IEEE INFOCOM 2024 - IEEE Conference on Computer Communications | 979-8-3503-8350-8/24/$31.00 ©2024 IEEE | DOI: 10.1109/INFOCOM52122.2024.10621097

Non-invasive performance prediction of high-speed softwarized network services with limited knowledge

Qiong Liu*, Tianzhu Zhang†, Leonardo Linguaglossa*
*LTCI, Télécom Paris, Institut Polytechnique de Paris, 91120, Palaiseau, France
†Nokia Bell Labs, 91300, Massy, France

***Abstract*—Modern telco networks have experienced a significant paradigm shift in the past decade, thanks to the proliferation of network softwarization. Despite the benefits of softwarized networks, the constituent software data planes cannot always guarantee predictable performance due to resource contentions in the underlying shared infrastructure. Performance predictions are thus paramount for network operators to fulfill Service-Level Agreements (SLAs), especially in high-speed regimes (e.g., Gigabit or Terabit Ethernet). Existing solutions heavily rely on in-band feature collection, which imposes non-trivial engineering and data-path overhead. This paper proposes a non-invasive performance prediction approach, which complements state-of-the-art solutions by measuring and analyzing low-level features ubiquitously available in the network infrastructure. Accessing these features does not hamper the packet data path. Our approach does not rely on prior knowledge of the input traffic, VNFs' internals, and system details. We show that (i) low-level hardware features exposed by the NFV infrastructure can be collected and interpreted for performance issues, (ii) predictive models can be derived with classical ML algorithms, (iii) and can be used to predict performance impairments in real NFV systems accurately. Our code and datasets are publicly available [1].**

***Index Terms*—Network function virtualization, performance prediction, service function chaining.**

I. Introduction

Modern networks are undergoing an unprecedented paradigm shift with the rise of network softwarization techniques, e.g., SDN and NFV. By replacing high-end, monolithic, proprietary hardware middleboxes with virtualized functions running on Commodity Off-The-Shelf (COTS) servers, this new breed of softwarized networks manages to enact agile, scalable, and efficient service provisioning with hugely reduced capital and operational expenditures. The advent of various packet acceleration stacks, such as Intel Data Plane Development Kit (DPDK) [1] and netmap [2], tremendously leveled up the traffic processing capabilities of software-based solutions, making them rival their hardware counterparts and enabling software network services to process tens of Millions-of-Packets-Per-Second (Mpps) on a single CPU core [3].

However, these benefits come at a cost: compared to traditional hardware middleboxes, the software data plane is more susceptible to unpredictable performance impairments due to contentions in the shared underlying infrastructure [4]. Real-time performance monitoring and prediction constitute the first fundamental step to establish service-level objectives and fully unleash the potential of softwarized networks [5]. Since the inception of the Internet, network monitoring has always been a fundamental ingredient in facilitating network operation, management, and maintenance. Traditionally, engineers inspected network status manually or using simple protocols like SNMP. Modern networks' growing scale and complexity make these approaches impractical. There's a trend towards more automated and intelligent network management systems, which may integrate SNMP data but also rely on additional sources and methods for analysis and decision-making.

Recent solutions commonly employ in-band feature collection for performance analysis. Albeit the optimistic outcomes, they are not always applicable due to the substantial instrumentation and engineering overhead. Moreover, as network services must serve incoming requests in sub-milliseconds, direct feature measurement can dramatically stress the software data plane, resulting in severe performance degradation and SLA violation, e.g., low throughput and high latency.

We propose a novel performance prediction approach for high-speed software data planes to complement existing solutions. Instead of in-band data collection, we consider low-level hardware features, e.g., CPU, memory, and PCI bus, which are ubiquitous in modern COTS servers and can be acquired with standard profiling tools. Albeit appearing less relevant than packet- and flow-level statistics, these features embody real-time network information and can deliver high-level insights via advanced analytics [5]–[7]. With limited knowledge of the network configuration, we collect such features using standard tools and analyze them in real-time with data-driven techniques. Our approach can accurately infer key performance indicators (KPIs), e.g., throughput and latency, based on validations on a real testbed. Compared to state-of-the-art solutions, our approach is lightweight, compatible with existing NFV systems, and adaptable to different network service configurations; it offers a viable entry point for performance diagnosis and bottleneck identification. We highlight that:

- Our approach does not impact the critical data path.
- Our approach can accurately predict throughput and outperforms two state-of-the-art solutions, i.e., SLOMO [5] and Dobrescu [6], reducing the average prediction error by 37% and 70%, respectively.
- Our model can predict the service latency under assorted traffic and network conditions.

This paper is organized as follows: we present the background and the related works about performance analysis for

[1]https://github.com/evesiphus/onvm

high-speed software data plane in Sec. II. We describe the design of our non-intrusive performance prediction approach in Sec. III and present a sensitivity analysis with different network conditions and service configurations in Sec. IV. We demonstrate the applications of our approach in Sec. V and discuss future directions in Sec. VI.

II. Background

A. High-speed softwarized networks

Traditionally, software-based solutions, such as the Click Modular Router [8], were favored for fast prototyping and functional testing thanks to their unparalleled accessibility and flexibility. However, specialized hardware middleboxes had far superior packet processing capabilities and were preferred for deployment in production-ready networks. Recent acceleration techniques (e.g., kernel-bypassing, polling, batching, parallel processing) have narrowed the performance gap between software- and hardware-based solutions [9], making software packet processing an integral part of the telco industry [1].

However, softwarized networks still bear several inherent limitations. In particular, the co-located Virtual Network Functions (VNFs) are susceptible to performance impairments due to the erratic contentions in the shared underlying infrastructure [5]. According to a survey by Gong et al. [10], network operators commonly encounter performance issues, and some spend ≥ 12 hours on performance diagnosis every month. Such problems are intricate to predict with voluminous and heterogenous traffic therein. The growing complexity of the software data plane and network service structure further compounds the situation [11]. As modern COTS servers keep gaining new functionalities, software data plane can have hundreds of configuration knobs, including hardware options and software parameters: this vast search space makes it extremely arduous to anticipate and prevent performance contentions. Furthermore, network service structures are transforming beyond the conventional linear service function chains (SFCs), and many research efforts strive for enhanced service provisioning via VNF parallelization that organizes the VNFs as Directed Acyclic Graphs (DAGs) [12]–[15]. As detailed in [13], 53.8% NF pairs in enterprise networks could be parallelized. Such convoluted interactions across infrastructure and service layers make it exceedingly challenging to predict the occurrence of performance issues [10]. There is an urgent need for novel approaches to monitor and identify performance issues with high accuracy and low overhead.

B. Related works

Existing solutions generally employ in-band network measurement: NFVPerf [16] employs packet mirroring for feature collection but heavily relies on expensive memory copy operations. PPTMon [17] employs event filtering and timestamp embedding to monitor the processing latency of VNFs, but it was not implemented for high-speed scenarios. These works cannot handle the huge input load in high-speed networks, e.g., the throughput can be up to 14.88 Mpps with the end-to-end network service latency in sub-microseconds for 64-byte synthetic packets on a 10 Gbps link.

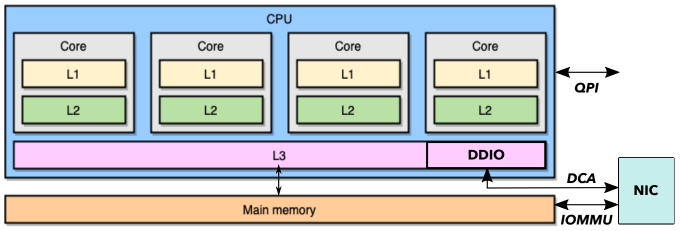


Fig. 1: The architecture of a modern COTS server.

Some solutions are designed for high-speed regimes but are subject to enormous integration overhead, huge resource footprint, and operational constraints. For instance, NFV-VIPP [18] can collect the execution states of DPDK-augmented VNFs but requires manually attaching a thread to each VNF, which is nontrivial in large-scale networks. Additionally, each monitoring thread needs to occupy one CPU core, which is difficult considering the limited number of cores on COTS servers. Microscope [10] and Printqueue [19] leverage the queuing information to deduce performance bottlenecks. However, the queuing occupancy might not always be easy to obtain in real networks - as VNF can come from varied vendors, code instrumentation and customization are mostly necessary, which can be highly burdensome given the diversified implementation and operation patterns.

To circumvent these obstacles, a line of works takes a novel approach by exploring low-level features to derive insights into service performance. In particular, Dobrescu et al. [6] propose extrapolating the performance of the software data plane using the cache features. Layered on this pioneering work, Shelbourne et al. [7] infer throughput and packet losses for singleton VNFs by exploring a larger set of low-level features, but they only analyze the impact of input traffic without considering other system-level performance interferences. Antonis et al. [5] develop SLOMO, a multivariable performance prediction framework. By investigating various system-level contentions to select the relevant features, SLOMO employs the gradient-boosting regression for throughput prediction. Although this work delivers accurate predictions, it is mainly designed for singleton VNFs. Our work digs deeper into end-to-end network services comprising multiple VNFs with different service topologies and employs Artificial Neural Networks (ANNs) to handle the more complex non-linear data patterns.

III. System design

A. Resource contention in a software data plane

A general architecture of a modern COTS server is depicted in Fig. 1, which includes, for simplicity, only the relevant components for packet processing. COTS servers are commonly multi-processor systems with multiple Non-Uniform Memory Access (NUMA) sockets, each with local memory and I/O buses to alleviate memory access contention. To further enhance data/instruction locality, COTS servers employ a multi-level cache design. As shown in Fig. 1, each CPU possesses multiple cores with built-in L1 caches (for both data and instruction) and L2 caches. Cores on the same NUMA node share the same L3 cache with a much larger size than L1/L2 caches. When packets arrive at the Network Interface Controller (NIC), they can be either directed to the main memory via Input–Output Memory Management Unit (IOMMU) or to the L3 cache via Direct Cache Access (DCA) if supported, e.g., the Intel Direct Data I/O (DDIO) cache. Cores on different NUMA nodes can only communicate via specialized interconnect, such as the Intel QuickPath Interconnect (QPI).

Given the complex layout of various components and the intricate interactions of the COTS servers, software packet I/O operations can suffer from multi-faceted interferences. We identify three primary sources of performance interference.

CPU share The allocated CPU share directly decides how fast the incoming packets can be processed. Despite the high frequency of modern CPUs, interference can still be caused by temporary down-/over-clocking (e.g., Intel Turbo boost), which introduces performance variance. Also, other workloads can be allocated to a VNF's cores due to misconfiguration, leading to performance losses. CPU isolation mechanisms, e.g., Linux `isolcpus`) can only alleviate the issue.

Multi-level caches While cache accesses are way faster than main memory, prior works have widely deemed multi-level caches the major performance bottleneckZ [6]. Many NFV frameworks streamline packet processing across multiple cores, which can cause severe contention for the Last-level caches (LLCs). Cache partitioning techniques, such as Intel Cache Allocation Technology (CAT), cannot always prevent such contentions, e.g., large incoming packets can contend for DDIO, which is referred to as the leaky DMA problem [20].

Memory bandwidth The contention for memory bandwidth also contributes to performance degradations on the packet path. For instance, VNFs with lower LLC shares can incur high cache misses, which saturate the memory bandwidth for all the co-located VNFs and network services [21].

Note that other sources of performance interference exist, but they are less impactful than the preceding three. For instance, packet I/O across multiple NUMA nodes can be extremely slow due to the QPI contention [22], which can be avoided with NUMA-aware design (e.g., in DPDK).

B. Design choices and reference architecture

Based on the discussion of the prior works in Sec. II-B, our architecture should respect the following design considerations. First, it must be as *general* as possible, to deliver accurate and efficient predictions of performance impairments for both individual VNFs and ground-up network services with limited knowledge about the deployment. Second, given the ever-increasing complexity of modern networks, it must be *lightweight* and easy to deploy with minimal engineering exertions. As part of the network management subsystem colocated with VNF execution, it should be *noninvasive* and introduce a negligible impact on the software data plane's normal operations and traffic. Finally, it must be *fast*, to enable real-time predictions using the available data.

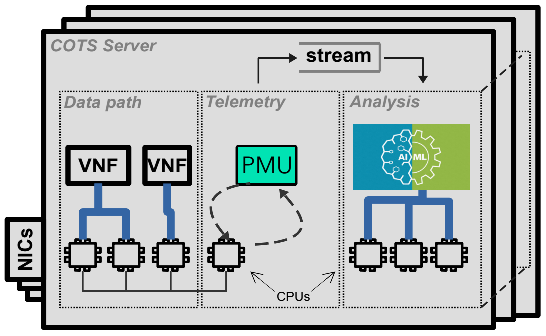


Fig. 2: The proposed architecture for performance prediction

Fig. 2 gives the general illustration of our approach. The prediction workflow consists of two fundamental steps, i.e., data collection and statistical learning. To overcome the limitations of in-band data collection, our approach measures the low-level hardware features of the shared network infrastructure. Although the internals of COTS servers are extremely complex, modern systems commonly offer various toolsets for performance monitoring, namely the Performance Monitoring Units (PMUs), which leverage a set of fixed and programmable counters to record the runtime execution events of a program from various system units, including the instruction pipeline, CPU caches, and configuration registers. This approach has several advantages. First, regardless of the peculiarities of network and system configurations, these features are always available and can be readily collected via standard profiling interfaces. High-level safeguard measures (e.g., encryption, private enclaves) do not hinder the collection of these low-level features. Second, compared to in-band measurement, collecting low-level features only happens at the hardware registers and thus causes much less intervention on the data path. Many low-level features are already available in the system registers, and our approach merely reuses them. This point is especially crucial in high-speed networks since even slight noise can cause noticeable performance losses [23]. Third, our approach does not require an in-depth understanding of the target NFV system components, such as the service, the management & operation (MANO) plane, and the internals of (third-party) VNFs. Operators are thus relieved from extra engineering efforts, code instrumentation, and unit & integration tests.

Once collected, the features are streamlined to retrieve the rich runtime information that they encode. Despite being less intuitive than the packet- and flow-level statistics, they are proven to be useful for insight distillation with proper analytics techniques. Existing solutions generally employ rule-based heuristics for performance analysis. Although these methods are effective in specific settings, their assumptions of the target networks do not always hold, especially given the rapid diversification of modern NFV systems [9]. The last decade has witnessed the prosperity of machine learning techniques, especially Artificial Neural Networks (ANNs), thanks to their outstanding pattern-matching capabilities from multi-dimensional data. Inspired by their historical success,

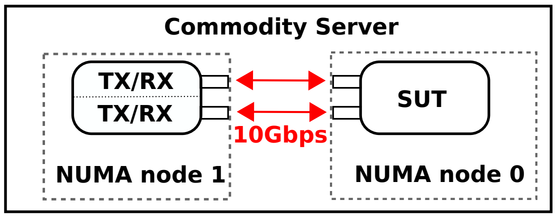


Fig. 3: Testbed environment settings

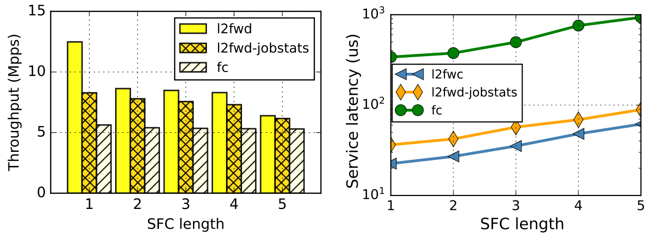


Fig. 4: The overhead of in-band data collection

we also implement an ANN model using the most relevant low-level features for performance prediction.

IV. Sensitivity analysis

This section presents our observational study of non-invasive data collection, focusing on the correlation between the low-level features and two common KPIs: throughput and latency.

A. Testbed environment

Hardware settings Our testbed is illustrated in Fig. 3. We conduct experiments on a COTS server with two Intel (R) Xeon CPUs E5-2660 v3 @ 2.60GHz with a three-level cache of 64K/256K/25600K. Each processor consists of 10 physical cores attached to a NUMA node. To minimize interference, all the cores used for our tests are isolated from the kernel scheduler with hyper-threading and turbo-boost disabled. Each NUMA node also hosts an Intel 82599ES 10-Gigabit dual-port NIC. We use fiber-optic cables to bridge each pair of ports, as highlighted with the red arrows in Fig. 3.

Software settings All the involved software packet processing components are based on the prevalent DPDK software stack (version 18.11.11 LTS). The testbed operates as an open loop: we deploy a TX process on NUMA node 1 to generate packets with designated traffic patterns continuously; the traffic goes through the target network services deployed on NUMA node 0 (Sec. IV-A); and an RX process to measure the end-to-end KPIs. We choose MoonGen [24], a high-speed traffic processing engine, to fulfill the roles of TX and RX. Meanwhile, we collect the low-level hardware features of individual VNFs using *perf* [25], a lightweight standard Linux profiling tool that interfaces with the system PMUs at a configurable frequency (e.g., a few hundred milliseconds). The low-level features are always available for collection using the VNFs' execution identifiers (e.g., process/thread/function IDs).

Network service deployment There are two ways to provision network services on NUMA node 0: (i) a general way that uses software switches to steer traffic across the VNFs to realize the service intent and (ii) a framework-specific way with customized service chaining mechanisms. Our approach works for both settings. For (i), we choose FastClick [26], a high-speed software switch based on the Click modular router, and reuse the off-the-shelf DPDK applications [1] as third-party VNFs. For (ii), we opt for OpenNetVM (ONVM) [3], a state-of-the-art NFV framework featuring flexible network service composition and high-speed packet steering. Any NFV frameworks operating in pipeline mode are also suitable for our study. The instantiated VNFs can operate either in containers or as bare-metal processes, and they can be purposely connected to form the intended network services. Each VNF runs in the busy-polling mode that monopolizes a CPU core (with quasi 100% usage). Our method also applies to VNFs running in interrupt mode, where multiple VNFs co-exist on the same core. Note that both FastClick and ONVM are selected because of their impressive performance and accessibility; our approach is generally applicable to other prevalent software switches (e.g., BESS [27] or Open vSwitch [28]) and NFV frameworks (e.g., ClickOS [29] or E2 [30]).

Workload generation Performance issues generally have two origins: overwhelming loads and insufficient resources [31]. We consider two basic workload generation schemes to expose latent performance issues: *load stimulus* and *resource stimulus*. The former composes the input traffic with special patterns to contrive load contentions, while the latter perturbs the resource shares of individual VNFs to fabricate resource contentions. To better control the imposed contentions, we employ competitor processes to expose and analyze the impact of resource contentions. The competitors are based on *stress-ng* [32], a standard stress-testing tool capable of generating bogus operations at system components, including CPU clock, multi-level caches, and memory, to emulate possible resource contention scenarios. For example, CPU contention can be created by pinning parasite competitors to a VNF's worker core. The cache contentions can be generated by thrashing existing lines. The memory bandwidth contention can be induced by injecting I/O requests. We can even generate multiple contentions by calculatedly assigning the competitors.

B. Data collection overhead

To analyze the impact of data collection on the data path, we first consider a sample SFC of identical VNFs, each performing Layer-2 packet forwarding. The VNFs are containerized with Docker and interconnected using FastClick. We chose three VNFs from the DPDK example library, i.e., *l2fwd, l2fwd-jobstats,* and *flow classification (fc)*. l2fwd only performs packet forwarding. l2fwd-jobstats and fc further collect packet- and flow-level statistics, respectively. MoonGen is configured to continuously inject 64B synthetic traffic to the SFC at line rate and measure the throughput from the receiving end. It also sends Precision Time Protocol (PTP) packets and synthetic traffic to measure the end-to-end latency every second. The

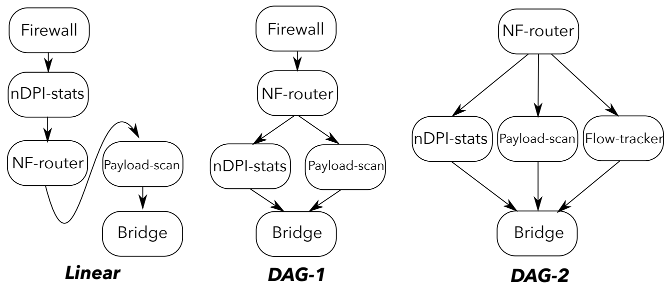


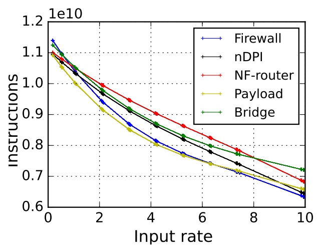
Fig. 5: Three typical network service topologies

SFC length is varied from 1 to 5 [2]. As illustrated in Fig. 4, in-band measurement causes enormous performance degradation. In particular, fc causes the throughput to drop by up to 50% while extending the latency by one order of magnitude.

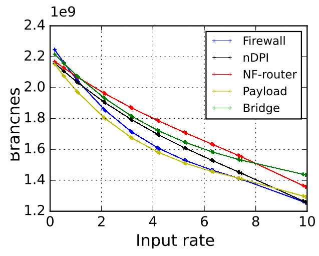
To demonstrate the advantage of our approach, we repeat the exact experiments for l2fwd with the addition of *perf* to collect the low-level features every 100 ms. To test different levels of interactions, we sequentially assign perf to the same worker cores as the VNFs, to different idle cores on the same NUMA node, and cores on the other NUMA node. In all cases, the perceived throughput and latency remain the same. We then inspect the framework-specific deployment and repeat the same tests for an SFC instantiated with varied lengths on ONVM. The observations for *perf*'s negligible impact still hold.

Observation 1. *Data collection via PMUs is non-intrusive and imposes negligible overhead on the data path compared to most existing solutions based on in-band data collection.*

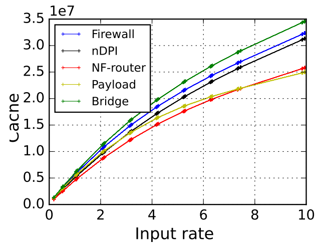
C. Sensitivity analysis: input traffic and service topology



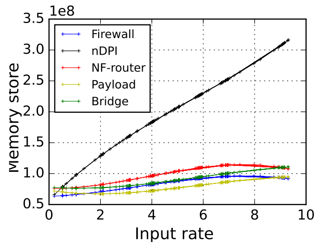
(a) Instructions



(b) Branches



(c) Cache references



(d) Memory stores

Fig. 6: Low-level features as a function of the input rate.

[2]Note that the performance deteriorates as the SFC gets longer, mainly due to the accumulated memory copying and inter-core communication overhead [23].

Perf collects hundreds of hardware features: as unrelated features can inject noise that harms an ML model's effectiveness, we must identify a refined subset of expressive features with strong predictive power. Hardware features that already encode information at the system level are treated in our approach as intermediate variables between the input traffic and the end-to-end KPIs. We now investigate their evolution with different input traffic and service topologies and show our insights into their impact on the output KPIs in Sec. IV-D.

We construct three typical network service topologies (or SFCs): a linear service chain and two directed acyclic graphs (DAGs), as shown in Fig. 5. The VNFs include *Firewall, nDPI-stat, NF-router, Payload-scan, Bridge,* and *Flow-tracker*, all implemented and open-sourced by ONVM developers. We then inject random traffic with a rate between 0-10 Gbps and inspect the features' tendencies.

Fig. 6 highlights the tendencies of four relevant features for the linear SFC, i.e., the number of instructions, branches, cache references, and memory accesses per time unit, each representing a critical factor of the CPU execution pipeline. The instruction rates drop with higher input rates, as shown in Fig. 6a, which aligns with the execution states of a CPU pipeline and packet I/O mechanism. When a VNF performs busy polling with only a few incoming packets, its execution pipeline is mostly populated with simple instructions that do not involve complex packet processing logic, and low-level parallelism allows the CPU to execute more than one instruction per clock cycle. With higher packet rates, the packet processing logic is invoked more frequently, leading to a lower instruction rate due to the increased code complexity. Similarly, the branches also follow this reasoning. The caches and memory constitute a major performance bottleneck, as identified by prior works [5], [6]. This observation also holds on our servers: as shown in 6c and Fig. 6d, the cache-reference rate and memory access rate (mem-stores) positively correlate with the input rate. Note that as the nDPI-stats requires frequent memory writes, its mem-stores are exceptionally higher than other VNFs. We replicate the same analysis with the two DAGs, and similar trends can still be observed.

Observation 2. *Some hardware features trend closely with the input traffic regardless of the service topologies, making them candidates for the intermediate variables between the input traffic and output KPIs. They can also carry information about the unique execution patterns of individual VNFs.*

D. Sensitivity analysis: KPIs

To locate the relevant features, we opt for Pearson's correlation coefficient to assay the statistical dependencies between the collected features and the KPIs, i.e., throughput and latency. To represent typical Internet traffic, we configure MoonGen to generate IMIX traffic that consists of a variety of packet sizes with the ratio 64B: 570B: 1514B = 7: 4: 1. We collect the low-level features and performance metrics for different service topologies under both load and resource stimuli tests.

Table I and Table II list the correlated features with different KPIs for the linear SFC. The results are coherent with our

Features \ VNF	Bridge	Payload-scan	NF-router	nDPI	Firewall	Average
LLC-load	0.94	0.98	0.97	0.98	0.96	0.97
Cache-reference	0.95	0.97	0.97	0.98	0.97	0.94
LLC-stores	0.97	0.96	0.96	0.97	0.97	0.97
L1-Dcache-load-misses	0.95	0.97	0.97	0.98	0.97	0.97
Instructions	0.79	0.92	0.86	0.78	0.89	0.92
Branches	0.79	0.92	0.87	0.79	0.89	0.83
Mem-stores	0.39	0.50	0.42	0.90	0.53	0.55
Cache Misses	0.32	0.18	0.35	0.65	0.36	0.38
Cycles	0.14	0.08	0.14	0.14	0.06	0.12

(a) Throughput

Features	Average
LLC-load	0.59
Cache-reference	0.58
LLC-stores	0.57
L1-Dcache-load-misses	0.55
Instructions	0.48
Branches	0.47
Mem-stores	0.11
Cache Misses	0.13
Cycles	0.02

(b) Latency

TABLE I: Correlated features with throughput and latency under load stimulus

Features \ VNF	Bridge	Payload-scan	NF-router	nDPI	Firewall	Average
LLC-load	0.80	0.67	0.55	0.54	0.44	0.60
Cache-reference	0.81	0.73	0.58	0.46	0.45	0.61
LLC-stores	0.80	0.74	0.60	0.45	0.32	0.58
L1-dcache-load-misses	0.81	0.72	0.58	0.45	0.45	0.60
Cycles	0.35	0.30	0.29	0.25	0.22	0.28
Instructions	0.21	0.21	0.24	0.21	0.19	0.21
Branches	0.21	0.21	0.24	0.21	0.19	0.21
Mem-stores	0.12	0.08	0.04	0.73	0.03	0.20
Cache Misses	0.00	0.06	0.04	0.03	0.05	0.03

(a) Throughput

Features	Average
LLC-load	0.31
Cache-reference	0.23
LLC-stores	0.18
Branches	0.17
L1-Dcache-load-misses	0.15
Cycles	0.14
Instructions	0.13
Mem-stores	0.12
Cache Misses	0.02

(b) Latency

TABLE II: Correlated features with throughput and latency under resource stimulus

tendency observations. There is no dominant feature that consistently achieves the highest correlation across different VNFs, which suggests the joint impacts of multiple features on the SFC performance. Note that similar feature correlations have also been observed with the DAG topologies; we omit them for space's sake.

Correlation with throughput Under load stimulus, the features at the top of Table. Ia show homogeneously high correlations with the throughput over different VNFs. In particular, cache-related features, especially cache-reference rate and L1-dcache-load-misses, strongly correlate with the throughput. In contrast, as shown in Table. IIa, the correlation of those same features under resource stimulus shows an ascending pattern: the correlation is small at the beginning of the chain, and increases towards the end. For most of the features, the last VNF (i.e., bridge) presents the highest overall correlation with the throughput. We observe that certain VNF-specific behaviors contribute to individual features' peculiarities. For instance, under both stimuli tests, nDPI's mem-stores feature shows a very high correlation with throughput, because this particular VNF requires frequent memory accesses.

Correlation with latency As shown in Table Ib and Table IIb, the cache-related features show the highest correlation with the latency. We also observe that latency has relatively weaker correlations with the hardware features than throughput. As the latency is measured as the round-trip time of the PTP packets, it is very susceptible to high-level system events, such as buffer overflow and bandwidth saturation, that are less intuitive to capture with the low-level features. We will detail our method to infer latency in Sec. V-C.

V. APPLICATIONS

This section presents two applications for our approach, i.e., inferring the throughput and latency of network services. We choose a lightweight ANN because of its outstanding performance in non-linear pattern-matching, generalizability to new data, continuous learning, and scalability to large datasets.

A. General workflow

We conceptualize a blueprint for performance prediction based on network stimuli tests. Specifically, load stimulus measures the pressure of traffic competition, and resource stimulus models how susceptible an SFC is to performance degradation due to the competitors' interference. The combination of them can also expose more performance issues. Building on these concepts, we realize a practical workflow consisting of two logical parts: (1) offline profiling for building the ANN model and (2) online inference for performance predictions.

We denote a network service as $S(V_s, topo_s)$, where V_s represents the constituent VNFs $= \{\text{VNF}_1^s, \cdots, \text{VNF}_k^s\}$ and $topo_s$ their topological composition. A network stimulus is denoted as $stimu_j$. We first initialize offline profiling to collect features and KPIs for the given $(S, stimu_j)$. Then, we exclude the unrelated features for the target KPIs (throughput or latency), according to Sec. IV-D. The collected data are proportionally decomposed into training (60%), validation (20%), and test (20%) datasets. We define $J(\boldsymbol{\theta})$ as the loss function to represent the average loss over the training samples $\boldsymbol{x} = \{x_1, x_2, \cdots, x_n\}$. Let $\mathbb{P}(\cdot)$ denote the probability under the underlying distribution, $\mathbb{E}_X(\cdot)$ denotes the expectation over the random variable X, and the Euclidean norm is denoted as $\|\cdot\|$. The penalty for a mismatch is calculated as follows:

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x},y\sim p_{\text{data}}} L(\boldsymbol{x}, y, \theta) = \frac{1}{n}\sum_{i=1}^{n} L(\boldsymbol{x}_i, y_i, \boldsymbol{\theta}) \quad (1)$$

where p_{data} denotes the real data distribution, $L(\boldsymbol{x}_i, y_i, \boldsymbol{\theta}) = -\log \mathbb{P}(y|\boldsymbol{x};\boldsymbol{\theta})$ the loss of sample $\boldsymbol{x}_i$, and n the sample size. Since the features have a wide range of scales (i.e., $[10^2 - 10^{10}]$), we use L2 regularization to avoid overfitting and improve generalization; the modified penalty is:

$$\hat{J}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} L(x_i, y_i, \boldsymbol{\theta}) + \frac{\alpha}{2}\|\boldsymbol{\theta}\|_2^2 \quad (2)$$

Where $\alpha \in [0, +\infty)$ balances the norm penalty term and the original objective. Note that $\alpha \to 0$ means no regularization, and the regularization penalty becomes larger as α increases. The corresponding gradient is:

$$\nabla_{\boldsymbol{\theta}} \hat{J}(\boldsymbol{\theta}) = \alpha\boldsymbol{\theta} + \nabla_{\boldsymbol{\theta}} \hat{J}(\boldsymbol{\theta}) \quad (3)$$

To solve (3), we use the Adaptive Moment Estimation (Adam) technique [33] to adjust each parameter's learning rate based on its past gradients and squared gradients. The objective is to minimize (2) when (3) goes to 0. The obtained model $\text{ANN}(S, stimu_j)$ is then deployed for online inference.

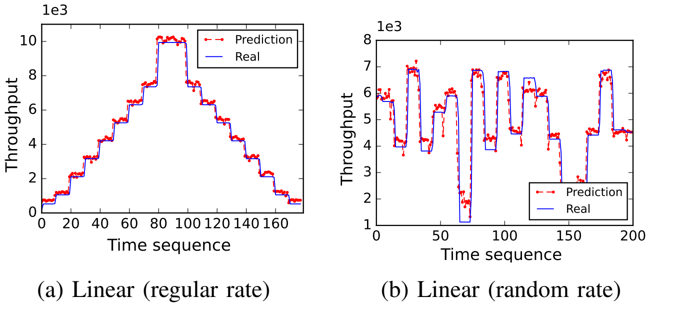
Hyperparameters considerations We narrowed our search space to develop an ANN that balances computational efficiency with performance. Our study utilizes a subtype of feedforward neural networks - the Multilayer Perceptron (MLP) - to tackle regression tasks influenced by numerous features. We employed the grid search methodology to identify the most effective ANN architecture. Unlike more intricate optimization algorithms such as random search or Bayesian optimization, grid search is advantageous for its simplicity and transparency. Due to space constraints, not all search results are reported in this paper. For instance, we found that the number of nodes and hidden layers has less impact on overall accuracy than the factors, e.g., optimizer, activation function, and batch size. We thus limited the number of hidden layers in the ANN to be ≤ 4, and the number of nodes per layer to be ≤ 64. The parameters of our defined search space are as follows.

- Hidden layers - 1-4
- Nodes per layer - 16, 32, and 64
- Activation function - ReLU and tanh for load stimulus, Sigmoid and ReLu for resource stimulus
- Optimizer - Adam and Stochastic Gradient Descent (SGD)
- Learning rates - adapted rate in 0.1, 0.01, and 0.001
- Number of epochs - 10-200 in interval 10
- Batch size - multiples of 8 to 128
- Regularization Drouput (probability of 0.1 to 0.5)

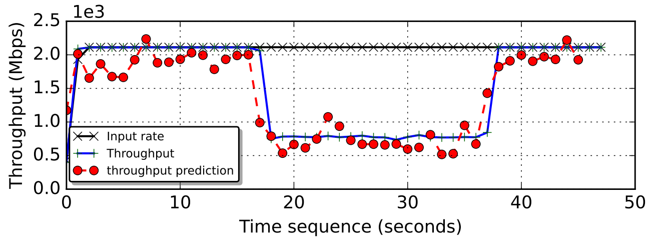
In the following sections, we only detail the generalizability and interpretability analyses for throughput prediction, but the observations equally apply to the other use cases.

B. Throughput prediction

Feature selection We build a refined feature set from the original one via the correlation analysis of Sec. IV-D. Then, we train ANN models on both sets and calculate their prediction errors. By removing the irrelevant features, the ANN's accuracy increases by 15%, while shrinking the training time by 40%, which justifies the need for a preliminary feature selection.



(a) Linear (regular rate) (b) Linear (random rate)



(c) Resource stimulus

Fig. 7: Throughput prediction

Accuracy Under load stimulus, our model achieves impressive overall accuracy: 98% for the regular rate and 92% for the random rate, as illustrated by the examples in Fig. 7a and 7b. Under resource stimulus, throughput prediction becomes more intricate due to diversiform contentions from the CPU, caches, and memory buses. Still, our model's accuracy remains commendable at 83%, as illustrated by the example in Fig. 7c.

Comparison with state-of-the-art Prior works on NFV throughput prediction consider the memory subsystems as the major performance bottleneck. In particular, Dobrescu et al. [6] employ linear models to estimate throughput based on cache access rate. However, these methods prove inadequate with newer hardware architectures. Also, they assume additive effects for multi-traffic contentions, which curbs the prediction power under more complex scenarios. SLOMO [5] achieves the contention-aware throughput prediction in the singleton VNF scenario via gradient boosting regression (GBR). SLOMO considers the joint effects of caches (e.g., LLC) and memory bandwidth. Nonetheless, it does not consider the complexities under SFC settings, and GBR is less effective with non-linear

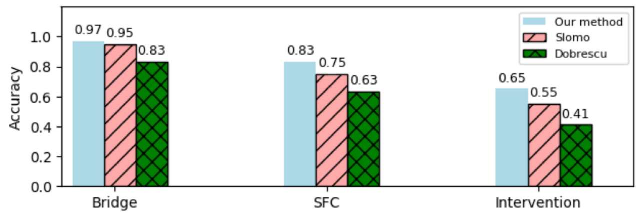
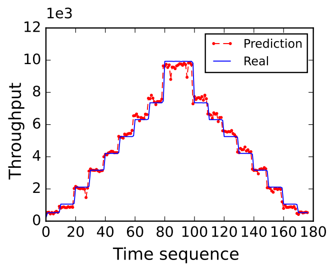
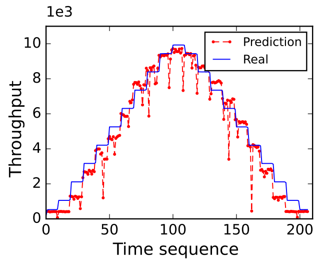


Fig. 8: Throughput prediction comparison



(a) DAG-1



(b) DAG-2

Fig. 9: Robustness across different topologies

patterns than ANN. We compare our throughput prediction model against these two related works.

In Fig. 8, we investigate three scenarios with increasing complexity. **Scenario 1**: Singleton VNF (i.e., ONVM bridge) under random load and resource stimuli; **Scenario 2**: Linear SFC, as defined in Figure 5, under random resource stimulus; **Scenario 3**: Same SFC as Scenario 2 under random load and resource stimuli. Although scenario 3 is hardly realistic, it provides rich contention samples to ascertain a KPI predictor's effectiveness. Our feature set differs slightly from SLOMO by excluding L2 cache and L3 occupancy features. This discrepancy does not affect the overall accuracy as we have already included abounding highly pertinent cache features.

In scenario 1, Our model can accurately predict throughput, with a mean accuracy of 97%, which outperforms SLOMO's 95% and Dobrescu's 83%; Overall, our model increases the prediction accuracy concerning Dobrescu and SLOMO by 18% and 7% on average, respectively, and reduces the average prediction error by 70% and 37%. Note that under the most challenging scenario 3, our model's accuracy is 65%, which we believe can benefit from further data and model enhancements. We leave this as future work.

Generalizability To evaluate the robustness of our model in uncharted scenarios, We test the accuracy of the ANN model (trained for the linear SFC) in the DAG topologies. As exemplified in Fig. 9a and 9b, our model can effectively generalize to different topologies, making it more suitable to deal with the dynamics in real networks. Note that the accuracy of DAG-1 is better than DAG-2 due to the involvement of an unseen VNF (i.e., Flow-tracker) in DAG-2.

Explainability Although neural networks are generally blackboxes for human cognition, their predictions can still be interpreted with advanced techniques, such as eXplainable AI (XAI). Our ANN model has a shallow architecture, making it easily accessible to XAI algorithms. The interpretation can then be combined with domain knowledge to extract actionable insights. As an illustrative example, we attribute the feature importance of our model under resource stimulus using the SHAP algorithm [34]. We specifically perturb the CPU shares of the VNFs of the linear SFC. Fig. 10 lists the normalized SHAP values quantifying the features' contributions to the final predictions. We notice that, albeit the cache features contribute the most, as expected, the influences of the "*_cycles" features are surprisingly high. Network engineers

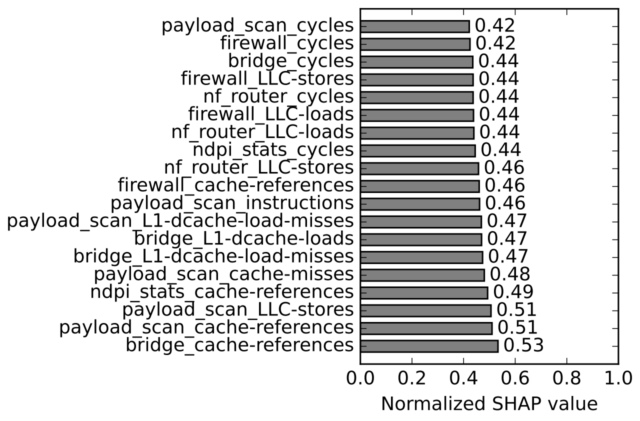
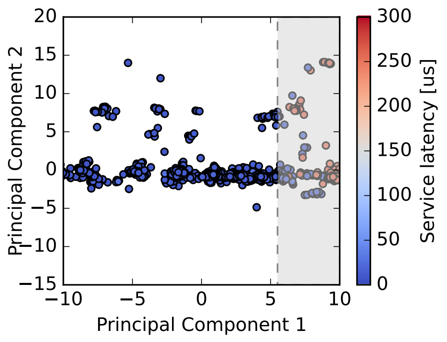
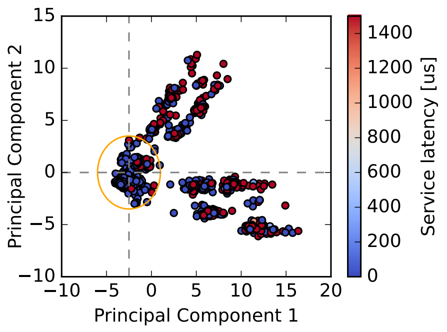


Fig. 10: XAI of resource stimulus



(a) Load stimulus



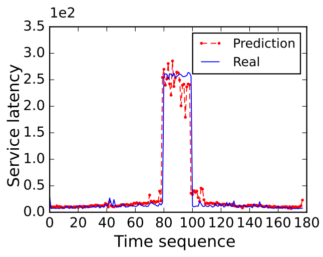
(b) Resource stimulus

Fig. 11: PCA for end-to-end service latency

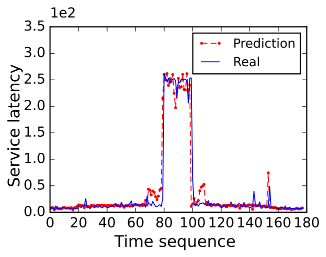
can immediately deduce the presence of parasite processes or malfunctioning CPU frequency-scaling governors. Other performance bottlenecks can be identified in a similar fashion.

C. Latency prediction

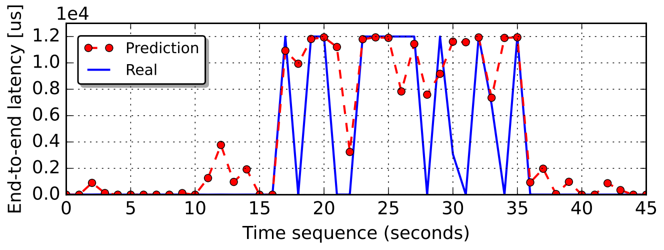
Latency prediction is intrinsically complicated due to more drastic state transitions under congestion, and pairwise correlation analysis is inadequate to uncover the complicated relationships, as discussed in Sec. IV-D. To better understand its predictability, we first conduct Principal Component Analysis (PCA) to explore the feature space, as shown in Fig. 11. We observe that two principal components carry enough information to separate the data points according to the latency. In Fig. 11a, we underline that the average service latency is



(a) Linear



(b) DAG-1



(c) Resource stimulus

Fig. 12: Latency prediction

around $50\mu s$ under load stimulus. The high-latency points are mostly clustered between 5 and 10 on the x-axis. Therefore, it is possible to periodically gather the hardware features and detect whether the network service is facing high latency via PCA. In Fig. 11b, we show PCA results under resource stimulus, and the low latency regions are clustered within a circular region centered on $(-5, 0)$. An interesting phenomenon is that outside the circular region, latency oscillates acutely between low levels (below $200\mu s$) and extreme values (above $1200\mu s$) since the artificially generated severe resource contentions lead to extremely high packet delays and jitters.

Following the general workflow, we build an ANN model based on these observations to predict latency. As illustrated by the two examples in Fig. 12a and 12b, this model can effectively predict the latency for the linear (86%) and DAG-1 (79%) under load stimulus. The prediction error is relatively large at high-speed regimes (around 80-100s of the x-axis), as the latency packet's round-trip time (RTT) is less predictable under network congestion. Fig. 12c shows an example of latency prediction under resource stimulus. We observe that the resource contention occurs between 15-35s, which leads to a nearly 40% throughput drop, hence significant packet losses. Note that we configure MoonGen to wait for a maximum of $120ms$ before assuming a latency packet is lost. During the contention period, the latency experiences significant fluctuations. Consequently, current methods are unsuitable for predicting the exact (artificial) latency under severe network congestion, as sporadic packet losses make the RTT hard to quantify. However, our model can still approximate the expected values with prior knowledge of the maximum latency and predict abnormal service latency and congestion periods.

VI. Conclusion and Future Directions

As network softwarization keeps gaining momentum, there is an urgent need for stable and predictable performance in the software data plane. Existing solutions for network performance diagnostics commonly rely on in-band data collection, which requires tremendous engineering overhead and interferes with the critical data path. We propose a novel approach that utilizes low-level hardware features for KPI prediction. Compared to in-band data collection, our approach is easily applicable to real-world NFV systems without an in-depth understanding of their implementation details. The low-level data collection imposes a negligible impact on the software data plane. We implement an ANN model that can accurately infer throughput and latency in high-speed networks. Our model is generalizable to network services with similar topological compositions, and its predictions can be interpreted with domain-specific knowledge to identify performance bottlenecks.

This paper presents our initial attempt for performance diagnoses using infrastructure-level features. Despite the optimistic results, the system settings are still preliminary and the ML workflow is not production-ready. We thus plan to extend the current work from the following directions:

Broader NFV system settings: Our current work is mainly evaluated using openNetVM and FastClick. We will further consider other prevalent NFV frameworks. Also, we plan to integrate more system profiling tools beyond perf, e.g., Intel PCM, Intel VTune profiler, and AMD uprof. Moreover, we only consider SFCs running in the pipeline mode and plan to cover SFCs in the run-to-completion mode. In addition, our current work only employs DPDK as the acceleration stack. We will evaluate other kernel-bypassing (e.g., Netmap [2]) or in-kernel techniques (e.g., eBPF/XDP). Furthermore, our current work considers VNFs running as bare-metal processes or inside Docker containers. Our approach's applicability to other virtualization techniques, e.g., virtual machines, should be examined. Similarly, we will adapt our method to protected execution environments, e.g., Intel SGX and its variants.

Multi resource contentions scenarios: We intend to broaden the scope of our framework, transitioning from single-node to network-wide performance predictions, thereby offering a more holistic view of network conditions. Besides, we will proceed to identify the performance bottlenecks, including the contentions for DDIO, LLC, CPU share, and memory bandwidth. The contention injection scheme should also be enriched to generate high-quality, representative data.

Automation monitoring and tuning: The present approach relies heavily on manual tuning without systematic logging, making the entailed datasets, parameters, and configuration dependencies difficult to trace. It is equally essential to automate the entire ML workflow, converting it into an end-to-end data processing pipeline with continuous data collection, model development, deployment, serving, and monitoring.

Acknowledgements

This work has been partially carried out in the context of the following projects: *a.* ANR project IONOS-DX (ANR-22-CE25-0003); *b.* Beyond5G, as part of the economic recovery plan "France Relance" and the investments for the future program. We thank Dr. Fabio Pianese for helping us form this work's initial ideas.

References

[1] "Intel DPDK," https://www.dpdk.org/, last accessed July 2023.

[2] L. Rizzo, "netmap: A novel framework for fast packet I/O," in *2012 USENIX Annual Technical Conference*, 2012, pp. 101–112.

[3] W. Zhang, G. Liu, W. Zhang, N. Shah, P. Lopreiato, G. Todeschi, K. Ramakrishnan, and T. Wood, "OpenNetVM: A platform for high performance network service chains," in *Proceedings of the 2016 workshop on Hot topics in Middleboxes and Network Function Virtualization*, 2016, pp. 26–31.

[4] W. Wu, K. He, and A. Akella, "Perfsight: Performance diagnosis for software dataplanes," in *Proceedings of the 2015 Internet Measurement Conference*, 2015, pp. 409–421.

[5] A. Manousis, R. A. Sharma, V. Sekar, and J. Sherry, "Contention-aware performance prediction for virtualized network functions," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 270–282.

[6] M. Dobrescu, K. Argyraki, and S. Ratnasamy, "Toward predictable performance in software packet-processing platforms," in *9th USENIX Symposium on Networked Systems Design and Implementation*, 2012, pp. 141–154.

[7] C. Shelbourne, L. Linguaglossa, T. Zhang, and A. Lipani, "Inference of virtual network functions' state via analysis of the CPU behavior," in *2021 33th International Teletraffic Congress*, 2021, pp. 1–9.

[8] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click modular router," *ACM Transactions on Computer Systems*, vol. 18, no. 3, pp. 263–297, 2000.

[9] T. Zhang, H. Qiu, L. Linguaglossa, W. Cerroni, and P. Giaccone, "NFV platforms: Taxonomy, design choices and future challenges," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 30–48, 2020.

[10] J. Gong, Y. Li, B. Anwer, A. Shaikh, and M. Yu, "Microscope: Queue-based performance diagnosis for network functions," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 390–403.

[11] P. Zheng, W. Feng, A. Narayanan, and Z.-L. Zhang, "NFV performance profiling on multi-core servers," in *2020 IFIP Networking Conference*. IEEE, 2020, pp. 91–99.

[12] H. Li, Y. Dang, G. Sun, G. Liu, D. Shan, and P. Zhang, "LemonNFV: Consolidating heterogeneous network functions at line speed," in *20th USENIX Symposium on Networked Systems Design and Implementation*, 2023, pp. 1451–1468.

[13] C. Sun, J. Bi, Z. Zheng, H. Yu, and H. Hu, "NFP: Enabling network function parallelism in NFV," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, 2017, pp. 43–56.

[14] Y. Zhang, B. Anwer, V. Gopalakrishnan, B. Han, J. Reich, A. Shaikh, and Z.-L. Zhang, "Parabox: Exploiting parallelism for virtual network functions in service chaining," in *Proceedings of the Symposium on SDN Research*, 2017, pp. 143–149.

[15] X. Lin, D. Guo, Y. Shen, G. Tang, and B. Ren, "Dag-sfc: Minimize the embedding cost of sfc with parallel vnfs," in *Proceedings of the 47th International Conference on Parallel Processing*, 2018, pp. 1–10.

[16] P. Naik, D. K. Shaw, and M. Vutukuru, "NFVPerf: Online performance monitoring and bottleneck detection for NFV," in *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks*. IEEE, 2016, pp. 154–160.

[17] N. Van Tu, J.-H. Yoo, and J. W.-K. Hong, "PPTMon: Real-time and fine-grained packet processing time monitoring in virtual network functions," *IEEE Transactions on Network and Service Management*, vol. 18, no. 4, pp. 4324–4336, 2021.

[18] M. Dodare, Y. Taguchi, R. Kawashima, H. Nakayama, T. Hayashi, and H. Matsuo, "NFV-VIPP: Catching internal figures of packet processing for accelerating development and operations of nfv-nodes," in *2019 15th International Conference on Network and Service Management*, 2019, pp. 1–4.

[19] Y. Lei, L. Yu, V. Liu, and M. Xu, "PrintQueue: performance diagnosis via queue measurement in the data plane," in *Proceedings of the ACM SIGCOMM 2022 Conference*, 2022, pp. 516–529.

[20] A. Tootoonchian, A. Panda, C. Lan, M. Walls, K. Argyraki, S. Ratnasamy, and S. Shenker, "{ResQ}: Enabling {SLOs} in network function virtualization," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018, pp. 283–297.

[21] V. R. Chintapalli, S. B. Korrapati, M. Adeppady, B. R. Tamma, B. R. Killi *et al.*, "NFVPermit: Towards Ensuring Performance Isolation in NFV-based Systems," *IEEE Transactions on Network and Service Management*, 2023.

[22] Z. Niu, H. Xu, L. Liu, Y. Tian, P. Wang, and Z. Li, "Unveiling performance of NFV software dataplanes," in *Proceedings of the 2nd Workshop on Cloud-Assisted Networking*, 2017, pp. 13–18.

[23] T. Zhang, L. Linguaglossa, M. Gallo, P. Giaccone, L. Iannone, and J. Roberts, "Comparing the performance of state-of-the-art software switches for NFV," in *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, 2019, pp. 68–81.

[24] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle, "MoonGen: A scriptable high-speed packet generator," in *Proceedings of the 2015 Internet Measurement Conference*, 2015, pp. 275–287.

[25] "perf: Linux profiling with performance counters," https://perf.wiki.kernel.org/index.php/Main_Page, last accessed July 2023.

[26] T. Barbette, C. Soldani, and L. Mathy, "Fast userspace packet processing," in *2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems*. IEEE, 2015, pp. 5–16.

[27] S. Han, K. Jang, A. Panda, S. Palkar, D. Han, and S. Ratnasamy, "SoftNIC: A software nic to augment hardware," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2015-155*, 2015.

[28] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar *et al.*, "The design and implementation of Open vSwitch," in *12th USENIX symposium on networked systems design and implementation*, 2015, pp. 117–130.

[29] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici, "ClickOS and the art of network function virtualization," in *11th USENIX Symposium on Networked Systems Design and Implementation*, 2014, pp. 459–473.

[30] S. Palkar, C. Lan, S. Han, K. Jang, A. Panda, S. Ratnasamy, L. Rizzo, and S. Shenker, "E2: A framework for NFV applications," in *Proceedings of the 25th Symposium on Operating Systems Principles*, 2015, pp. 121–136.

[31] A. Aghasaryan, M. Bouzid, and D. Kostadinov, "Stimulus-based sandbox for learning resource dependencies in virtualized distributed applications," in *2017 20th Conference on Innovations in Clouds, Internet and Networks*. IEEE, 2017, pp. 238–245.

[32] "Kernel/Reference/stress-ng - Ubuntu Wiki," https://wiki.ubuntu.com/Kernel/Reference/stress-ng, last accessed July 2023.

[33] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[34] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," *Advances in neural information processing systems*, vol. 30, 2017.