutions for the *placement* of NFs in one server is to treat all CPU cores as equal, and allocate isolated CPU cores to different NFs, in order to avoid performance degradation for NFs [3], [5].

However, above solutions overlook the fact that CPU cores in a many-core system are actually *unequal*. Using different core sets to support the same SFC could result in significantly different throughput. This is because many-core systems today usually adopt Non-Uniform Memory Access (NUMA) architecture for high scalability [6]. Fig. 1 shows a typical architecture of the Intel x86 many-core system. CPU cores are grouped into *nodes*. Each node contains multiple cores and its own local memory. Randomly selecting CPU cores to support an SFC could suffer from seriously compromised throughput due to the following two reasons.

(1) *Bottleneck incurred by cross-node memory access.* When packets come into server through NIC, they are stored in the memory which located at the same node of the NIC. While CPU cores in other nodes can access the memory via Intel QuickPath Interconnect (QPI), local memory access inside one node is much faster than remote access. To study its effect on the performance of NFV, we place a simple SFC (*Router* $\rightarrow$ *NIDS*) on a many-core system with two nodes in four ways shown in Fig. 2a and evaluate their performance when packets come in through Node #0. As illustrated in Fig. 2b, the performance of worst-case placement (i.e., P-B) achieves less throughput than the best-case placement (i.e., P-A) by 39.2 percent. An intuitive solution is to place all NFs in an SFC in the same node to avoid remote memory access. However, the number of cores in one node is limited.

- *Heng Yu, Junxian Shen, Congcong Miao, Jianping Wu, and Jilong Wang are with the Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing 100084, China, with the Beijing National Research Center for Information Science and Technology (BNRist), Beijing 100084, China, and also with the Peng Cheng Laboratory, Shenzhen, Guangdong 518066, China. E-mail: {yuheng20, shenjx19}@mails.tsinghua.edu.cn, jamesmiao@tencent.com, {jianping, wjl}@cernet.edu.cn.*
- *Zhilong Zheng and Chen Sun are with the Alibaba Group, Hangzhou 310052, China. E-mail: {zhiyou.zzl, qichen.sc}@alibaba-inc.com.*
- *Hongxin Hu is with the Department of Computer Science and Engineering, University at Buffalo, the State University of New York, Buffalo, NY 14260 USA. E-mail: hongxinh@buffalo.edu.*
- *Jun Bi is with the Beijing National Research Center for Information Science and Technology, Beijing 100084, China. E-mail: junbi@tsinghua.edu.cn.*

Manuscript received 29 Aug. 2019; revised 12 Feb. 2021; accepted 17 Feb. 2021. Date of publication 3 Mar. 2021; date of current version 17 Mar. 2021.
(Corresponding authors: Congcong Miao and Jilong Wang.)
Recommended for acceptance by W. Yu Ph.D.
Digital Object Identifier no. 10.1109/TPDS.2021.3063613

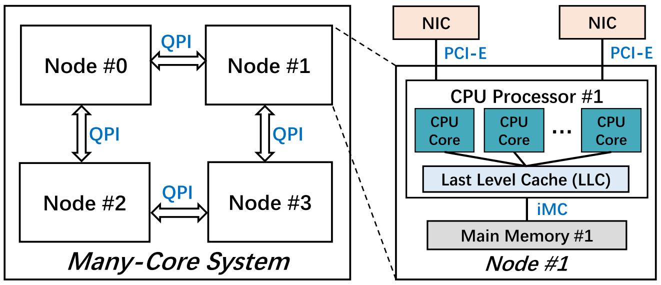


Fig. 1. A high-level view of a typical many-core system.

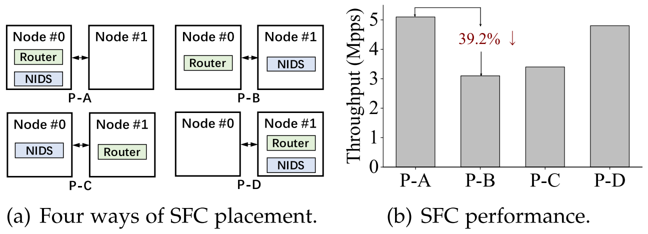


Fig. 2. Effect of remote memory access on SFC throughput. The Network Interface Card (NIC) is connected to Node #0. So we call Node #0 *local node*, and Node #1 *remote node*.

An SFC may be placed across multiple nodes for better resource utilization [1]. Moreover, we can observe from Fig. 2b that even switching the node assignment of two NFs (P-B and P-C) could lead to different SFC performances. This is because different NFs expose different performance sensitivity to cross-node memory access.

(2) *Bottleneck incurred by intra-node resource contention.* As shown in Fig. 1, CPU cores in each node may contend to shared resources such as last-level cache (LLC), integrated memory controller (iMC), and QPI. Some recent research efforts [7], [8], [9] have revealed that co-locating multiple NFs at the same node could decrease the throughput of a single NF by 12.4 to 50.3 percent due to resource contention.

Above observations motivate us to design an optimal mechanism to place multiple SFCs in a many-core system, with the goal of achieving *maximized aggregate throughput of all SFCs*. Many research efforts have been devoted to addressing the placement problem in NFV [10], [11], [12]. However, they all focused on finding right servers for SFC placement, while ignoring the placement inside one server. Meanwhile, the problem of placing multiple threads in many-core systems to achieve optimal execution performance has been well studied in the system and architecture community [13], [14], [15], [16]. However, they relied on frequent migration of threads for optimality, while migrating NFs in the NFV context incurs significant performance overhead [17]. We present more details in Section 2.

In this paper, we propose `Octans`, an NFV orchestrator for optimal SFC placement inside a server. To obtain the key factors for optimization, we start by formulating the problem with a Non-Linear Integer Programming (NLIP) model. Then we identify the key factor for problem solving as evaluating how much the throughput of an NF could be affected by NFs in the same SFC or different SFCs, which we refer to as *performance drop index*. This task imposes unique challenges from two aspects: (1) to evaluate the throughput drop of NFs caused by resource contention, we are challenged to find an *unified* set of system-level metrics that could represent the property of *massive and heterogeneous NFs* in NFV; and (2) NFs with different chaining methods and NICs could introduce different types of shared resources, which will affect the aggregate performance of SFCs. We are challenged to precisely model NF performance under the SFC and multiple NICs context. To address above challenges, we use a formal approach to find performance metrics and construct a comprehensive model for accurate performance drop prediction. Finally, due to the NP-hardness of our problem, we propose a heuristic algorithm to quickly find an optimal or near-optimal deployment solution for multiple SFCs. Besides, we propose a greedy algorithm to support incremental deployment of SFCs.

`Octans` makes the following major contributions:

- We identify the problem of SFC placement in a many-core system, and present `Octans`, an NFV orchestrator, to maximize aggregate throughput of SFCs. We introduce related work and highlight the novelty of `Octans` (Section 2).
- We formulate the optimization problem using an NLIP model (Section 3). To evaluate the performance drop index due to cross-node memory access and intra-node resource contention, we introduce a formal approach to find performance metrics, and present an accurate model for performance drop prediction. Finally, we design two online placement algorithms to efficiently produce an optimal or near-optimal solution for one-time and incremental deployment (Section 4).
- We introduce the architecture and workflow of `Octans` (Section 5). Extensive evaluation results show that `Octans` can achieve reasonably prediction results for different NFs (2.3 percent prediction error on average) and different numbers of SFCs with varied lengths (2.5 percent prediction error on average). Moreover, `Octans` can improve the aggregate performance comparing to two alternative placement mechanisms by 27.1 to 45.2 percent for one-time deployment and 20.9 to 38.1 percent for incremental deployment. Finally, `Octans` has a high chance (50~77 percent) to find an optimal deployment solution in a short time (Section 6).

2 Related Work and Octans Novelty

This section summarizes state-of-the-art researches on SFC placement in NFV, optimal thread scheduling in many-core systems, as well as works that touch upon the problem of optimal NF placement on many-core systems. However, to the best of our knowledge, `Octans` is the first to formally model the optimization problem, thoroughly study performance variation of NFs in many-core systems, and propose two efficient algorithms to solve the problem.

SFC Placement in NFV. Many efforts have been devoted to NF and SFC placement in NFV [18], [19], [20], [21], [22]. Moens *et al.* [10] presented an Integer Linear Programming (ILP) model to minimize the number of used servers in a service provider network. Kuo *et al.* [12] jointly considered NF placement and chaining across servers to better utilize network resource. Zhang *et al.* [20] jointly optimize SFCs placement and request scheduling for resource utilization and average latency. Li *et al.* [23] focused on providing guaranteed performance for NFs by placing NFs in the right server. Xiao *et al.* [18] utilize deep reinforcement learning to optimize SFCs placement among multiple servers.

However, all existing works optimized SFC placement by mapping NFs to the right *servers*. Regarding SFC placement within a server, above works treated all CPU cores as equal, and did not consider the widely adopted NUMA architecture in modern many-core systems. In contrast, `Octans` studies the problem of placing multiple SFCs in a many-core system, in order to maximize aggregate throughput of all SFCs in a server. Although cores inside one node can be further split into groups (e.g., two logic cores share one physical core), we disable hyper threading and ignore this fact in our paper. `Octans` deeply understands NF performance with respect to cross-node memory access and intra-node resource contention. Therefore, `Octans` could work with existing works to map right NFs to right *CPU cores*.

Optimal Thread Scheduling in Many-Core Systems. Optimizing thread-to-core placement to maximize execution performance in a many-core system has been extensively studied in the system and architecture community [13], [15], [24], [25], [26], [27]. However, they are designed for different purposes. For instance, Sanidhya Kashyap *et al.* [25] try to implemente scalable blocking synchronization primitives. Justin Funston *et al.* [24] targeted at improving the performance of virtual containers. Besides, the key idea of most above works is to measure the system performance metrics (e.g., LLC misses and memory load) at runtime, and *dynamically re-schedule* co-locating threads to different cores for optimal thread-to-core mapping. For example, Zhuravlev *et al.* [13] proposed dynamically scheduling co-locating threads according to the change of LLC misses. Rao *et al.* [15] suggested utilizing more metrics such as data locality and sharing overhead, and converted them into a unified parameter for optimal VM scheduling in a server.

A natural question is *whether above solutions can be directly adopted to solve the SFC placement problem*. Our answer is *no*. Above solutions achieve optimal thread execution performance by frequently migrating threads across cores or nodes. While threads are light-weight and easy to migrate, most NFs in NFV are stateful and suffer from significant performance overhead during NF migration [17], [28]. To achieve high performance, advanced NFV systems place NFs on dedicated CPU cores that cannot be scheduled by the operating system. This makes it difficult to migrate NFs among cores. Besides, the placement policy of normal OS (e.g., Linux [29]) does not consider NFV related features (e.g., SFC) and could lead to poor performance under most conditions. Therefore, `Octans` maps NFs to cores by designing a *static placement* mechanism [30]. `Octans` thoroughly investigates and models NF performance in a many-core system and proposes two efficient placement algorithms to maximize aggregate SFCs throughput.

SFC Placement in Many-Core Systems. Some recent researches [31], [32], [33], [34] have revealed that placing NFs on different cores in a NUMA system could result in different performance. Sieber *et al.* [32] reported this problem by presenting several evaluations in an NFV environment. However, they did not present any solution for optimal placement. Wang *et al.* [31] presented a *locality-first-mapping* algorithm by placing an entire SFC in one node to avoid cross-node overhead. However, the impact of intra-node contention was not considered. Hu *et al.* [33] investigated how the performance of *pipelined* software components varies when they are placed on different cores. However, their solution is also based on dynamically scheduling, which could be impractical in real-world NFV systems as we discussed above. Li *et al.* [34] proposed how to share CPU cores among NFs and tune CPU quota for each NF at edge cloud. However, our target scenario is high performance NFV in data center and each NF monopolizes a CPU core.

3 Problem Formulation and Challenges

In this section, we first formulate the problem with a NLIP model. Then we identify the key factors for problem solving as evaluating NF performance drop index, and introduce the challenges in retrieving this index in many-core systems.

3.1 Formulation of the Optimal Placement Problem

Placement Requirement of SFCs. Placement requirement of an SFC is usually described as an ordered sequence of NFs in the chain. Assume there is a set of SFCs S that require to be deployed, each of which is associated with an array of chained NFs $e(i)$, where $i \in S$.

A Many-Core System. Commonly in a many-core system, there are multiple nodes numbered incrementally and equipped with identical amounts of CPU cores. Besides, each node could be independently equipped with zero or multiple NICs. We generalize a many-core system with K nodes, each node has N cores and aggregated throughput of $C(k)$ from its NICs, where $k \in K$.

Performance Decomposition. We set a binary variable x_{ij}^k to indicate whether NF j of chain i is located on node k. When this NF runs without contention, the performance it can achieve is referred to as *ideal performance* and is defined as P_{ij}^k. As introduced in some work [7], [23], P_{ij}^k of each NF can be measured by placing the NF on different nodes when no contention exists. Furthermore, when an NF co-locates with other NFs in node k, its performance is referred to as *interfered performance* and is defined as ϕ_{ij}^k. It is obvious that the interfered performance of an NF is a reduced value of ideal performance. Thus, we define a *performance drop index* (denoted by λ_{ij}^k) to relate these two variables for NFs that locate on node k (shown in Eqn. (2)). Note that λ_{ij}^k is not a constant and varies with different SFCs placement solutions.

Similar to an end-to-end system, the processing capacity of an SFC is determined by the *bottleneck element* in the chain [35], i.e., the NF with the lowest performance. Hence, the objective to maximize the aggregate performance across all required SFCs when deploying is formulated as

$$max \sum_{i \in S} \min_{j \in e(i)} \{\phi_{ij}^k\}, \quad \forall k \in K, \tag{1}$$

where

$$\phi_{ij}^k = \sum_{k \in K} x_{ij}^k \cdot P_{ij}^k \cdot (1 - \lambda_{ij}^k), \quad \forall i \in S, j \in e(i), \tag{2}$$

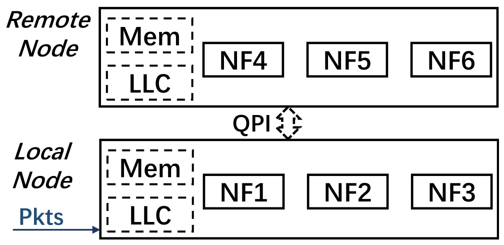


Fig. 3. A placement example on two nodes.

s.t.

$$\sum_{k\in K} x_{ij}^k = 1, \quad \forall i \in S, j \in e(i) \tag{3}$$

$$\sum_{i\in S, j\in e(i)} x_{ij}^k \le N, \quad \forall k \in K \tag{4}$$

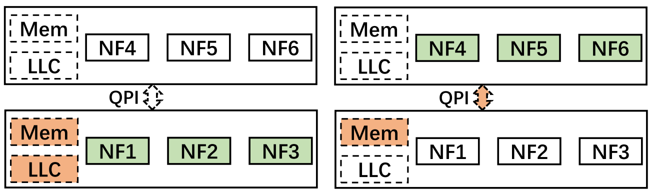
$$\sum_{i\in S} \phi_{i1}^k \le C(k), \quad \forall k \in K. \tag{5}$$

More specifically, Eqn. (2) describes the relationship between ideal performance and interfered performance. Constraint (3) prevents any NF from being repeatedly or not deployed. Constraint (4) specifies the maximum deployed number of NFs cannot excess the capacity of cores in each node during deployment. Constraint (5) specifies the maximum aggregated throughput of SFCs receive from NICs of one node cannot excess the capacity of NICs on that node during deployment.

3.2 Key Factors for Problem Solving and Challenges

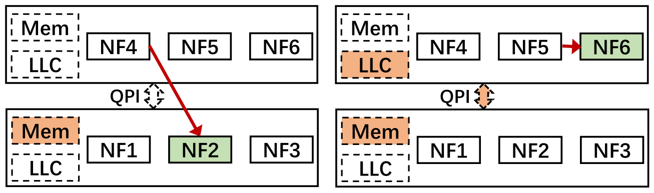
Problem formulation gives us a solid starting point for finding optimal placement solution for S. However, a critical parameter, i.e., λ_{ij}^k needs to be predicated for problem solving since we cannot try all possible placement solutions to directly get their value. However, predicting performance drop in the NFV context is not-trivial due to the following three challenges.

The Heterogeneity of NFs. As introduced in Section 2, NFs are usually stateful and deployed on dedicated cores, which reveals that the placement should be static to avoid distractions from other complexities, such as state migration and careful packet buffering design. For the prediction of static placement, the inputs we have are only the types and counts of NFs (i.e., which NFs and how many of them) that might be co-located. However, it is tricky to adopt these inputs



(a) Shared resources on local node. (b) Shared resources on remote node.

Fig. 4. Shared resources for separated NFs, which are highlighted with orange-red.



(a) Shared resource changes due to cross-node chaining. (b) Shared resource changes due to cache-prefetching chaining.

Fig. 5. Different shared resources for the same NFs because of different SFC methods.

directly for the prediction model since they are not quantifiable. Some work [7] shows that we can use a manually analyzed and calculated system-level performance metric value (i.e., LLC references per packet) to quantify an NF. However, manual analysis and metric value calculation could be burdensome and platform dependent. Furthermore, this problem could be worse in the NFV context because NFs are usually heterogeneous and diverse.

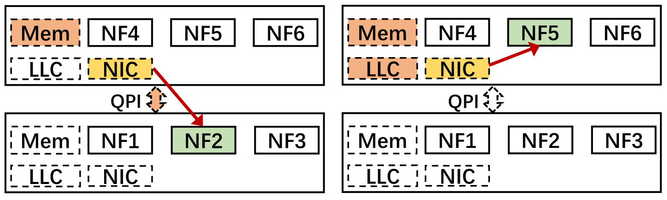
SFC Complicates the Prediction. Performance drop of an application (NF) in an interference environment is usually related to the shared resources contention and its competitors [13], [15]. However, the chain of an SFC could change the shared resources and competitors of an NF, in contrast to which NFs are separated.

For example, consider six co-locating NFs on two nodes (the nodes where packets come in are usually called local nodes, others are relatively remote nodes), as shown in Fig. 3. For separated NFs, of which *(i)* in the local node can be considered as sharing LLC between NF1~NF3 and main memory controller between NF1~NF6 (Fig. 4a), and *(ii)* in the remote node can be considered as sharing QPI between NF4~NF6 (Fig. 4b). We can see that NFs in the same node have the same type of shared resources and competitors, thus one general model for these NFs might be accurate [7].

However, this model could be inaccurate when NF chaining is introduced. Fig. 5a shows that *(iii) cross-node chaining* between two NFs can change shared resources of the subsequent NFs (i.e., NF2) in a local node to be more stressed on memory controller and almost no LLC benefit, which could incur performance drop. Moreover, Fig. 5b shows that *(iv) cache-prefetching chaining* can change shared resources of the subsequent NFs (i.e., NF6) in a remote node to be less competing on QPI and benefit the performance improvement from memory cache hits in the remote LLC.

Multiple NICs Complicate the Prediction. In a server with multiple NICs which could locate at different nodes, the traffic of an NF could come from different NICs. However, the way of traffic entering a server can also change the shared resources and competitors of an NF. For instance, we consider the example in Fig. 4 except that both nodes are equipped with NICs. Since the NIC stores traffic to memory of the same node, both nodes could become local node for some NFs at runtime. If the traffic of an NF comes from another NIC, such as NF2 in Fig. 6a, it no longer benefits from the LLC and competes for QPI and memory controller. Similarly, as shown in Fig. 6b, the competing resources of NF5 also changes to LLC and memory controller.

Therefore, we can see that more specific considerations are required in the prediction model due to the possible



(a) Shared resource changes for NF2. (b) Shared resource changes for NF5.

Fig. 6. Different shared resources for the same NFs because of different NICs.

changes on shared resources and competitors introduced by SFC and multiple NICs.

Octans. To address above challenges, we propose Octans. For performance prediction, Octans takes the performance metric of each NF as the input of prediction model as well as considering the specific features introduced by SFC and multiple NICs. Moreover, Octans provides two placement algorithms to search optimal or near-optimal solutions for online deployment, including a heuristic-based algorithm for quickly one-time deployment and a greedy algorithm for incremental deployment. Next, we will introduce the design of Octans in detail.

4 Octans Design

In this section, we present the design of Octans. We first show how to automatically find performance metrics for NFs even they are heterogeneous, and what metric is used in Octans (Section 4.1). Then, we introduce the performance prediction model for NFs and extend it to SFC and multiple NICs (Section 4.2). Finally, we present two online placement algorithms for one-time and incremental deployment (Section 4.3).

4.1 Relating NFs With Performance Metrics

As mentioned in Section 3, the type of NFs cannot be directly used as the input of prediction model. Instead, we should find some performance metrics to describe an NF. Moreover, as shown in Table 1, many potential system-level metrics can be adopted [15], [38]. Therefore, we should formally and automatically detect the appropriate metrics with some criterions. To achieve this goal, we first identify the requirements these metrics should meet.

R1: The metric value should not vary even with contention. Since the metric we try to find should be able to describe an NF, its value should not vary when co-locating with other NFs. For example, if a metric value is measured as v_0 when an NF runs solely, all values ($v_1 \sim v_6$) measured when co-locating with other 1~6 NFs should be near to v_0.

R2: The metric value should be sensitive to NF changes. This describes an intuition that different NFs should have different metric values. Here, we make a mild assumption that the NFs with similar intrinsic properties such as program complexity can achieve similar performance. Therefore, if the measured performance and metric values of a set of NFs when they run solely are $(p^1, p^2, p^3, \ldots)$ and $(v^1, v^2, v^3, \ldots)$, and $p^1 \neq p^2 \neq p^3$, the metric values should be approximately regarded as $v^1 \neq v^2 \neq v^3$.

TABLE 1
A Non-Exhaustive List of Commonly Used Metrics, Which can be Profiled by Existing Tools Such as OProfile [36] and Intel PCM [37]

Metric	Description
CPU_CLK_UNHALTED	Clock cycles when not halted
INST_RETIRED	Number of instructions retired
RESOURCE_STALLS	Cycles during which resource stalls occur
LLC_MISSES	Cache misses in LLC
L2_MISSES	Cache misses in Level 2 cache
IPC	Instructions per CPU cycle

With the help of above two requirements, we can use a general but formal way to find metrics. We re-interpret the two requirements as,

- *Near-zero variance of metric values when measured in competing environment:* Guided by R1, for each candidate metric, we calculate the variance (denoted by ρ_i^2) of its measured values when co-locating an NF with different types and numbers of NFs, i.e.,

$$\rho_i^2 = \frac{\sum_{j=1}^{N}(m_{ij} - \mu_i)^2}{N}, \tag{6}$$

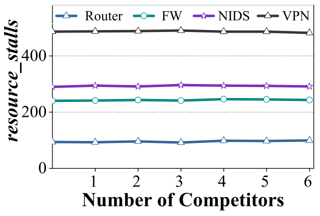
  where, m_{ij} is the jth sample of the value of metric i and μ_i is the mean value of all N samples. Moreover, all values used in this equation are normalized to [0,1].

- *Strong correlation between metric value and performance when measured in non-competing environment:* Guided by R2, there should exist strong correlation between metric value and performance when an NF runs solely (i.e., ideal performance). For concise expression, we abuse P_j to define the ideal performance of NF j. Some work [16], [38] has demonstrated that there is *linear correlation coefficient* between the value of some metrics and the performance. In our work, we follow their findings and assume this linear correlation. We adopt Pearson Correlation Coefficient [39] to calculate the correlation (denoted as $r(P_j, m_i)$), i.e.,

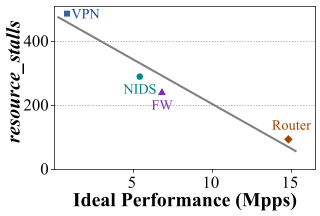
$$r(P_j, m_i) = \frac{cov(P_j, m_i)}{\sigma_{P_j}\sigma_{m_i}}, \tag{7}$$

  where, $cov(P_j, m_i)$ is the cross-correlation of NF j between the ideal performance and metric i, and $\sigma_{P_j}^2 = cov(P_j^2)$, $\sigma_{m_i}^2 = cov(m_i^2)$ are the variances of different measured values of performance and metric.

Metrics Used in Octans. We measured a bunch of metrics and performance values according to Eqns. (6) and (7). The candidate metrics we used are from two performance metric profiling tools, i.e., OProfile [36] and Intel Performance Counter Monitor (PCM) [37]. Table 1 shows a subset of these metrics. Since some metric values are accumulated from NF launching (i.e., inst_retired and resource_stalls), they are divided by the number of processed packets. We empirically



(a) Validation for R1.



(b) Validation for R2.

Fig. 7. Requirement validation for the metric *resource_stalls*.

set $\rho_i^2 \leq 0.01$ as *near-zero variance* and $\left|r(P_j, m_i)\right| \geq 0.9$ as *strong correlation*. Metrics that meet both conditions are considered as appropriate ones.

Experiments based on sampled data reveal that the appropriate metric is *resource_stalls*. Note that we only choose the best metric for easier modeling. Nevertheless, we find that it is enough for our model to achieve accurate prediction (Section 6.2). Besides, we check whether this metric meets the two requirements defined before. Fig. 7a shows that for all four NFs, *resource_stalls* remains almost unchanged as the number of competing NFs increases, which meets the first requirement. Fig. 7b shows that *resource_stalls* and the ideal performance have near linear relationship, which can meet the second requirement. Furthermore, we try to understand this metric from an architecture perspective. Since this metric records the cycles of core resource related stalls (i.e., lacking buffer for load/store instructions and branch misprediction recovery), it does not vary due to contention of shared resource (i.e., LLC and memory controller) and is highly related to intrinsic program structure and complexity of NFs. Note that this metric shows per packet resource stalls, it is not affected by the incoming packet rate. Therefore, we choose *resource_stalls* to represent different NFs, and use it as the input of our prediction model.

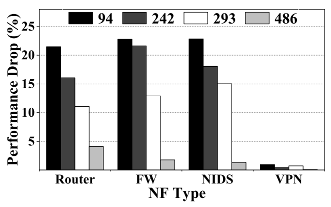
4.2 Prediction Modeling

4.2.1 Preliminary Analysis of the Prediction Model

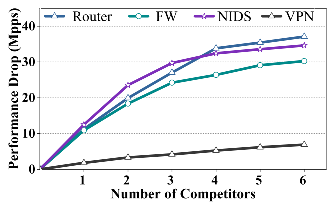
Contention to shared resources is the root cause of performance degradation for co-locating NFs in a server [7], [15], [38]. According to observations from Section 3.2, we classify competing resources into two types of factors that could cause performance degradation including LLC misses and memory contention (including memory controller and QPI). Moreover, we utilize the system-level metric of NFs (i.e., *resource_stalls*) to predict performance drop index λ (we reuse λ here for brevity). Next, we first show how to quantify the competing level introduced by NFs under one type of competing resource. Based on that, we provide the prediction model for λ with multiple competing resources under NUMA architecture. Then, we refine the prediction model to improve prediction accuracy according to SFC characteristics. Finally, we extend the prediction model to a server with multiple NICs.

4.2.2 Prediction Model for Separate NFs

To predict the performance drop for separate NFs, we begin with the simplest prediction model with one type of shared resource. Then we extend this prediction model to multiple



(a) Performance drop with different co-locating NFs (represented by *resource_stalls*).



(b) Performance drop with different number of competitors.

Fig. 8. Performance drop with different NF type and number.

competing resources. Finally we provide a general prediction model for separate NFs under NUMA architecture.

Calculating the competing level introduced by NFs under one type of shared resource. We define a competing level function f to map the metric value of competing NFs to performance drop. To explore this function, we first check the relationship between the competing level and metric value, i.e., how the competing level varies when the metric value changes. It is hard to directly capture the change of competing level. Instead, we use the performance drop to imply it. Fig. 8a shows that for different NFs, a smaller *resource_stalls* (NF with higher ideal performance) incurs higher performance drop, i.e., higher competing level. From this observation, we define a property for this function,

- *Property 1:* The function shows *negative correlation* to *resource_stalls*.

Furthermore, we check that how the competing level varies with the number of NFs changes, i.e., aggregate *resource_stalls*. Fig. 8b shows that with the number of competing NFs increases, the performance drops more, i.e., higher competing level. Moreover, we can observe that the increasing gradient of performance drop becomes smaller. Therefore, we define another property for this function,

- *Property 2:* The function shows *increment property and decreasing gradient* with aggregate *resource_stalls* increases.

With the help of above two properties, we can approximately describe the competing level function as

$$f(\{x_1, x_2, \ldots, x_n\}) \approx \sum_{1 \leq k \leq n} \frac{1}{x_k}. \tag{8}$$

It can be easily proved that the function in Eqn. (8) meets the two defined properties. Although this function seems simple and intuitive, our evaluation (Section 6) shows it is accurate enough to describe the competing level in prediction model.

Modeling performance drop index under multiple competing resources. According to [38], in an interference environment, we could model the performance drop as a linear function of different competing resources. Therefore, we define λ as

$$\lambda = \alpha \cdot f(\{m^i | i \in NF_{LLC}\}) + \beta \cdot f(\{m^i | i \in NF_{mem}\}) + C, \tag{9}$$

where, NF_{LLC} and NF_{mem} indicate the competitors on LLC and memory. α, β and C are parameters to aggregate the effect of competing level on LLC and memory. Note that even with the same competing level in share resource, the

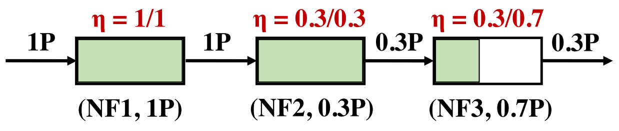


Fig. 9. Throughput limitation imposed by SFC. P represents the performance unit for NFs.

performance of some NFs drops a lot while others almost not be affected. Therefore, we use different parameter groups for different type of NFs to distinguish their sensitivity to competing resources.

A general prediction model for separate NFs under NUMA architecture. Since every NF could be placed on all nodes, there should be distrinct parameter groups for the local and remote nodes due to the asymmetric cost of LLC misses and memory access [14]. Thus, we define (α_l^i, β_l^i, C_l^i) for the case that NF i is deployed in the local node, and (α_r^i, β_r^i, C_r^i) for the case of remote nodes. Here, we adopt the same parameters for all remote nodes. It is because that all remote nodes use the same interconnection (i.e., QPI) to access the memory on local node.

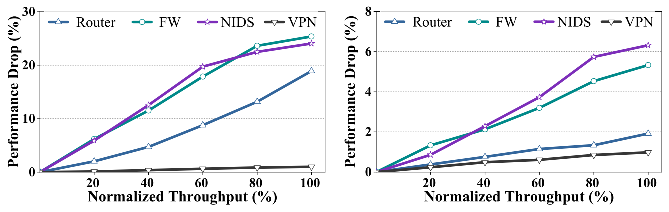
Let $\{m^l\}$ and $\{m^r\}$ denote the measured *resource_stalls* of those NFs locate at a local node (NF_l) and a remote node (NF_r). Consider an NF in a local node shown in Fig. 4a, it will compete for LLC with the NFs in the local node, and compete for memory controller with all NFs inside the same server. For an NF in a remote node, we set its LLC misses to "1", i.e., no cache hit for packet access. This is because it cannot access the cache on the local node. Moreover, since the bandwidth of QPI is lower than the main memory controller, we consider QPI as the competing resource and the NFs in the remote node as the competitors. Therefore, we construct the model as

$$\lambda^i = \begin{cases} \alpha_l^i \cdot f(\{m^l\}) + \beta_l^i \cdot f(\{m^l\} \cup \{m^r\}) + C_l^i, \ \forall i \in NF_l \\ \alpha_r^i \cdot 1 + \beta_r^i \cdot f(\{m^r\}) + C_r^i, \ \forall i \in NF_r \end{cases}. \tag{10}$$

4.2.3 Fine-Tuning Prediction Model for SFC Characteristics

As mentioned in Section 3.2, SFC will complicate the prediction model since different chaining methods (e.g., cross-node) could change the shared resources and competitors of an NF. Besides, the competing level introduced by an NF could change since it is related to the current throughput. However, the throughput of NFs in an SFC are potentially imbalanced [40] and the chaining methods could limit the throughput they can achieve. To address these problems, we first tune the competing level introduced by an NF due to imbalanced throughput, then we tune the prediction model to reflect the shared resource and competitors changes.

(1): Tuned model for throughput limitation. When multiple NFs are chained to form an SFC, the throughput of subsequent NFs cannot exceed their previous NFs. That is to say some NFs could process far less traffic than their maximum capacity. For example in Fig. 9, there are three NFs (NF1, NF2 and NF3) and their capacity are 1P, 0.3P and 0.7P, respectively. When they are chained sequentially, the throughput of NF3 can only achieve 0.3P rather than 0.7P. Furthermore, the competing level introduced by an NF varies as the throughput changes. In order to explore the



(a) Performance drop with increasing throughput on local node. (b) Performance drop with increasing throughput on remote node.

Fig. 10. Performance drop with increasing throughput.

relationship between them, we co-locate different NFs with a competing NF (i.e., Router) in both nodes and change its throughput. As shown in Fig. 10, in both local and remote node, when the throughput of competing NF increases, the performance of co-locating NF drops more, i.e., higher competing level. However, the value we used in competing level function for each NF represents the competing level introduced with the maximum throughput.

To tune the prediction model for throughput limitation imposed by SFC, we add an coefficient η to the competing level introduced by an NF. This coefficient is the ratio of actual throughput to its maximum throughput on that node. However, we cannot get the actual throughput of an NF before deployment. Thus, we use the minimal throughput of its previous NFs (including itself) in an SFC as its actual throughput, which could represent an upper bound of competing level. Take Fig. 9 as an example, coefficients of NF1 and NF2 are "1" while NF3's coefficient is set to "3/7". Thus, we refine the competing level function as

$$f(\{x_1, x_2, \ldots, x_n\}) \approx \sum_{1 \le k \le n} \eta \cdot \frac{1}{x_k}, \tag{11}$$

where

$$\eta = \frac{\min_{1 \le m \le \varphi_2(k)} \{P_{\varphi_1(k)m}\}}{P_{\varphi_1(k)\varphi_2(k)}}, \tag{12}$$

x_k is used to represent the metric (*resource_stalls*) of kth NF, $\varphi_1(k)$ and $\varphi_2(k)$ are functions to map the kth NF to the $\varphi_2(k)$th NF in SFC $\varphi_1(k)$. $P_{\varphi_1(k)\varphi_2(k)}$ represents the ideal performance of $\varphi_2(k)$th NF in SFC $\varphi_1(k)$. Note that η is a constant for a special SFC once the placement is determined.

(2): Tuned model for chaining methods. As introduced in Section 3.2, two chaining methods should be considered. The first one is *cross-node chaining*. As shown in Fig. 5a, for an NF (i.e., NF2) in the local node, its upstream NF is in the remote node. This makes the NF has no chance to read packets from its local LLC. Thus, we should set its cache misses to "1", i.e.,

$$\lambda^i = \alpha_l^i \cdot 1 + \beta_l^i \cdot f(\{m^l\} \cup \{m^r\}) + C_l^i. \tag{13}$$

The second method is *cache-prefetching chaining*. As shown in Fig. 5b, for an NF (i.e., NF6) in the remote node, we do not regard it as those NFs (e.g., NF4) that have no chance to read packets from LLC. This is because that its upstream NF (i.e., NF5) could load packets into remote LLC as mimicking a cache prefetcher [41]. As a result, NF4 has chance to access packet from remote LLC and no longer to

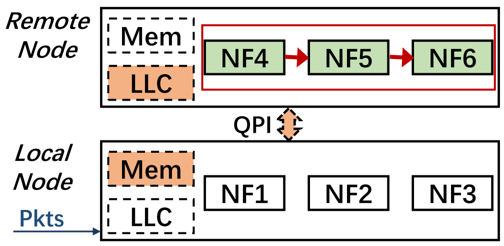


Fig. 11. LLC sharing for a subchain.

be “1” for LLC misses. Moreover, We consider the NFs in the remote node as its LLC competitor. Thus, we tune the model for *cache-prefetching chaining* as

$$\lambda^i = \alpha_r^i \cdot f(\{m^r\}) + \beta_r^i \cdot f(\{m^r\}) + C_r^i. \tag{14}$$

In summary, whether an NF in an SFC can benefit from LLC is determined by the relative location to its upstream NF. If its upstream NF is in the same node, it can be benefit from LLC due to cache-prefetching from its upstream NF, and consider NFs in the same node as its LLC competitors. Otherwise, it suffers from cache misses no matter it locates at local or remote node. Thus, we provide a general prediction model for different chaining methods as

$$\lambda^i = \begin{cases} \alpha_l^i \cdot \delta_l^i + \beta_l^i \cdot f(\{m^l\} \cup \{m^r\}) + C_l^i, \ \forall i \in NF_l \\ \alpha_r^i \cdot \delta_r^i + \beta_r^i \cdot f(\{m^r\}) + C_r^i, \ \forall i \in NF_r \end{cases}, \tag{15}$$

where

$$\delta_l^i = f(\{m^l\}) \cdot y_i + (1 - y_i) \tag{16}$$

$$\delta_r^i = f(\{m^r\}) \cdot y_i + (1 - y_i), \tag{17}$$

y_i is a binary variable to indicate whether its upstream NF located at the same node. δ describes cache contention of an NF, which could be $f(\{m\})$ or ”1”.

Moreover, since NFs in a cache-prefetching chain share the same piece of LLC, we regard these NFs as only one competitor to other NFs for LLC sharing. For example in Fig. 11, since the subchain consisting of NF4, NF5 and NF6 accesses the same packet memory sequentially, they only occupy one piece of cache. Currently, we use the first NF (i.e., NF4) in this subchain to calculate competing level.

4.2.4 *Extending Prediction Model to Multiple NICs*

To support higher aggregate throughput, many servers are equipped with multiple NICs which could locate at different nodes. However, as mentioned in Section 3.2, multiple NICs will complicate the prediction model since the way of traffic entering a server could change the shared resources and competitors of an NF. Furthermore, the shared resources of an NF are related to its local node, which could change for different NFs and can only be known at runtime. To address this problem, we treat the NIC which receives traffic for NFs as a special NF chained before these NFs, except that this special NF can only be located at nodes that are equipped with NICs. Although this special NF does not introduce competition for LLC (or only little) and memory, it changes the shared resource of NFs in a server.

Algorithm 1. Heuristic Placement Algorithm

```
 Input : $(S)$ - SFC sets
 Output : $X$ - Placement of each NF on the SFCs
 1: // 1: Initial placement
 2: $X_{init} \leftarrow$ Binary-search all possible placement for $s \in S$ with Constraint 3, 4 and 5.
 3: // 2: Move some subchains in the remote nodes to local node
 4: $Perf_{opt} \leftarrow 0$ // The optimal performance
 5: foreach $x \in X_{init}$ do
 6:     $(S_{local}, S_{remote}) \leftarrow$ The SFCs placed in the local and remote node from $x$
 7:     Update_Placement $(S_{local}, S_{remote})$
 8:     DFS$(S_{local}, S_{remote})$
 9: function DFS $(S_{local}, S_{remote})$
10:     foreach $s \in S_{remote}$ do
11:         $NF_{bottleneck} \leftarrow \min\{NF \in s\}$ // The NF with lowest performance
12:         $s_{subchain} \leftarrow s[0 : NF_{bottleneck}]$ // The subchain
13:         if Enough cores for $s_{subchain}$ in the local node then
14:             $(tmp\_S_{local}, tmp\_S_{remote}) = (S_{local} + S_{subchain}, S_{remote} - S_{subchain})$
15:             Update_Placement$(tmp\_S_{local}, tmp\_S_{remote})$
16:             DFS$(tmp\_S_{local}, tmp\_S_{remote})$
17: function Update_Placement $(S_{local}, S_{remote})$
18:     $Perf_{prediction} \leftarrow$ Performance prediction from Eqns. (18), (19), (20) and (21)
19:     if $Perf_{opt} < Perf_{prediction}$ then
20:         $Perf_{opt} = Perf_{prediction}$
21:         $X = \{S_{local}, S_{remote}\}$ // updating placment
```

As shown in Fig. 6, the NIC (highlighted with yellow) which receives traffic for an NF (i.e., NF2) acts as a packet generator and sends traffic to its subsequent NF. Since the NIC can send traffic to LLC (in the same node) directly (with Intel DDIO [42]), it will share LLC with those NFs in the same node. Therefore, if its subsequent NFs locate at the same node, those NFs would benefit from LLC hits (Fig. 6b), or they could suffer from LLC misses (Fig. 6a). Fortunately, we can reuse the tuned models (i.e., Eqn. (15)) since NICs only changes the LLC sharing of its subsequent NFs, and we can regard the NIC as a special NF to mimic LLC impact. However, memory contention of NFs in an SFC could be changed by the location of this special NF. In each node, if the subsequent NFs of an SFC locate at the same node with this special NF (i.e., the node with a NIC), they compete for local memory, which we refer to as local NF (i.e., NF5 in Fig. 6b). Otherwise, they compete for QPI and remote memory, which we refer to as remote NF (i.e., NF2 in Fig. 6a). What needs to be noted here is that we have changed the definition of local and remote NFs, which is only determined by their relative location to the special NF. Therefore, we extend the prediction model for multiple NICs in each node as

$$\lambda^i = \begin{cases} \alpha_l^i \cdot \delta_l^i + \beta_l^i \cdot \theta_l^i + C_l^i, \ \forall i \in NF_l \\ \alpha_r^i \cdot \delta_r^i + \beta_r^i \cdot \theta_r^i + C_r^i, \ \forall i \in NF_r \end{cases}, \tag{18}$$

where

$$\delta_l^i = \delta_r^i = f(\{m^l\}) \cdot y_i + (1 - y_i) \tag{19}$$

$$\theta_l^i = f(\{m_l^l\} \cup \{m_l^r\}) \tag{20}$$

$$\theta_r^i = f(\{m_r^l\}), \tag{21}$$

δ_l^i and δ_r^i indicate LLC contention for local and remote NFs. Since all NFs on this node compete for LLC ($f(\{m^l\})$) and whether an NF can benefit from LLC is only determined by the relative location of its upstream NF (y_i), the LLC contention for local and remote NFs imposes no difference. θ_l^i indicates memory contention for local NFs. These NFs compete for memory with NFs whose memory (i.e., packet buffer) locate at this node, no matter the NF itself locates on this node ($f(\{m_l^l\})$) or other nodes ($f(\{m_l^r\})$). θ_r^i indicates memory contention for remote NFs. These NFs only compete for QPI with other remote NFs on this node ($f(\{m_r^l\})$).

One can see that if measuring samples are given, including the placement of SFCs and performance drop index of each NF, it is easy to approximate the parameters (α, β, C) for each NF. In Octans, we use a fast learning algorithm, Least Mean Squares (LMS), to approximate them.

4.3 Online Placement Algorithms

So far, we have constructed the prediction model for the performance drop index in Eqn. (2), which makes our formulation solvable. A naive approach to find optimal solution is brute-force search, but it can be expensive. Since we can see that its time complexity is $O(K^N)$ (K is the total number of available nodes and N is the total number of NFs in all SFCs). However, the network operator usually needs fast response for placement requests [8], which implies a short time for solution searching. Furthermore, our problem is a Multidimensional Assignment Problem (MAP) (an NF can be assigned to any one of multiple nodes), which is a known NP-complete problem [30]. There is no polynomial-time algorithm to find an optimal solution. Besides, in some cases, we need incremental deploy SFCs to servers since not all placement requests come at the same time and we could utilize severs which is underloaded to deploy more SFCs. Therefore, we first propose a heuristic algorithm to find the optimal or near-optimal solution to deploy multiple placement requests simultaneously, then we also propose a greedy algorithm to support incremental deployment.

Our heuristic algorithm follows two major steps: (1) *initial placement: chain-based search.* Instead of searching placement for all NFs, we first try to place an entire chain on the same node. It is based on an important observation that a cache-prefetching chain always has lower performance drop against a cross-node chain from Eqns. (13) and (14). This is because that the output of $f(\{m^l\})$ and $f(\{m^r\})$ is always smaller than 1; and (2) *moving subchains in a remote node to the local node.* A common sense is that an NF in the local node usually has higher performance than in a remote node due to benefiting of lower memory access latency (i.e., iMC is faster than QPI). Hence, to improve the chance to find an optimal solution based on the initial placement, we try to move some subchains in the remote node to the local node. Besides, since the throughput of an SFC is determined by its bottleneck-NF (i.e., with the lowest performance), the split point we choose in an entire chain to form a subchain is the bottleneck-NF to improve its performance.

We show the heuristic algorithm in Algorithm 1. It first produces the initial placement via binary-search for all possible placements with entire SFCs across all nodes (line 2). Then, it iterates each initial placement to find potential placement with higher aggregate performance by moving some subchains from a remote node to the local node (lines 5-8). We adopt deep-first search (DFS) to find the bottleneck-NF for each SFC in remote nodes (lines 9-16), meanwhile update the optimal solution if the moving operation produces higher performance (lines 17-21).

Although this heuristic algorithm could find an optimal or near-optimal placement solution for multiple SFCs in a short time, it cannot be directly applied to incremental deployment. This is because it trades performance for time complexity and could miss optimal solution. However, we only need to deploy one SFC at a time in incremental deployment and the length of SFC is usually less than seven NFs [43]. Therefore, the optimal placement solution of an SFC is likely to be found quickly even with brute-force search since we adopt static placement mechanism and do not change deployed SFCs. For this reason, we propose a greedy algorithm and apply optimal placement for each SFC in incremental deployment.

Algorithm 2. Greedy Placement Algorithm

```
1: Input: $(NF_1, NF_2, \ldots, NF_n)$ - SFC and deploying server.
2: Get the status of all deployed SFC sets on the deploying
   server, including their placement and used NICs.
3: Binary-search all possible placement for $(NF_1, NF_2, \ldots, NF_n)$
   and predict the aggregate throughput of this server based on
   the prediction model with Constraint 3, 4 and 5.
4: Output: placement for $(NF_1, NF_2, \ldots, NF_n)$ with highest
   aggregate throughput.
```

We show the greedy algorithm in Algorithm 2. When it receives a SFC (line 1), it first gets all deployed SFC sets and their status on the server (line 2). Then, it searches all possible placement solutions and predicts throughput based on our models (line 3). Finally, it chooses the solution with highest aggregate throughput as output (line 4).

5 Octans Architecture and Workflow

According to the design in Section 4, we present the architecture and workflow of Octans as shown in Fig. 12. Octans takes SFC placement requests on a specific server (i.e., S) as the input, and checks whether or not existing new NFs in inputing SFCs. If existed, they are sent to a *sandbox environment* to calculate the metric value (i.e., *resource_stalls*) and generate performance model, which is supported by two major modules, *Metric Profiler* and *Model Generator*; If not, requests are immediately sent to *Placement Engine*, to calculate placement solution. Next, we show the detail of these three modules.

Metric Profiler. Metric Profiler automatically profiles *resource_stalls* for every NF. In current implementation, it uses OProfile [36] to measure the count of *resource_stalls*.

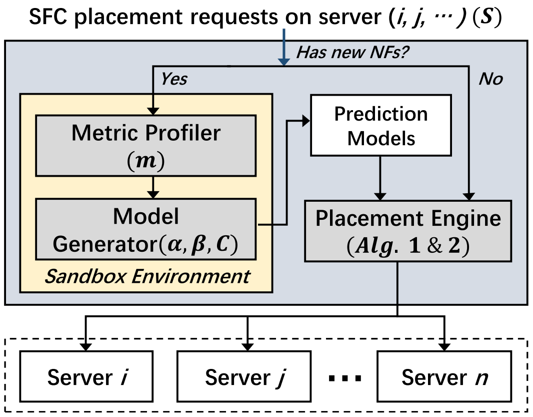


Fig. 12. Architecture and workflow of `Octans`.

Since `Octans` focuses on the performance in throughput, we generate 10 Gbps traffic (this rate can be set) with minimum size packets (i.e., 64B). Metric Profiler records the total number of packets it processed (excluding dropped packets) and the total count of *resource_stalls* after a period of running, then calculates *resource_stalls* per packet as the metric value.

Model Generator. Model Generator generates the three parameters (α, β, C) for every NF. First, it runs each NF solely to record the performance as ideal performance (i.e., P_{ij}^k in Eqn. (2)). Second, it co-locates this NF with different types and numbers of NFs (including itself) to measure the interfered performance ϕ_{ij}^k, meanwhile records the performance drop index with calculation $\frac{P_{ij}^k - \phi_{ij}^k}{P_{ij}^k}$ as sampled data. After sampling enough data (this might take a long time, but it can be completed offline), this generator adopts LMS algorithm to approximate the parameters. Finally, it updates the record of *Prediction Models*, where stores the prediction model for every NF.

Placement Engine. Placement Engine runs the online placement algorithms (Section 4.3). It takes the placement requests (i.e., S) as input, and retrieves the model of each NF from *Prediction Models*. After an optimal or near-optimal solution is found, it launches the required NFs in SFC requests on assigned CPU cores of target servers.

6 Implementation and Evaluation

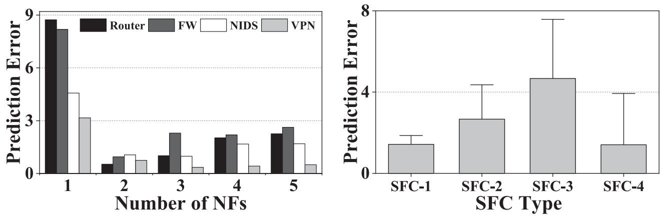
6.1 Implementation and Experiment Setup

Implementation. We have implemented a prototype of `Octans`. The infrastructure we used to run NFs and SFCs is built on a high-performance NFV platform, OpenNetVM [3]. The NFs we used include *IPv4Router* (Router), *Firewall* (FW), *NIDS*, and *VPN*. The three modules in `Octans` are written in Python language.

Experiment Setup. We run `Octans` and OpenNetVM on the same server, which is a two-node server that is equipped with two Intel Xeon E5-2650 v4 CPUs (2.20 GHz, 12 physical cores), 128 GB total memory, and two dual-port 10G NICs (Intel X520-DA2, 40 Gbps in total). We use DPDK Pktgen [44] to generate traffic, which runs on another server that has the same configuration as the previous one. Both servers run Ubuntu 14.04 (with kernel 3.16.0-30), and DPDK version 18.02. The two NICs are located on Node-0, hence we treat Node-0 as the local node and Node-1 as the remote node. We allocate 1 core to `Octans`, 1 core to the openNetVM manager, and 4 cores for networking I/O and packet forwarding. Therefore, we have 6 available cores in the local node and 12 available cores in the remote node. Except for special statement, we use the same setup for all evaluations.

We evaluate `Octans` with the following goals.

- Demonstrate that `Octans` can achieve accurate performance drop prediction for a wide range of NFs and SFCs even with multiple NICs on different nodes.
- Demonstrate that `Octans` can improve aggregate performance by searching optimal or near-optimal placement solution for one-time and incremental deployment, compared with two alternative placement solutions.
- Demonstrate that `Octans` can search the placement solution within a short time, and has reasonable chance to find the optimal solution.



(a) Prediction error for different NFs. (b) Prediction error for different SFCs.

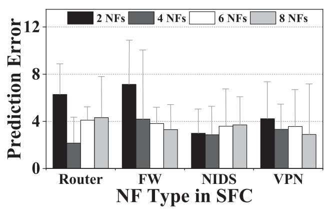
Fig. 13. Prediction accuracy for separate NFs and SFCs.

6.2 Prediction Accuracy

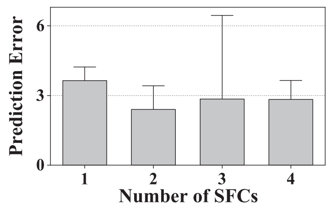
To demonstrate the prediction accuracy of performance drop for NFs and SFCs, we compare the performance drop between our prediction value ($\lambda^{predicted}$) and the measured value ($\lambda^{measured}$) by deploying them into our testbed. We calculate the prediction error as $\frac{|\lambda^{measured} - \lambda^{predicted}|}{\lambda^{measured}}$.

Prediction Accuracy for Different NFs. We first evaluate the prediction accuracy for separated NFs when co-locating with competitors. We use a synthetic NF (ideal performance between NIDS and Firewall) as the co-locating NF (competitor) for our evaluated NFs. Fig. 13a shows the prediction error of the four NFs when they co-locate with different number of competitors. From this figure, we can see reasonablely accurate results, an average of 2.9, 3.3, 2.0 and 1.0 percent deviation from the measured performance drop for Router, Firewall, NIDS and VPN, respectively. Also, we can observe that with the number of competitors increases, the prediction results become more accurate. For example, the prediction error reduces from 8.7 to 2.3 percent for the Router. This is because the gradient of performance drop becomes smaller with the competing level increases, and our prediction model can capture this feature.

Prediction Accuracy for Different SFCs. We then evaluate the prediction accuracy for four customised SFCs and mark them as: SFC-1 (Router, Firewall, NIDS, VPN), SFC-2 (Firewall, Router, NIDS, Firewall), SFC-3 (NIDS, Router, Firewall, NIDS), and SFC-4 (Router, NIDS, Router, Firewall).

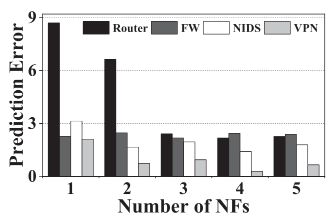


(a) Prediction error for different length of SFCs.

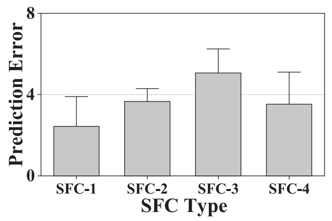


(b) Prediction error for different number of SFCs.

Fig. 14. Prediction accuracy for different length and number of SFCs.



(a) Prediction error for different NFs with multiple NICs.



(b) Prediction error for different SFCs with multiple NICs.

Fig. 15. Prediction accuracy for separate NFs and SFCs with multiple NICs.

We randomly generate 10 types of placement for every SFC. Fig. 13b shows the average prediction error of these SFCs. We can see that all of them have reasonably accurate results, and the worst-case of average error (SFC-3) is less than 4.7 percent.

Effect of the Varied Length of SFC. To show more reliable prediction results, we evaluate whether the length of SFC has the potential to incur higher prediction error. We use four SFCs, and NFs in each SFC are the same. For different lengths of SFC, we randomly generate up to 20 types of placement. Fig. 14a shows the prediction error for the four SFCs with different lengths. We can still see reasonably accurate results, which range from 2.2 to 6.3 percent, 3.3 to 7.1 percent, 2.9 to 3.7 percent, and 2.9 to 4.2 percent for these SFCs, respectively.

Effect of the Varied Number of Co-Locating SFCs. We also evaluate whether the number of co-locating SFCs has the potential to incur higher prediction error. We use an SFC with three NFs (Router, Firewall, NIDS), and increase the number of co-locating SFCs from 1 to 4. We randomly generate up to 20 types of placement for each testing scenario. Fig. 14b shows that the prediction error varies from 2.4 to 3.6 percent, which is also a reasonably accurate result.

Prediction Accuracy With Multiple NICs on Different Nodes. We evaluate whether the prediction accuracy falls with multiple NICs for separate NFs and SFCs. In this evaluation, we move one NIC to Node-1 and allocate two CPU cores on Node-1 for networking I/O and packet forwarding. For simplify and intuitive comparison, we use the same workloads and factors in Fig. 13 except that these NFs and SFCs randomly receive traffic from different NICs. Fig. 15a shows the prediction error of the four separate NFs when co-locating with different number of competitors. From this figure, we can see basically similar prediction error with Fig. 13a, with an average of 4.4, 2.3, 1.9 and 0.9 percent deviation from the measured performance drop for Router, Firewall, NIDS and VPN. Fig. 15b shows the prediction error of the four customised SFCs with randomly 10 types of placement. From this figure, we can also see basically similar prediction error with Fig. 13b and the worst-case of average prediction error (SFC-3) is less than 5.1 percent. Therefore, above evaluations could demonstrate that our extending to multiple NICs is accurate and reliable.

Prediction Accuracy Without Tuning for SFC and Multiple NICs. To show the improvement of prediction accuracy with tuning for SFC and multiple NICs, we reevaluate the experiments in Figs. 13b and 15a but only with primary prediction model. Compared with the SFC tuning in Fig. 13b, the primary prediction model can achieve an average of 5.9, 14.8, 15.6 and 20.8 percent deviation from the measured performance drop for SFC-1, SFC-2, SFC-3 and SFC-4. Compared with the multiple NICs tuning in Fig. 15a, the average deviation of primary predication model can be 14.8, 22.6, 21.4 and 6.3 percent for Router, Firewall, NIDS and VPN. Both prediction results are much worse than our final prediction model and therefore prove the importance of fine tuning for SFC and multiple NICs.

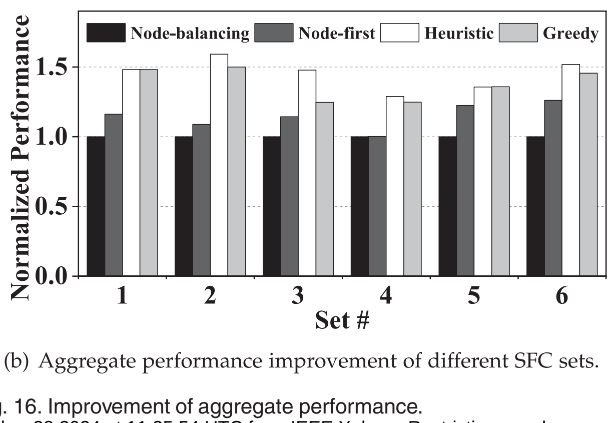
6.3 Improvement of Aggregate Performance

To demonstrate that `Octans` can improve the aggregate performance by finding optimal or near-optimal SFCs placement with our online algorithms (Heuristic and Greedy), we compare them with two alternative placement mechanisms that could be used in current systems: (1) *node-balancing placement:* it places NFs by evenly dividing them to all nodes for balancing the core utilization on each node; and (2) *node-first placement:* it tries to place NFs on the local node first until no core is available, then it places NFs on other nodes. We use 6 customised sets of SFCs and each with different SFC requests (Fig. 16a).

Fig. 16b shows the normalized performance with placement algorithms in `Octans` and the two alternative mechanisms. We can see that in all SFC sets, the heuristic placement algorithm achieves highest aggregate performance and compared with Node-balancing placement, it

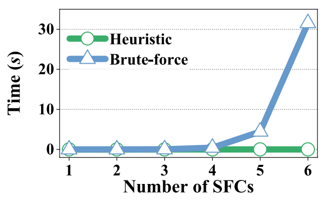
Set #	SFCs
1	(Router, Firewall, NIDS); (Router, NIDS); (VPN, Firewall)
2	(Router, Firewall, VPN); (Router, Firewall); (Firewall, NIDS)
3	(Router, Firewall, NIDS); (Router, Firewall, VPN); (NIDS, Firewall)
4	(Router, Firewall, NIDS); (Firewall, NIDS, VPN); (Firewall, Router)
5	(Router, Firewall, NIDS); (Router, Firewall, VPN); (Firewall, NIDS, VPN)
6	(Router, Firewall, NIDS, VPN); (Router, Firewall, NIDS); (NIDS, Router)

(a) The SFC sets used in evaluation.

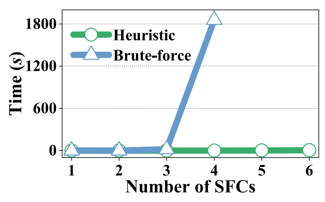


(b) Aggregate performance improvement of different SFC sets.

Fig. 16. Improvement of aggregate performance.

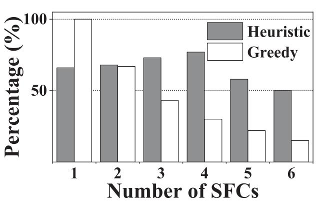


(a) Time cost with different number of SFCs in a 2-node server.

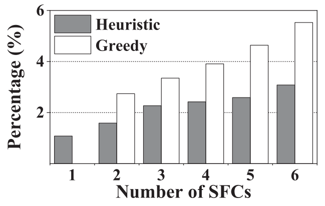


(b) Time cost with different number of SFCs in a 4-node server.

Fig. 17. The time cost of our heuristic algorithm.



(a) Chance to find an optimal solution.



(b) Deviation from the optimal solution.

Fig. 18. Optimality gap of the online algorithms.

can achieve higher aggregate performance by an average of 45.2 percent and ranging from 28.9 to 59.1 percent. Also, compared with node-first placement, it can achieve higher aggregate performance by an average of 27.1 percent and ranging from 10.8 to 46.1 percent. Besides, to support incremental deployment, the aggregate performance of greedy placement algorithm could be lower than the heuristic algorithm by an average of 5.4 percent, however it can still be higher than the two alternative mechanisms by an average of 38.1 and 20.9 percent

6.4 Efficiency of Online Placement Algorithms

Time Cost of Solution Search. Since the greedy algorithm in Octans receives placement requests one by one and the NFs of each request is usually less than seven [43], it could find the optimal solution for a request in a short time. Therefore, we only compare the heuristic algorithm in Octans to the naive *brute-force search* algorithm to evaluate its time cost. We use an SFC with 3 NFs and varied number (1∼6) of SFCs as the input for each algorithm. In each set of SFCs, we randomly replace NFs and repeat it up to 100 times. A many-core system usually has 2 or 4 nodes [45]. Therefore, we also evaluate the effect of the number of nodes (setting 10 cores in each node) on calculation time. We only allocate one CPU core (2.2 GHz) to run the algorithm.

Fig. 17a shows that in a 2-node server, even in the worst-case (18 NFs in 6 SFCs), our heuristic algorithm can find a solution with an average time of 0.017s, which takes 1853x less time than the *brute-force search algorithm* (31.5s). Also, in a 4-node server as shown in Fig. 17b, our heuristic algorithm can find a solution with an average time of 5.3s in the worst-case, but the *brute-force search* needs to spend more than 1858s (we use the case with 4 SFCs as this value due to the long calculation time of 5 and 6 SFCs).

The Chance to Find Optimal Solutions. Our online algorithms are based on heuristic and greedy, hence they cannot guarantee an optimal solution. We evaluate the chance that our algorithms can find the optimal solution, and if a near-optimal solution could be found, what the deviation it is. The optimal solution is found by brute-force search algorithm (but taking long time), and we run the three algorithms with a simulated 2-node server. The SFCs and NFs generation are similar to the previous evaluation, and we also repeat each set up to 100 times. Fig. 18a shows the probability Octans could have to find an optimal solution across different numbers of SFCs. For one-time deployment, the heuristic algorithm has a 50∼77 percent chance to find an optimal solution. For incremental deployment, the chance to find an optimal solution decreases to 15 percent. However, as shown in Fig. 18b, even with a near-optimal solution, it has little deviation from the optimal one. With the number of SFCs grows, the deviation of heuristic algorithm increases from 1.08 to 3.08 percent while the deviation of greedy algorithm increases reasonably up to 5.53 percent.

7 Conclusion

We have presented Octans, an NFV orchestrator that searches optimal or near-optimal placement for SFCs in a many-core NFV system. Starting with an NLIP model, Octans first constructs an accurate model to predict an unknown parameter with automatically identified metric of NFs. Then, it provides two online placement algorithms to quickly find a solution for placement requests. Our evaluation built upon OpenNetVM shows that Octans provides accurate prediction results, significantly improves the aggregate performance of SFCs, and has a high chance to find optimal solutions in a short time.

Acknowledgments

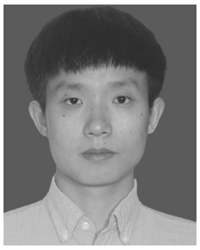
The authors would like to thank anonymous TPDS reviewers for their valuable suggestions. This work was supported in part by National Key R&D Program of China under Grant 2020YFE0200500 (Joint Research on IPv6 Network Governance: Research, Development and Demonstration).

References

[1] S. Palkar *et al.*, “E2: A framework for NFV applications,” in *Proc. 25th Symp. Operating Syst. Princ.*, 2015, pp. 121–136.
[2] C. Sun *et al.*, “NFP: Enabling network function parallelism in NFV,” in *Proc. Conf. ACM Special Interest Group Data Commun.*, 2017, pp. 43–56.
[3] W. Zhang *et al.*, “OpenNetVM: A platform for high performance network service chains,” in *Proc. Workshop Hot Topics Middleboxes Netw. Function Virtualization*, 2016, pp. 26–31.
[4] X. Fei *et al.*, “Paving the way for NFV acceleration: A taxonomy, survey and future directions,” *ACM Comput. Surv.*, vol. 53, 2020, Art. no. 73.
[5] OpenStack, “Openstack,” 2018. [Online]. Available: https://www.openstack.org/
[6] S. Blagodurov, A. Fedorova, S. Zhuravlev, and A. Kamali, “A case for NUMA-aware contention management on multicore systems,” in *Proc. 19th Int. Conf. Parallel Archit. Compilation Techn.*, 2010, pp. 557–558.
[7] M. Dobrescu *et al.*, “Toward predictable performance in software packet-processing platforms,” in *Proc. 9th USENIX Symp. Netw. Syst. Des. Implementation*, 2012, Art. no. 11.
[8] A. Tootoonchian *et al.*, “ResQ: Enabling SLOs in network function virtualization,” in *Proc. 15th USENIX Symp. Netw. Syst. Des. Implementation*, 2018, pp. 283–297.
[9] C. Zeng, F. Liu, S. Chen, W. Jiang, and M. Li, “Demystifying the performance interference of co-located virtual network functions,” in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 765–773.
[10] H. Moens and F. D. Turck, “VNF-P: A model for efficient placement of virtualized network functions,” in *Proc. 10th Int. Conf. Netw. Service Manage. Workshop*, 2014, pp. 418–423.

[11] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *Proc. IEEE Conf. Comput. Commun.*, 2015, pp. 1346–1354.
[12] T.-W. Kuo, B.-H. Liou, K. C.-J. Lin, and M.-J. Tsai, "Deploying chains of virtual network functions: On the relation between link and server usage," *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1562–1576, Aug. 2018.
[13] S. Zhuravlev *et al.*, "Addressing shared resource contention in multicore processors via scheduling," *ACM SIGARCH Comput. Archit. News*, vol. 38, pp. 129–142, 2010.
[14] B. Lepers *et al.*, "Thread and memory placement on NUMA systems: Asymmetry matters," in *Proc. USENIX Annu. Tech. Conf.*, 2015, pp. 277–289.
[15] J. Rao, K. Wang, X. Zhou, and C.-Z. Xu, "Optimizing virtual machine scheduling in NUMA multicore systems," in *Proc. IEEE 19th Int. Symp. High Perform. Comput. Archit.*, 2013, pp. 306–317.
[16] M. Liu and T. Li, "Optimizing virtual machine consolidation performance on NUMA server architecture for cloud workloads," in *Proc. 41st Int. Symp. Comput. Archit.*, 2014, pp. 325–336.
[17] A. Gember-Jacobson *et al.*, "OpenNF: Enabling innovation in network function control," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 163–174, 2014.
[18] Y. Xiao *et al.*, "NFVdeep: Adaptive online service function chain deployment with deep reinforcement learning," in *Proc. Int. Symp. Qual. Service*, 2019, pp. 1–10.
[19] X. Fei, F. Liu, H. Xu, and H. Jin, "Adaptive VNF scaling and flow routing with proactive demand prediction," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 486–494.
[20] Q. Zhang, Y. Xiao, F. Liu, J. C. S. Lui, J. Guo, and T. Wang, "Joint optimization of chain placement and request scheduling for network function virtualization," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 731–741.
[21] Q. Zhang, F. Liu, and C. Zeng, "Adaptive interference-aware VNF placement for service-customized 5G network slices," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 2449–2457.
[22] P. Jin, X. Fei, Q. Zhang, F. Liu, and B. Li, "Latency-aware VNF chain deployment with efficient resource reuse at network edge," in *Proc. IEEE Conf. Comput. Commun.*, 2020, pp. 267–276.
[23] Y. Li, L. T. Xuan Phan, and B. T. Loo, "Network functions virtualization with soft real-time guarantees," in *Proc. 35th Annu. IEEE Int. Conf. Comput. Commun.*, 2016, pp. 1–9.
[24] J. Funston *et al.*, "Placement of virtual containers on NUMA systems: A practical and comprehensive model," in *Proc. USENIX Annu. Tech. Conf.*, 2018, p. 281–293.
[25] S. Kashyap *et al.*, "Scalable NUMA-aware blocking synchronization primitives," in *Proc. USENIX Annu. Tech. Conf.*, 2017, pp. 603–615.
[26] T. Wang, H. Xu, and F. Liu, "Multi-resource load balancing for virtual network functions," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 1322–1332.
[27] X. Fei, F. Liu, H. Jin, and B. Li, "FlexNFV: Flexible network service chaining with dynamic scaling," *IEEE Netw.*, vol. 34, no. 4, pp. 203–209, Jul./Aug. 2020.
[28] S. Rajagopalan *et al.*, "Split/merge: System support for elastic execution in virtual middleboxes," in *Proc. 10th USENIX Symp. Netw. Syst. Des. Implementation*, 2013, pp. 227–240.
[29] J.-P. Lozi *et al.*, "The Linux scheduler: A decade of wasted cores," in *Proc. 11th Eur. Conf. Comput. Syst.*, 2016, Art. no. 1.
[30] Y. Jiang, X. Shen, C. Jie, and R. Tripathi, "Analysis and approximation of optimal co-scheduling on chip multiprocessors," in *Proc. Int. Conf. Parallel Archit. Compilation Techn.*, 2008, pp. 220–229.
[31] Y. Wang, "NUMA-aware design and mapping for pipeline network functions," in *Proc. 4th Int. Conf. Syst. Informat.*, 2017, pp. 1049–1054.
[32] C. Sieber *et al.*, "Towards optimal adaptation of NFV packet processing to modern CPU memory architectures," in *Proc. 2nd Workshop Cloud-Assisted Netw.*, 2017, pp. 7–12.
[33] Y. Hu and T. Li, "Towards efficient server architecture for virtualized network function deployment: Implications and implementations," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2016, pp. 1–12.
[34] M. Li, Q. Zhang, and F. Liu, "Finedge: A dynamic cost-efficient edge resource management platform for nfv network," in *Proc. IEEE/ACM 28th Int. Symp. Qual. Service*, 2020, pp. 1–10.
[35] Y. Zhang, W. Wu, S. Banerjee, J.-M. Kang, and M. A. Sanchez, "SLA-verifier: Stateful and quantitative verification for service chaining," in *Proc. IEEE Conf. Comput. Commun.*, 2017, pp. 1–9.
[36] OProfile, "Oprofile," 2018. [Online]. Available: http://oprofile.sourceforge.net/news/
[37] Intel, "Intel performance counter monitor," 2017. [Online]. Available: https://software.intel.com/en-us/articles/intel-performance-counter-monitor/
[38] R. Nathuji *et al.*, "Q-clouds: Managing performance interference effects for QoS-aware clouds," in *Proc. 5th Eur. Conf. Comput. Syst.*, 2010, pp. 237–250.
[39] J. Benesty *et al.*, "Pearson correlation coefficient," in *Noise Reduction in Speech Processing*. Berlin, Germany: Springer, 2009.
[40] W. Ma, O. Sandoval, J. Beltran, D. Pan, and N. Pissinou, "Traffic aware placement of interdependent NFV middleboxes," in *Proc. IEEE Conf. Comput. Commun.*, 2017, pp. 1–9.
[41] J. Collins *et al.*, "Pointer cache assisted prefetching," in *Proc. 35th Annu. ACM/IEEE Int. Symp. Microarchit.*, 2002, pp. 62–73.
[42] Intel, "Data direct I/O (DDIO)," 2014. [Online]. Available: https://www.intel.com/conten/www/us/en/io/data-direct-i-o-technology.html
[43] S. Kumar *et al.*, "Service function chaining use cases in data centers," *IETF SFC WG*, 2015.
[44] I. D. Community, "DPDK pktgen," 2018. [Online]. Available: http://pktgen-dpdk.readthedocs.io/en/latest/
[45] Dell, "Dell poweredge servers portfolio guide," 2018. [Online]. Available: http://www.dell.com/downloads/global/products/pedge/en/pedge-portfolio-brochure.pdf

Heng Yu received the BS degree from the School of Communication and Information Engineering, University of Electronic Science and Technology, Chengdu, China, in 2017. He is currently working toward the PhD degree at the Institute of Network Science and Cyberspace, Tsinghua University, Beijing, China. His research interests include cloud computing, serverless computing, and network function virtualization.

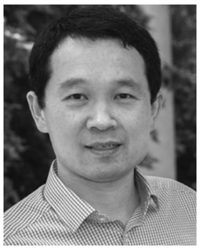


Zhilong Zheng received the BS degree from the Department of Software Engineering, Chongqing University, Chongqing, China, in 2015, and the PhD degree from the Department of Computer Science, Tsinghua University, Beijing, China, in 2020. He is currently with Alibaba Inc. His research interests include wireless transport optimization and high-performance networking stack.

Junxian Shen received the BS degree from the Department of Computer Science and Technology, Tsinghua University, Beijing, China, in 2019. He is currently working toward the master's degree at the Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing, China. His research interests include network function virtualization, cloud computing, and serverless computing.

Congcong Miao received the BS degree from the Beijing University of Posts and Telecommunications, Beijing, China, in 2015, and the PhD degree from the Department of Computer Science and Technology, Tsinghua University, Beijing, China, China, in 2020. He is currently a senior engineer with Tencent. His research interests include network measurement and management, machine learning in networking, and cloud computing.

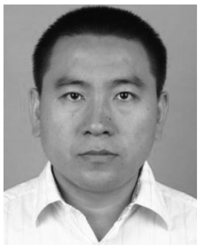
Chen Sun received the BS degree from the Department of Electronic Engineering, Tsinghua University, Beijing, China, in 2014, and the PhD degree from the Department of Computer Science and Technology, Tsinghua University, Beijing, China, in 2019. He is currently a network engineer with Alibaba Group. He has published papers in SIGCOMM, INFOCOM, MM, the *IEEE Journal on Selected Areas in Communications*, the *IEEE/ACM Transactions on Networking*, and so on. His research interests include multimedia transmission, network telemetry, and network function virtualization.



Hongxin Hu (Member, IEEE) received the PhD degree in computer science from Arizona State University, Tempe, Arizona, in 2012. He is an associate professor with the Department of Computer Science and Engineering, University at Buffalo, the State University of New York. His current research interests include security in emerging networking technologies, security in Internet of Things (IoT), machine learning for security and privacy, and security and privacy in social networks. He has published more than 100 refereed technical papers, many of which appeared in top conferences and journals. He received the NSF CAREER Award, in 2019. He is the recipient of the best paper awards from ACSAC 2020, IEEE ICC 2020, ACM SIGCSE 2018, ACM CODASPY 2014, the Best Paper Award Honorable Mentions from ACM SACMAT 2016, IEEE ICNP 2015, and ACM SACMAT 2011.

Jun Bi (Senior Member, IEEE) received the BS, CS, and PhD degrees from the Department of Computer Science, Tsinghua University, Beijing, China. He was a Changjiang Scholar distinguished professor with Tsinghua University and the director of Network Architecture Research Division, Institute for Network Sciences and Cyberspace, Tsinghua University. His previous research interests included Internet architecture, SDN/NFV, and network security. He successfully led tens of research projects, published more than 200 research papers and 20 Internet RFCs or drafts, owned 30 innovation patents, received national science and technology advancement prizes, IEEE ICCCN Outstanding Leadership Award, and best paper awards. He was a distinguished member of the China Computer Federation (CCF).

Jianping Wu (Fellow, IEEE) received the BS, MS, and PhD degrees from Tsinghua University, Beijing, China. He is currently a full professor and the director of the Network Research Center and a PhD supervisor with the Department of Computer Science and Technology, Tsinghua University. Since 1994, he has been an in-charge of the China Education and Research Network. His research interests include the next-generation Internet, IPv6 deployment and technologies, and Internet protocol design and engineering.



Jilong Wang (Member, IEEE) received the PhD degree in computer science from Tsinghua University, Beijing, China, in 1996. He is currently a professor with the Institute for Network Sciences and Cyberspace, Tsinghua University. His research interests include network architecture, network measurement, cyberspace mapping, and cyberspace governance.