Contention-Aware Performance Prediction For Virtualized Network Functions

Antonis Manousis, Rahul Anand Sharma, Vyas Sekar, Justine Sherry
Carnegie Mellon University

ABSTRACT

At the core of Network Functions Virtualization lie Network Functions (NFs) that run co-resident on the same server, contend over its hardware resources and, thus, might suffer from reduced performance relative to running alone on the same hardware. Therefore, to efficiently manage resources and meet performance SLAs, NFV orchestrators need mechanisms to predict contention-induced performance degradation. In this work, we find that prior performance prediction frameworks suffer from poor accuracy on modern architectures and NFs because they treat memory as a monolithic whole. In addition, we show that, in practice, there exist multiple components of the memory subsystem that can separately induce contention. By precisely characterizing (1) the pressure each NF applies on the server's shared hardware resources (contentiousness) and (2) how susceptible each NF is to performance drop due to competing contentiousness (sensitivity), we develop SLOMO*, a multivariable performance prediction framework for Network Functions. We show that relative to prior work* SLOMO *reduces prediction error by 2-5× and enables 6-14% more efficient cluster utilization.* SLOMO*'s codebase can be found at https://github.com/cmu-snap/SLOMO.*

CCS CONCEPTS

• **Networks** → Network Performance Evaluation;

ACM Reference Format:

Antonis Manousis, Rahul Anand Sharma, Vyas Sekar, Justine Sherry. 2020. Contention-Aware Performance Prediction For Virtualized Network Functions. In *Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication (SIGCOMM '20), August 10–14, 2020, Virtual Event, NY, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3387514.3405868

KEYWORDS

Network Functions Performance; Packet Processing Software

1 INTRODUCTION

Network Function Virtualization (NFV) entails implementing Network Functions (NFs) in software on shared, general-purpose infrastructure [5, 41]. In this vision, NF instances are spun up and down and migrated between servers. In order to reduce cost and be resource efficient, multiple virtualized NFs are *co-resident* on the same server hardware.

SIGCOMM '20, August 10–14, 2020, Virtual Event, NY, USA
© 2020 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-7955-7/20/08.
https://doi.org/10.1145/3387514.3405868

Unfortunately, co-resident NFs can interfere with each other as they share hardware resources, primarily in the memory subsystem [47–49]. As a result, NF performance (*i.e.,* throughput and latency) can degrade relative to when they run alone.

This contention-induced performance drop suggests that to ensure that Service Level Agreements are met, NFV orchestration systems (*e.g.,* AT&T Domain 2.0 [3]) need suitable *performance prediction* mechanisms. Such predictions can inform NFV orchestration tasks such as NF provisioning, deployment, and auto-scaling. For example, before launching a new NF on a server that is already running other instances, knowing if and by how much the performance of each one of the co-runners will suffer can help make better run-time decisions. Given the infeasibility of profiling all possible combinations of NF configurations ahead of time, we need systematic *contention-aware* techniques for performance prediction.

Prior works in NFV performance prediction build on the observation that the memory subsystem is the root cause of performance degradation [24, 38, 39]. These works use linear models to correlate a variable associated with cache utilization of a competitor (*e.g.,* working set size or cache access rate) with net slowdown for a competing target NF. However, we observe that under newer hardware architectures and line rates reaching up to 100Gbps, this line of prior work proves to be inadequate, with prediction errors as high as 70% (§2). This motivates us to systematically analyze the memory subsystem and understand the various sources of contention to enable better prediction.

Our analysis (§3) indicates that with state-of-the-art servers and NFs, contention manifests *simultaneously and independently* at three different chokepoints across the memory subsystem: the packet I/O subsystem that delivers packets into the last-level cache (LLC), evictions from the last-level cache, and the main memory access bandwidth. Thus, we argue that we need to revisit contention aware modeling, exploring both what metrics we use to measure cache utilization of the competitor and what models we use to correlate these metrics with slowdown – now from a more challenging, multivariable perspective.

To guide our exploration, we follow a blueprint from the architecture community [38, 39, 49, 50] based on the (1) *contentiousness* that captures the pressure a NF places on shared hardware and (2) *sensitivity* or how susceptible a NF is to performance degradation as a function of the competitors' aggregate contentiousness. The blueprint, however, leaves open three key technical challenges: identifying suitable domain-specific metrics for capturing contentiousness; evaluating models for measuring sensitivity; and ensuring composability of the contentiousness metrics, *i.e.,* that given the

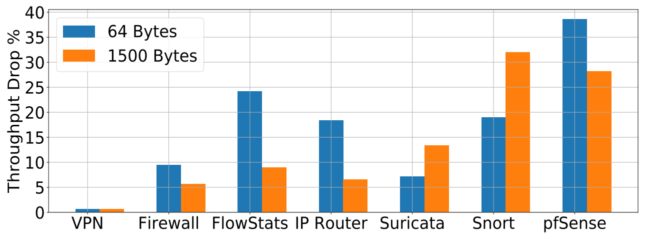

Figure 1: *Contention-induced throughput drop (Maximum observed drop across experiments with real competitors).*

independent contentiousness of an NF_A and an NF_B, one can compute the combined contentiousness of NF_A and NF_B imposed upon a third NF_C. Indeed, because this blueprint is so generalizable, prior NF performance prediction approaches can be viewed as adhering to this blueprint as well [20, 24, 38].

Using a data-driven approach, we tackle the above technical challenges as follows.

Contentiousness Metrics: As contention happens simultaneously and independently at multiple points across the memory subsystem, contentiousness should be measured across multiple dimensions. We show that a carefully selected set of hardware counters, measuring resource utilization at CPU-socket- and server-level granularities are sufficient to quantify the contentiousness of a mix of competing NFs.

Sensitivity Modeling: While the overall sensitivity of a NF is a non-linear and non-continuous function of a multivariable contentiousness vector, we show that it can be accurately modeled as a piecewise function of linear models using *ensemble* techniques from the machine learning literature [22, 54].

Composition: Aggregate contentiousness metrics can be composed using simple (*e.g.,* avg, addition) functions allowing us to estimate the combined contentiousness of any combination of real NFs.

Building on these insights, we design a new NFV performance prediction framework called SLOMO. Our results show that SLOMO's throughput predictions are accurate, with an average end-to-end prediction error $\leq 8\%$, while reducing by half the prediction error of prior work. Furthermore, we demonstrate that SLOMO's prediction can improve cluster efficiency by 6-14% when combined with cluster schedulers designed to meet SLA guarantees.

2 BACKGROUND AND MOTIVATION

We begin by describing the NFV performance prediction problem (§2.1). Then, we show that prior work suffers poor accuracy with modern hardware and line rates (§ 2.2).

2.1 NFV and Contention

We consider an operator managing a modern NF cluster framework (*e.g.,* E2 [41], OPNFV [10], AT&T Domain 2.0 [2]). The cluster framework manages and deploys a library of software-packaged NFs of different types such as firewalls [40], intrusion detection systems (IDS) [45], WAN optimizers [36], *etc.* The operator may purchase similar software from different vendors (*e.g.,* an IDS by Palo Alto Networks or an open-source Suricata IDS) and for each such function, they have a desired configuration and an expected traffic profile over which the function is expected to run. For the rest of this paper, when we refer to NF_i, we refer to one software package along with its expected configuration and a traffic profile.[1] We also assume that the cluster has a few *server configurations* $Arch_k$ (*e.g.,* Intel Broadwell, Skylake *etc.*) [17, 33, 38]. Since most cluster deployments standardize these configurations in practice we only have a handful of possible options for $Arch_k$.

Network Functions that are scheduled to co-run on shared hardware may experience slowdown as a result of contention for the hardware's shared resources [47–49]. Figure 1 illustrates the throughput drop that NFs (presented in Table 1) experience as a result of resource contention. The performance degradation is measured relative to the performance each NF achieves when run in isolation and can sometimes be as high as 35%.[2]

The NF performance prediction problem is defined as follows. Given as input a set of NFs $S = \{NF_i\}$, a target $NF_{target} \in S$ (*i.e.,* a NF whose performance drop we would like to estimate), a competing workload $Comp_j = \{S \setminus NF_{target}\}$ (*e.g.,* the set of NFs the target may be co-located with) and the hardware configuration $Arch_k$, our goal is to estimate the expected throughput of NF_{target} when it is run together with $Comp_j$ on a server instance of $Arch_k$.

What is special about NFV workloads? A large body of prior work in the architecture community focuses on performance prediction [16]. A natural question then is, why should NF performance prediction be any different?

General workloads can contend for a wide range of system resources such as memory capacity, storage bandwidth, CPU resources (integer/floating point compute units), or the CPU-socket interconnection network [20]. However, given the common NFV deployment practices (*e.g.,* running NFs on dedicated and isolated cores, maintaining NUMA- and interrupt-core affinity *etc.* [25, 51]), NF performance almost exclusively depends on contention in the memory subsystem. Furthermore, NFs exhibit idiosyncratic and extreme interactions with the cache hierarchy, with very little data reuse for packet data, but very high reuse for data structures (*e.g.,* rule or routing tables) which are accessed for every packet.

In short, general-purpose frameworks focus attention on modeling system components which are *not* important to NFs, while at the same time paying *little* attention to the intricacies of memory. Consequently, we observe in §2.2 and §7 that general-purpose approaches have substantial error when applied to NFV workloads.

2.2 Existing Approaches

Prior work addresses contention-related performance degradation either through performance prediction or through hardware resource isolation. Below, we discuss each approach.

Performance Prediction: Dobrescu *et al.* [24] identified that performance variability is a critical obstacle to the adoption of software-based packet processing. In this early but forward-looking work, the authors carefully identified that memory contention was the key source of slowdown in software NF implementations.

[1] We use the terms NF and NF instances interchangeably.

[2] In both cases we run the NF in its own core and dedicate the same amount of resources. We delay a more detailed description of our setup to §3.

Application	Description	Configuration	Mpps
Stateless Firewall	Stateless firewall (Click)	1K sequential rules	8.70
IP Router	IP router with RadixIPLookup (Click)	130K rules	5.75
FlowStats	Flow Statistics with AggregateIPFlows (Click)	3600 sec Flow Timeout	0.91
VPN	VPN with IPSec elements (Click)	Encryption & Authentication	0.36
Maglev LB	Maglev Load Balancer (NetBricks [42])	Default Netbricks configuration [42]	7.50
Snort	IDS in Intrusion Detection mode	Snort3.0 Community Ruleset [13]	0.45
Suricata	IDS in Intrusion Detection mode	Suricata-5.0.1 Ruleset [15]	0.97
PFSense	Open Source stateful Firewall	1K rules	0.150

Table 1: *NFs used in* SLOMO. *The reported throughput corresponds to a solo run of the NF on a Broadwell architecture with 64B packets and 400K unique flows.*

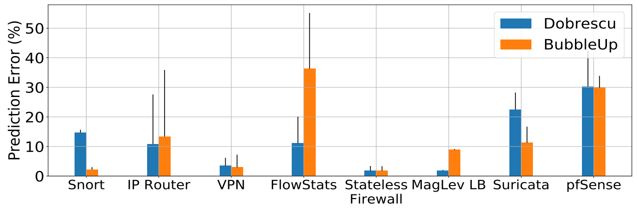

Figure 2: *Prediction Error of a general-purpose performance prediction model [39] and Dobrescu's CAR-based prediction framework [24]*

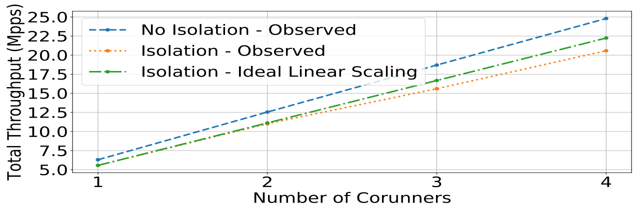

Figure 3: *LLC partitioning leads to inefficient utilization and does not eliminate slowdown*

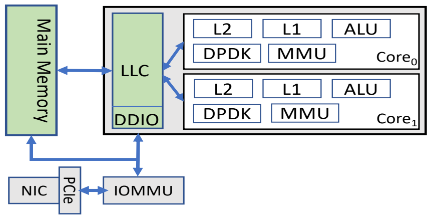

Figure 4: *System resources traversed by a network packet*

This prior work modeled memory as a *monolithic* source of contention which could be quantified through a *single* metric, using the cache access rate (CAR) of the competing NFs as the only quantifier of contention. This echoed prior work, BubbleUP, a general-purpose framework which also modeled memory contention and its relationship to performance using a single metric, using the working set size of the competing work loads instead [39]. Figure 2 illustrates the corresponding prediction error of these techniques when replicated on modern servers and NFs. In the following sections, we explain the differences between NFs that result in different levels of performance drop.

As we discuss in §3, modern memory architectures are complex and hence it is insufficient to represent this contention through a single metric. Indeed, in §3.1 we demonstrate that by combining both metrics proposed by Dobrescu and Mars – CAR and working set size – one can achieve better predictions than either metric alone. Furthermore, these two metrics alone are incomplete – in §3.2 and §3.3 we show that neither CAR nor working set size can measure the impact of contention over the DDIO cache, nor contention over bandwidth to main memory.

Performance Isolation: ResQ by Tootoonchian *et al.* [51] argues in favor of isolating shared resources to *prevent* contention-specific performance degradation. This work leverages Intel's Cache Allocation Technology (CAT) and DPDK packet buffer sizing in order to provide dedicated, non overlapping LLC partitions to the co-running NFs. However, we find that isolation is an incomplete solution to managing contention-induced slowdown because partitioning tools (*e.g.,* CAT) fail to completely isolate all sources of contention. In addition, we show that isolation can lead to inefficient resource utilization.

Figure 3 illustrates these observations through the aggregate throughput of co-running Click-based IP routers when run with and without LLC isolation. Each line indicates the aggregate observed throughput as a function of the number of co-runners. With only 4 co-running identical NF instances and equally-sized dedicated LLC partitions, we observe 24% lower aggregate throughput compared to when the NFs share the entire LLC, highlighting the loss of statistical multiplexing of the LLC and consequently reduced system efficiency. Furthermore, we observe that the total throughput can be up to 11% lower than ideal linear scaling, which we would expect to observe as a result of perfect isolation among the co-runners. This shows that even if CAT isolated the LLC perfectly, it does not isolate all possible sources of contention. In §7, we show that ResQ's approach can be *combined* with a performance prediction technique to predict contention-induced slowdown in the presence of CAT and, in that case, provides better predictability than either performance prediction or isolation on its own.

3 SOURCES OF CONTENTION

While prior work argues that the memory subsystem is characterized by one primary source of contention, our analysis shows that, operating at high rates on a modern architecture, contention is instead multifaceted. To expose the different sources of contention, we look at the life of a packet *i.e.,* the system components it traverses on its way to the NF. Figure 4 visualizes this process at a high level. During I/O, a packet arriving at the NIC will traverse the PCIe bus and will be DMA'ed either to DRAM or directly to the LLC, if the server supports DDIO [7]. The NF will read the incoming packet from the memory hierarchy and the NF will apply its packet processing logic to it. Depending on the nature of the NF, data from auxiliary data structures stored in the memory hierarchy might be requested. Ultimately, the packet will be written back to the NIC following the reverse process.

Along these datapaths, we identify three independent sources of contention; (1) contention in the LLC (*i.e.,* contention that compromises fast access to auxiliary data structures containing necessary

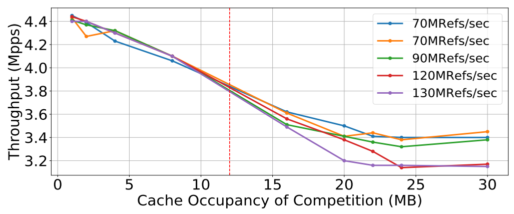

Figure 5: *Target NF throughput as a function of the LLC occupancy of the competing NFs.*

data for packet processing), (2) contention for DDIO resources during packet I/O (*i.e.,* contention that slows down packets on the direct path between the NICs and the LLC), and (3) contention for main memory bandwidth (*i.e.,* contention that increases the latency to service a LLC miss from main memory). In what follows, we present the underlying mechanisms of each contention source and argue why using multiple metrics can accurately quantify contention.

Experimental setup: We experiment with two *x*86 architectures: (1) Intel Xeon E5-2620 v4 (Broadwell) and (2) Intel Xeon Silver 4110 (Skylake). Both server architectures are equipped with two physical NICs (XL710-40Gbps NIC for Broadwell, Mellanox MT27700 - 100Gbps for Skylake) that were partitioned among co-running NFs using SR-IOV [14]. We relied on DPDK for packet acceleration for all NFs except PFSense which does not support packet acceleration.

We ran our evaluation on a range of NFs shown in Table 1 drawn both from research prototypes (*e.g.,* Click [32], NetBricks [42]) and popular software (*e.g.,* Snort [13], Suricata [15], PFSense [19]). These NFs span a broad spectrum in terms of complexity of packet processing, ranging from simple NFs (*e.g.,* stateless Firewall) to more complex ones (*e.g.,* Snort). To the extent it was possible, we configured NFs with public rulesets (100s-1000s of rules) [13, 15].

In this section, we measure slowdown of real NFs under competition with a few carefully crafted artificial competing NFs, designed to illuminate each form of contention independently. Note that the point of these examples is not to demonstrate that the causes of contention *always* occur independently: in practice they may have correlated causes – *e.g.,* an increase in overall request rates increases both cache access rate and memory bandwidth. Nonetheless, the fact that each bottleneck *can* be contended independently motivates the need for models that capture all causes of bottlenecks.

3.1 Contention in the last level cache

Prior approaches to modelling memory contention focus solely on contention in the LLC, which increases packet-processing time due to evictions of NF data structures to main memory. Both Dobrescu *et al.* and BubbleUp focus on this category of contention, but model it using different metrics. Dobrescu *et al.* measure the *rate* at which a competing NF accesses the cache (CAR), but BubbleUp measures the working set size or cache occupancy of the competing NF. We find that on our servers, measuring *both* provides the best insight into slowdown.

Observation 1: ***LLC contention depends both on cache occupancy and the rate at which the competing NFs accesses the last level cache.***

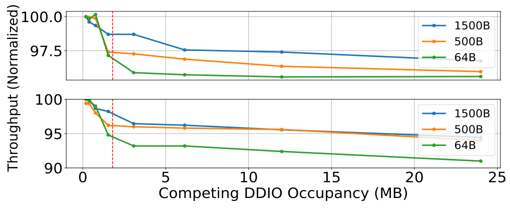

Figure 6: *Performance drop as a function of DDIO space occupied by competition*

We run a Click-based IP Router (see Table 1) against one Click-based synthetic competitor that, for every packet it receives, performs a configurable number of reads to a data structure of configurable size in the LLC. The IP-router in isolation occupies approximately 8MB of cache space. To guarantee isolation from other sources of contention, we ensure (1) that the allocated packet buffers don't introduce DDIO contention (see § 3.2) and (2) that the competing NFs use separate memory channels.

Figure 5 visualizes the experiment results for these two configuration parameters of the synthetic workload. When cache occupancy is low – less than the red line marking exhaustion of the available LLC space – occupancy is the best predictor of performance. After the cache is saturated, CAR becomes the dominant factor although we still observe a downward trend correlated with occupancy.[3]

3.2 Contention during packet I/O

Modern x86 Intel architectures offer Data Direct I/O (DDIO), an optimization to DMA that copies packets directly between the NIC and a dedicated slice of the LLC [7, 23, 34]. DDIO essentially partitions the LLC into a primary cache and an I/O cache; at startup every NF allocates a fixed number of buffers in the I/O cache to store the packets for that NF. Contention can occur when the total number of packets concurrently in the system exceeds the amount of space in the I/O cache – even though the remainder of the LLC remains relatively underloaded.[4] Consequently, we need to model contention in the slice of the cache dedicated to DDIO separately from modeling the remainder of the LLC.

Observation 2: ***DDIO contention depends both on the utilization of the DDIO space by the competing NFs and the rate at which they access it.***

We find that while not all target NFs are equally sensitive to DDIO-contention, all NFs (irrespective of packet size or number of allocated buffers) suffer some level of DDIO-related slowdown. In Figure 6, we illustrate an instance of a Stateless Firewall that competes with a simple Click-based, L2-forwarding NF. In the top graph, the Firewall has 524KB buffers allocated and in the lower graph it has 3MB of buffers allocated. The contending L2 forwarder is configured to ensure minimal LLC contention outside the DDIO slice, and process traffic at rates up to 100Gbps/NF. In the L2 forwarder NF, we vary (i) the size of the buffers allocated (occupancy)

[3]Dobrescu's analysis assumed that the NF is *highly contended* and hence the occupancy of the competing NFs far exceeded the size of the LLC – to the right of the red line.
[4]ResQ addresses this problem just like contention in the LLC: by partitioning and hence isolating sub-components of the I/O cache. They achieve this through careful allocation of buffers sized not to exceed the overall cache space. This worked well at 10Gbps speeds, but pushing towards 100Gbps, the number of concurrent packets can exceed space in the I/O cache leading to packet loss and performance slowdown.

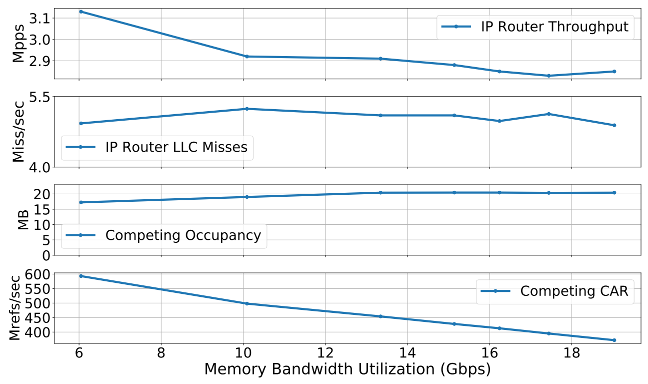

Figure 7: *Performance vs utilized memory bandwidth*

and (ii) the size of the packets sent to the L2 forward, and hence the rate of packet arrivals and memory accesses. Just as in Figure 5 which describes LLC contention, we see that slowdown in the I/O slice of the cache is a function of *both* occupancy and access rate. As a result, for accurate predictions we need to measure access rate and occupancy of the I/O slice of the cache in addition to access rate and occupancy of the overall LLC.[5]

3.3 Contention for memory bandwidth

Until now, we have focused on how NFs suffer slowdown due to data structures being evicted from the LLC or packets being evicted from the I/O slice of the cache. We now discuss how the *cost* of eviction can vary as competitors increase their bandwidth utilization between the cache and main memory.

Observation 3: ***Main memory latency depends on the aggregate memory bandwidth consumption of the competing NFs.***

We observe that a target NF (Click IP router) can experience up to 18% of throughput drop as a result of main memory bandwidth contention. To ensure that contention is limited in main memory bandwidth utilization we use Intel's Cache Allocation Technology (CAT) [6] to partition the last level cache of a Broadwell server in two segments, one for the target NF (2MB) and the second (18MB) for the competitors. The competitors consisted of aggressive Click-based competitors that have a high cache miss rate and hence generate large amounts of memory traffic. The target NF executes a fixed number of accesses.

Figure 7 shows the performance of the target NF as a function of the total memory bandwidth utilization and observe the clear correlation between the two metrics. Additionally, we confirm that first, slowdown is uncorrelated with contention in the LLC as the target NF LLC miss rate stays relatively stable and second, that LLC isolation is not sufficient for eliminating contention.

4 SLOMO **OVERVIEW**

SLOMO conceptually follows a blueprint for performance prediction based on *contentiousness* and *sensitivity* [20, 21, 38, 39]. Specifically, contentiousness measures the pressure a NF places on shared hardware; sensitivity models how susceptible a NF is to performance degradation due to the competitors' aggregate contentiousness.

Building on these concepts, we realize a practical workflow for SLOMO that conceptually consists of two logical parts: (1) an offline component that is responsible for characterizing contentiousness and modeling sensitivity of the available NF instances; and (2) a prediction component responsible for making performance predictions given a target NF instance and a mix of real competitors.

4.1 Offline Profiling

Given a set of NFs $S = \{NF_i \ldots\}$ and a server architecture $Arch_k$, SLOMO first runs a constant number of offline profiling operations to characterize (1) the sensitivity and (2) the contentiousness of the different $(NF_i,\ Arch_k)$ tuples. To do so, the operator runs each NF_i on the server with multiple configurations of a *tunable synthetic workload*, configured to apply different amounts of pressure to the system resources and, thus, contend with NF_i. For each synthetic configuration x, x's contentiousness is represented by a vector V_x. In §5, we discuss our choice of synthetic workload and in §5.1, we discuss our methodology for choosing the contentiousness metrics in V.

To *profile for sensitivity*, we measure, on every architecture, NF_i's performance P_i^x in response to each synthetic contentiousness vector V_x. The dataset consisting of all pairs $\{(V_x,\ P_i^x), \ldots\}$ is used to train a *sensitivity model* $M_i : V \longrightarrow P$ to predict NF_i's performance in response to any real contentiousness vector. We discuss SLOMO's sensitivity models in §5.2.

To *profile for contentiousness*, the operator collects a set of vectors $\{V_i^x\}$, where each vector characterizes NF_i's contentiousness associated with every synthetic run x. Said differently, V_i^x measures the pressure that NF_i applies on the shared resources in the presence of x, as if NF_i were an additional competitor. In §5.3, we describe how we use the individual contentiousness vectors to compose the contentiousness of any mix of real competitors.

These profiling datasets are specific to a particular NF type, configuration, traffic workload and server architecture. In practice, a typical cluster may use only one or a small number of server architectures which do not change frequently. However, it is possible that, after deployment, an NF's ruleset or its traffic workload might change. In §6 we discuss how SLOMO can adapt its existing models M_i to extrapolate NF_i's performance as a result of changing operating conditions.

Even though prior work [24, 39] followed a similar workflow, they used only linear models for sensitivity with one solitary metric for contentiousness. Given our analysis in §3, we take a first-principles approach to learn *multivariable* models and metrics. In §5, our data-driven approach to contentiousness characterization and sensitivity modeling.

4.2 Online Predictions

At run time, the operator uses the pre-computed V_i's and M_i's for predictions. In the most basic scenario, the operator has two NFs, NF_A and NF_B which they want to run side-by-side on the

[5]While DDIO space is fixed, isolation between DDIO and the rest of the LLC is not perfect *i.e.,* packet buffers can still evict LLC data and vice versa. This is orthogonal to our claim that the two sources of contention can be independent of each other.

same server. To predict NF_A's throughput while running alongside NF_B, the operator simply takes NF_B's contentiousness vector V_B and plugs it into NF_A's sensitivity model M_A to produce the performance slowdown P_A^B.

Nonetheless, some predictions are more challenging. Consider an operator now with three NFs: NF_A, NF_B, and NF_C. The operator now wants to run all *three* NFs side by side, and once again wants to predict NF_A's throughput under this deployment. The problem here is that, although the operator has pre-computed V_B and V_C, the operator does not know $V_{B,C}$ – the contentiousness upon NF_A when running alongside *both* NF_B and NF_C.

SLOMO provides a function for *composition* $CF : V_B, V_C \rightarrow V_{B,C}$, this allows the operator to compute $V_{B,C}$ offline based on the pre-computed contentiousness vectors of each NF. After composing V_B and V_C, the operator can use this for prediction just as in the basic scenario. Dobrescu's approach that used competing CAR as the sole metric of contentiousness, implemented composition by summing together each competitor's CAR values when run solo on the hardware. In §5.3, we discuss why measuring contentiousness during a solo run introduces prediction inaccuracies and discuss SLOMO's implementation of CF.

5 SLOMO IN DEPTH

Having laid out the three key components to SLOMO (contentiousness characterization, sensitivity modeling, and contentiousness composition), we now describe how we design each of these components, taking a data-driven approach. Seen in this light, modeling sensitivity is a *model fitting* process (§5.2). Similarly, choosing contentiousness metrics is as *feature selection* process whose goal is to identify metrics that quantify the competition's pressure on the shared hardware and have strong predictive power in the context of a sensitivity model (§5.1). Finally, composition is a simple regression modelling problem (§5.3).

Candidate contentiousness metrics: We choose our candidate contentiousness metrics to be those exposed by the Intel PCM framework, a performance monitoring API enabling real-time collection of architecture-specific *resource utilization metrics* [8]. The resulting PCM vector contains an extensive pool of metrics (*e.g.,* main memory traffic, the LLC hit rate *etc.*) that characterize resource usage at core-, CPU-socket- and system-level granularities. That said, a natural limitation of SLOMO is that it is limited by the pool of metrics exposed by PCM. For instance, PCM does not provide visibility into the internals of a NIC. Thus, any congestion for NIC resources (*e.g.,* increased NIC queue occupancy from incoming traffic) will not be taken into consideration.

Synthetic competition: Our profiling methodology assumes the presence of a representative training dataset that adequately samples the large space of contentiousness vectors $\{V\}$. To produce this dataset, and following general guidelines discussed in prior work [39], we exercise the effects of contention on each NF with a *synthetic workload of tunable intensity* that samples the space of possible contentiousness values that a NF could generate. To that end, we designed an artificial Click-based NF that covers the contentiousness space by applying incremental pressure (1) to the I/O datapath through the number of allocated packet buffers and

Metric	Definition	CC
IPC	Instructions/Cycle	0.78
INST	Instructions Retired	0.78
L3MISS	LLC Misses	0.893
L3MPI	LLC Misses/Instruction	0.88
L3HIT	LLC Hit Rate	0.8
L3OCC	LLC Occupancy	0.82
LMB	Local NUMA Bandwidth	0.88
L2HIT	L2 Cache Hit Rate	0.68
L2MPI	L2 Misses/Instruction	0.75
L2MISS	L2 Misses ≡ CAR	0.72
READ	Memory read traffic	0.89
WRITE	Memory write traffic	0.68
LLCMISSLAT	LLC Read Miss latency	0.9
RMB	Remote NUMA Bandwidth	0.24
QPI	QPI utilization	0.18
FREQ	CPU frequency	0.06

Table 2: *PCM metrics with high correlation with performance. The CC column shows the mean correlation coefficient across experiments.*

(2) to the packet-processing datapath by performing configurable numbers of memory operations (*i.e.,* random Reads and/or Writes) to a data structure of configurable size stored in the LLC. SLOMO exercises these configurations for various traffic patterns (*i.e.,* rate, packet sizes and flow counts) and number of co-running instances of the synthetic NF. SLOMO profiles each NF with more than 1000 different configurations of the synthetic workload and the resulting dataset consists of the following sets of measurements: (1) PCM values when the synthetic workload and NF under test run solo; (2) PCM values when both the synthetic workload co-runs with the NF under test; and (3) Performance of target NF when running with the synthetic competitor. In the rest of the section, we show how we use each one of these datasets. Note that the synthetic competing workloads are only used during offline profiling and not in runtime.

Assumptions on NF Deployment: SLOMO follows best practices established by prior work that optimize performance stability for software packet processors [4, 11, 12]. Specifically, we run NFs on dedicated and isolated cores that use local memory and NICs (NUMA affinity), we ensure interrupt-core affinity is maintained in cases where packet acceleration is not used, we disable power-saving features of the CPU (*e.g.,* idle states, frequency scaling) and disable transparent huge pages.

5.1 Contentiousness Metrics Selection

A simple way to select contentiousness metrics would be to use the entire PCM vector (~600 metrics). Doing so, however, is "noisy" as it includes metrics that are unrelated to plausible sources of contention (*e.g.,* metrics of unused CPU sockets) and, thus, hurts model accuracy and interpretability [29].

Ideally, we need to identify a subset of *expressive* PCM metrics *i.e.,* metrics with strong predictive power in the context of a sensitivity model. Given that we don't have a model at this point, we opt for *model-free* techniques to enable this process. Among the many available techniques *e.g.,* the Pearson correlation coefficient [18],

Metric	Definition	CC
WRITE	Memory Write traffic	0.90
READ	Memory Read traffic	0.88
IPC	Instructions/Cycle	0.88
L3OCC	LLC Occupancy	0.80
L2MPI	LLC Accesses/Instruction	0.76
L3HIT	LLC Hit Rate	0.40
LMB	Local NUMA bandwidth	0.20

Table 3: *DDIO Contentiousness*

Metric	Definition	CC
L3MISS	LLC Miss Rate	0.98
L3OCC	LLC Occupancy	0.87
L3HIT	LLC Hit Rate	0.79
LMB	Local NUMA bandwidth	0.76
L2MISS	CAR	0.76
L2HIT	L2 Hit Rate	0.36
RMB	Remote NUMA bandwidth	0.13

Table 4: *LLC Contentiousness*

Metric	Definition	CC
READ	Memory Read traffic	0.81
WRITE	Memory Write traffic	0.81
LMB	Local NUMA bandwidth	0.80
L3MISS	LLC Miss Rate	0.79
L3OCC	LLC Occupancy	0.77
L3HIT	LLC Hit Rate	0.67
RMB	Remote NUMA bandwidth	0.15

Table 5: *Memory Contentiousness*

information gain [52] or PCA, we use Pearson's correlation coefficient to analyze the statistical dependency between the various PCM metrics and the observed performance of each target NF.

Observation 4: ***PCM metrics at CPU-socket- and System- level granularities adequately capture aggregate contentiousness.***

Table 2 lists the metrics that consistently exhibit high correlation to performance across all experiments. Our first key observation is that these metrics correspond to CPU-socket- or System-level granularity (instead of core-level metrics). That is because performance degradation is the result of the competition's *aggregate* contentiousness which is naturally captured at these granularities.[6]

We also observe that as there is no clear winner among the chosen metrics in terms of the magnitude of correlation coefficient, indicating that contentiousness entails multiple variable. This is in agreement with §3 that argued for different sets of metrics for different sources of contention. To confirm this hypothesis, we revisit the experiments of §3 and identify the best metrics for each type of contention.

Observation 5: ***Different sources of contention are best captured by different metrics. As NFs can depend on multiple contention sources, contentiousness should be quantified with an ensemble of metrics.***

Tables 3, 4, and 5 show the highest ranked metrics for each source of contention. We see that:

1. **DDIO contention** We find that DDIO contention is best quantified through memory bandwidth utilization metrics. As DDIO operations are managed by the socket's DMA engine, packet buffer evictions are not captured by LLC utilization metrics and as such they prove to be uncorrelated to this source of contention.
2. **LLC contention** In contrast to DDIO contention, LLC contention is best captured through LLC-related metrics. The LLC occupancy of the competition, its cache access rate, LLC miss rate and LLC misses per instruction exhibit the highest correlation with performance.
3. **Memory Bandwidth** Finally, with respect to memory bandwidth contention we find that the key contentiousness metrics are, unsurprisingly, memory bandwidth utilization metrics (*e.g.,* Read/Write bandwidth) as well as the Local NUMA node bandwidth utilization.

[6]In a setup that only uses one CPU socket, the correlation values at socket and system level granularities coincide.

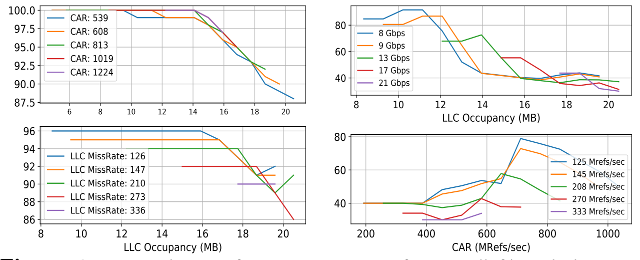

Figure 8: *2D rendering of sensitivity curves for VPN (left) and FlowStats (right). Y axis is normalized performance.*

In all, we conclude that out of the initial pool of ∼ 600 PCM metrics, the ∼ 15 aggregate PCM metrics can capture the contentiousness from the key chokepoints we previously identified. These metrics will also serve as the input to a target's sensitivity model to predict performance.

5.2 Modeling Sensitivity

Sensitivity captures the relationship between the contentiousness of the competition and the performance of the target NF. Modeling sensitivity, therefore, can be viewed as a *regression problem* as its input (contentiousness of the competition) and output (target NF performance) are both continuous variables.

Our first observation is that sensitivity modeling needs to happen on a per NF-basis instead of having a global sensitivity model. Indeed, with different NFs responding differently to the various sources of contention, the importance of each metric in the context of a sensitivity model is NF specific.

We train sensitivity models using synthetic, NF-specific contentiousness observations, created as described in §5.1. At run time, we replace the synthetic inputs with the aggregate contentiousness of the real competitors. For testing, we generate for each NF and architecture a dataset of real experiments where each target NF is co-run with various combinations of NFs drawn from our pool of NF instances with replacement. Each target NF is run 5 times on each architecture with 150 different configurations of competitors.

Observation 6: ***Sensitivity can be a complex function that cannot be captured by simple regression models.***

We find that sensitivity is a non-linear and non-continuous function of its multivariate input and as such, it cannot be accurately modeled with techniques such as regressions (linear/polynomial *etc.*), decision trees, simple neural nets *etc.* Nonetheless, a common pattern that we detect across sensitivity functions are *phase transitions i.e.,* sharp changes in the properties of sensitivity as a result of increasing contentiousness. For instance, for a fixed value of competing CAR, low (competing) LLC occupancy values naturally trigger few evictions of target NF data. As a result, the target's performance decreases slowly as a function of the competitor's increasing occupancy. However, as soon as the aggregate LLC occupancy of the target NF and its competitors exceed a critical threshold *i.e.,* the available cache space, the probability of target NF evictions sharply increases and with that the rate of performance degradation for the target NF (Figure 8).

Observation 7: ***Sensitivity can be modeled as a piecewise function of its inputs. Gradient Boosting Regression, an ensemble modeling technique enables accurate sensitivity modeling.***

This observation introduces our key insight behind sensitivity modeling in SLOMO. By viewing sensitivity as a *piecewise* function of contentiousness, we can model the different sub-spaces of sensitivity separately and then combine the resulting models into a larger, comprehensive one. Fortunately, the machine learning literature offers a class of methods, namely *ensemble methods*, that are designed for that exact purpose by combining, into a comprehensive robust model, many smaller models that focus on specific areas of the sensitivity space [22, 54].

Ensemble methods build a family of base estimators, with each estimator focusing on a subsection of the function to be modeled. These weaker base estimators are then aggregated to obtain a strong learner that performs better. Among the several variations of existing ensemble techniques (*e.g.,* bagging, boosting, stacking), we choose Gradient Boosting Regression (GBR) [27].[7] In §7 we quantitatively show that its predictions outperforms a large collection of other well-known modeling tools.

In summary, our modeling technique captures the intricacies of a complex sensitivity function and improves on prior work which uses simpler linear models.

5.3 Measuring Contentiousness

In §4.2, we introduced the example of three co-running NFs, NF_A, NF_B, and NF_C where we wanted to predict the performance of NF_A running alongside NF_B and NF_C. We now discuss (a) how NF_B and NF_C's contentiousness vectors, (V_B, V_C) are measured in detail, and (b) how to compute their combined contentiousness, $V_{B,C}$ from measurements, extracted during offline profiling.

Measuring NF_i's contentiousness: A starting point might be to measure the PCM counters for a given NF_i while this NF_i runs alone on the server. However, this is inaccurate because an NF's resource utilization (and hence its vector V_i) changes in the presence of competition (*e.g.,* by as much as 5× for metrics like LLC occupancy). That is, in the case of NF_B, we need to estimate V_B^C *i.e.,* its contentiousness *while competing with* NF_C. We make two key choices to estimate this vector.

(1) We measure contentiousness for NF_i while it is running against the various synthetic competitors NF_x – many times over. Each time NF_i is subjected to a unique V_x, we measure NF_i's contentiousness resulting in a set of potential V_i's

[7] GBR is a technique that produces strong predictors with lower bias, by *sequentially* fitting weak learners in an adaptive way, giving more importance to observations in the dataset that were badly handled by previous models in the sequence.

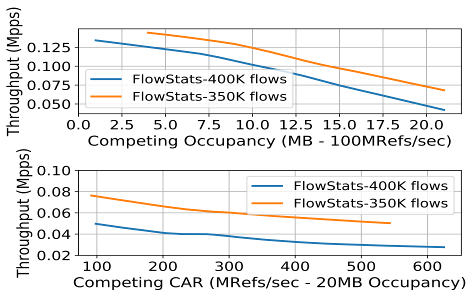

Figure 9: *FlowStats's sensitivity changes with the number of concurrent flows*

(2) We then group the V_i^x's based on how many co-runners (utilized cores) our synthetic competitor was simulating. For our example above, we group all V_B^x where NF_B ran against one simulated co-runner; we then take the *average* of all such vectors. This value is our estimate of V_B^C.

Hence, in reality, during the profiling phase we do not generate one contentiousness vector for each NF. Instead, we measure one contentiousness vector per configuration of the synthetic competitor and during runtime, we select the appropriate subset of these measurements based on the number of competitors.

Composition: Having estimated the individual contentiousness of NF_B and NF_C in the presence of two competitors, the next step is to compose them in order to get an estimate of $V_{B,C}$. Composition is possible because the aggregate contentiousness metrics we wish to estimate are by definition the sum or average of the constituent per-core metrics (*e.g.,* the CAR of a CPU-socket is the sum of each core's CAR within the socket) [1]. Thus, having estimated the individual contentiousness of each competitor, composing the aggregate boils down to applying the appropriate linear operator for each metric. §7 evaluates the prediction accuracy of using this compositional model versus observed actual aggregate contentiousness across experiments and shows that our composition mechanism adds less than 5% additional prediction error.

6 EXTRAPOLATING SENSITIVITY

So far, we have considered a rather strict definition of NF_i that includes its *type* (*e.g.,* firewall, IDS *etc.*), *configuration* and *traffic profile*. While this definition has simplified the exploration of the sensitivity/contentiousness design space, NF_i might experience modifications during its lifecycle (*e.g.,* migrations across servers, changes in configuration or changes in traffic profile) which will effectively lead to the creation of a new, unknown, NF instance, NF_i'. A natural concern for operators is whether SLOMO's design is general enough to enable them to extrapolate quick-yet-accurate performance predictions for NF_i', without triggering a slow offline profiling operation immediately after NF_i is modified. In other words, having the ability to extrapolate NF_i''s performance predictions by leveraging existing profiles of NF_i can save the operator time and thus enhance SLOMO's usability. In this section, we use a running example to present empirical heuristics to extract NF_i's sensitivity as well as their limitations.

We consider the example of FlowStats, an NF that is heavily sensitive to memory contention and explore its changes in sensitivity as a function of the number of *unique traffic flows* it receives (all other parameters of configuration, traffic and competition remain fixed). In our experiment, we first reduce the number of traffic flows by 50K from 400K to 350K (~ 12%) and observe that as that number decreases (1) the NF becomes less sensitive to the same amount of contention and (2) its solo throughput increases. By the same token, when we increase the flow count by the same amount (from 400K to 450K), we observe the opposite behavior.

To explain this behavior, we look at the resulting change in the NF's shared resource utilization. In the case of FlowStats, which keeps per-flow state, the reduction in concurrent flows leads to a smaller hash table in memory and hence a reduced probability of contention over the LLC. Figure 9 visualizes the effects of this change across two key dimensions of FlowStats' sensitivity, competing LLC occupancy and competing CAR.

We observed similar effects in other NFs. For instance, changes in the size of the ruleset of an IP Router (±20%) also resulted in a change the LLC utilization of the router and corresponding change in the NF's sensitivity curve.[8] This leads to our key observation:

Observation 8: ***Changes in NF_i's traffic profile of configuration translate in changes in NF_i's reliance on shared memory resources and thus to its sensitivity.***

Recall from §5.2 that we use an *ensemble of linear models* as our prediction function. Our insight for extrapolation, therefore, is to modify each linear model to adjust to changes in the LLC utilization of the target – whether due to traffic or ruleset variation – as follows.

Slope with respect to CAR: As the LLC occupancy of the target NF goes down, there is less opportunity for the competition and the target to contend over the same cache line. Hence, we expect that as LLC occupancy of the target goes down, the same value of CAR for the competition would result in fewer evictions. Assuming that our initial model of competitor's CAR and target's throughput has a slope α, we model the new slope α' as the ratio of the new cache occupancy of the target to the original cache occupancy of the target: $\alpha' = \frac{OCC_{\text{target}}^{\text{new}}}{OCC_{\text{target}}^{\text{old}}}\alpha$.[9]

X-shift with respect to Occupancy: The slowdown for the target NF with regard to cache occupancy is really a function of the *total* cache occupancy. In particular, we consistently see an inflection point in the sensitivity curve when the sum of the target and competition's cache occupancies surpass the total available LLC space. If the target's occupancy reduces, we hence expect to see this inflection point at a *higher* level of occupancy by the competition. Hence, we shift the sensitivity function along the x-axis (occupancy of competition) relative to the difference in the occupancy of the target $\delta = OCC_{\text{target}}^{\text{new}} - OCC_{\text{target}}^{\text{old}}$.

Y-shift with respect to solo performance: Finally, when the occupancy of the target changes, the baseline, solo throughput of the NF may shift. To account for this, we shift along the y-axis (performance) by β, for $\beta = \text{SoloPerf}_{\text{target}}^{\text{new}} - \text{SoloPerf}_{\text{target}}^{\text{old}}$.

[8]Throughout these experiments, we ensured that incoming traffic would touch the entirety of the ruleset and thus, the addition of rules would not be redundant

[9]In reality, the relationship between CAR and probability of evicting target NF data follows a geometric distribution and should be modeled as such [24].

Scope of extrapolation: So far, our discussion about extrapolation heuristics is based upon the assumption that NF_i's change is "small" and thus there is overlap between the sensitivity profiles of NF_i and $NF_{i'}$. We find this to be a reasonable assumption as it allows us to "transfer" the knowledge of NF_i's sensitivity profile to NF_i' without a significant loss of accuracy. On the other hand, in cases where configurations or traffic profiles differ significantly (*e.g.,* a firewall with 1 vs. 10K rules), there is little to no overlap between the respective sensitivity profiles and, as a result, accurately extrapolating performance drop is not possible.

In §7, we demonstrate the promise of our simple extrapolation scheme. We show that despite the inherent complexity of the sensitivity function, our approach on average adds 3 percentage units to the error, and significantly outperforms the accuracy of predictions where extrapolation was not used.

7 EVALUATION

In this section we evaluate our approach and show that:

1. SLOMO is *accurate*, with a mean prediction error of 5.4%, reducing Dobrescu's 12.72% error by 58% and BubbleUp's 15.2% average error by 64% (7.1).
2. SLOMO's predictions are *robust* across operating conditions (7.1).
3. The design decisions behind each of SLOMO's components contribute to improved accuracy (7.2).
4. SLOMO is *efficient* and enables smart scheduling decisions in an NFV cluster (7.3).
5. SLOMO is *extensible*, allowing the accurate extrapolation of the sensitivity function of new NF instances to account for changes in an NF's traffic profile or configuration (7.4).

For each experiment we choose the target NF (*i.e.,* type, configuration and traffic profile), the number and type of (real) contenders and the architecture the NFs will run on. For NF profiling and testing we use the methodology described in §5. Unless mentioned otherwise, we estimate performance using the composed contentiousness of the competition (*i.e.,* we always assume a cold start scenario). SLOMO fully profiles each NF type on two architectures (Broadwell and Skylake) and for 6 different configurations of traffic profile (*i.e.,* 64/1500B and 40K/400K/4M traffic flows, uniformly distributed across the space of possible destination IPs). The NFs processed the maximum traffic until we saturated the corresponding core. For the Click-based IP Router and Stateless Firewall, we also experimented with 2 ruleset sizes (small/large).

7.1 Accuracy

What is the end-to-end prediction accuracy of SLOMO? We compare SLOMO's absolute mean prediction error against (1) Dobrescu's CAR-based prediction model [24] and (2) the more general-purpose BubbleUp prediction framework by Mars *et al.* [39]. Overall, SLOMO reduces the absolute mean prediction error with respect to Dobrescu's model and BubbleUp by 57% and 64% respectively.

Figure 10 shows SLOMO's end-to-end average absolute prediction error for all experiments across NF types. We make the following observations. First, SLOMO's both average prediction error and error variance are across the board lower than those of prior work.

Cases where the prediction errors between SLOMO and prior work are similar (*e.g.,* VPN, stateless firewall, Maglev) correspond to NFs that do not show high dependence on the memory subsystem, either because they are CPU bound (*e.g.,* VPN) or because their rulesets impose little memory overhead (*e.g.,* stateless firewall). For instance, in the case of a stateless firewall that co-ran with 7 competitors, its observed throughput was 4.12Mpps, SLOMO's was 4.10Mpps, Dobrescu's 4.02Mpps and that of BubbleUP 4.02Mpps. However, for IP Router, a NF that depends heavily on the memory subsystem, its observed throughput was 3.46Mpps, SLOMO's prediction was 3.30Mpps, Dobrescu's 2.79Mpps and that of BubbleUP 3.00Mpps. We note, however, that FlowStats and IP Router exhibit higher error variance. We revisit this observation later in this subsection.

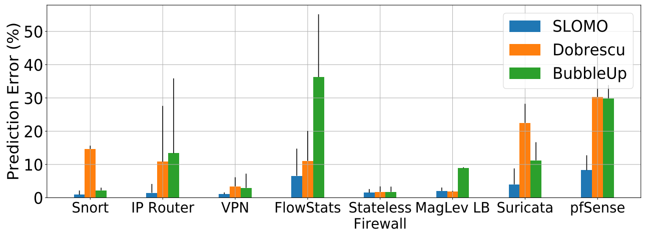

Figure 10: *End to End prediction error by NF type*

Does SLOMO **under/overestimate performance?** To see if SLOMO over- (positive error) or under-predicts (negative error) performance, we show the signed prediction error across NF types in Figure 11. We observe that there is no clear trend in favor of either sign, indicating SLOMO's prediction error is unbiased. On the other hand, we observe that predictions made with the CAR-based model tend to overestimate the amount of performance drop the target NF experiences.

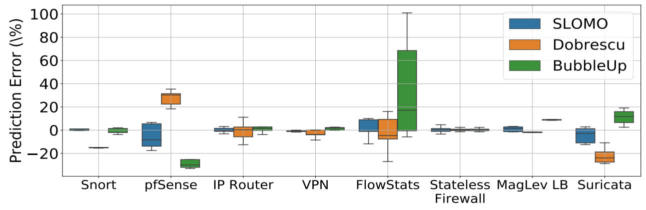

Figure 11: *Signed prediction error for IP Router, FlowStats, VPN.*

How robust are SLOMO**'s predictions?** To evaluate SLOMO's robustness, we look at how both its signed and/or unsigned prediction error changes as a function of key data dimensions as described in the following points:

1. The absolute error follows an increasing trend as a function of the number of competing NFs (Figure 12). We attribute this to an additive, composition-related error factor that is introduced for every additional competitor.
2. SLOMO occasionally over-predicts the performance drop in cases where memory sensitive NFs (*e.g.,* IP Router, FlowStats) are co-run with up to two contenders. We attribute this error to a gap in our training dataset that did not sufficiently cover areas of low contentiousness. We still observe, however, that this error is substantially lower than that of the CAR-based model (Figure 13).
3. SLOMO's absolute error follows an increasing trend as a function of the unique flow count of the traffic received by the target NF (Figure 14). Indeed, a higher number of unique flows will result in higher utilization of the target NF's auxiliary structures, which also explains the error variance of FlowStats and IPRouter. These NFs are not sensitive to contention when the number of traffic flows is low, hence their prediction error is very small. As the flow count increases, performance degradation increases and so does the associated error.
4. SLOMO's average prediction error does not change significantly as a function of packet size processed by the target NF (Figure 14). However, we observe a slightly higher variance in SLOMO's prediction errors which we trace back to a number of experiments where SLOMO overestimated the performance drop of NFs that due to their lower packet rate (as a result of the higher packet size) did not experience any performance drop.
5. Finally, from Figure 14 we observe that SLOMO's error, also appears largely independent of the competing traffic rate (load offered to the system). This highlights that SLOMO's design can adapt to different types of competitors that apply varying amounts of contention to the shared resources.

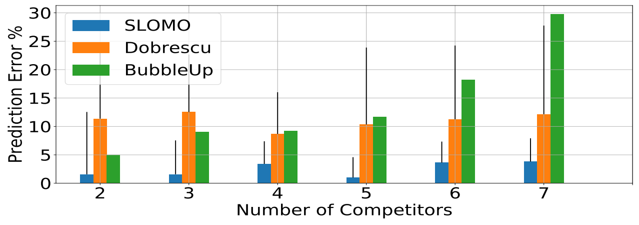

Figure 12: *End to End prediction error by number of competing NFs*

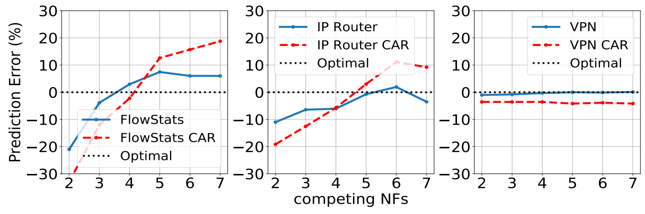

Figure 13: *End to End prediction error by NF type*

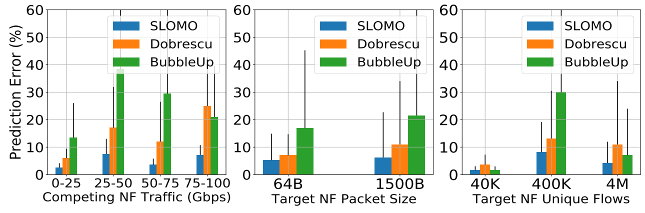

Figure 14: *Prediction error as a function of traffic profile*

7.2 Factor Analysis

We now examine design decisions behind SLOMO's components.

How does our choice of contentiousness metrics impact accuracy? We first experimentally estimate the number of metrics needed to accurately capture contentiousness. We train each model with one metric at a time and measure the prediction error of the resulting model on our testing dataset. The leftmost column of Table 6 (*Top 1*) shows the absolute error for the best 1-metric model for each NF. We observe that the error is consistently larger than SLOMO's, which validates our choice of using multiple metrics. The second column further corroborates this observation by showing the drop in prediction error that is observed with the best (NF-specific) 3-metric sensitivity model.

We then look at the specific metrics that need to be considered in the ensemble to improve accuracy. Recall that in §5 we found that different sources of contentiousness are captured by different metrics. Given that, we measure the prediction error of a sensitivity model fitted (1) with LLC-specific metrics and (2) with main-memory metrics only (columns 3 and 4). We observe that while both subsets of metrics improve the accuracy against a single metric model, error is further reduced when they are combined in one model (final column). This observation shows that our ensemble of contentiousness metrics adequately captures the various sources of contention in the memory subsystem.

NF	Top 1	Top 3	LLC	Mem	SLOMO
FlowStats (B)	17.9%	8.7%	5.5%	7.1%	7.5%
IP Router (B)	8.6%	3.1%	6.2%	4%	3.2%
Maglev LB (B)	3.2%	1.2%	1.5%	2%	1.2%
Suricata (B)	23.7%	12.7%	15.5%	13.2%	10.4%

Table 6: *Absolute prediction error for different sets of contentiousness metrics*

How do key contentiousness metrics change across NFs/architectures? Table 7 contains the top 3 metrics for a collection of NF instances. For the same NF,the corresponding set of metrics changes across architectures (**B**roadwell, **S**kylake). We notice, however, that in the Skylake server, Memory Writes are a metric that appears commonly in the top-3 of contention-prone NFs. In our effort to understand the explanation behind this observation, we found that Skylake and Broadwell architectures differ significantly in the organization of their memory hierarchy. Specifically, Skylake servers have a substantially smaller LLC (11MB vs. 20MB) and a non-inclusive[10] write-back cache policy that for NFV workloads results in producing large amounts of writes to main memory [9].

NF	Metric 1	Metric 2	Metric 3
FlowStats (B)	LMB	LLC MISS	LLC OCC
FlowStats (S)	LLC OCC	CAR	MEM WRITE
IP Router (B)	LLC OCC	MEM READ	CAR
IP Router (S)	MEM WRITE	LLC HIT	LLC OCC
Snort (B)	MEM WRITE	CAR	MEM READ
Snort (S)	MEM WRITE	CAR	LLC MISS

Table 7: *Top 3 metrics per NF*

How does Gradient Boosting Regression compare to other candidate techniques for sensitivity modeling? Figure 15 evaluates our choice to capture the complex piece-wise nature of sensitivity with Gradient Boosting Regression. We compare GBR's prediction error on the same dataset with that of a large set of modeling techniques and we find that GBR is the only technique that achieves an average prediction error of less than 10% and outperforms other techniques by up to 2 orders of magnitude.

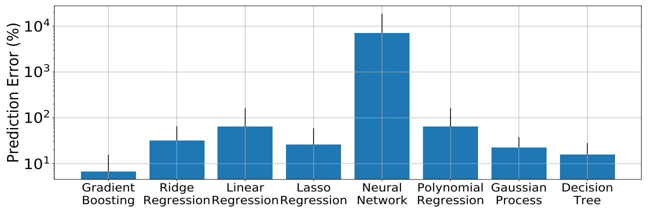

Figure 15: *Average prediction error for various common modelling techniques including GBR*

[10]A non-inclusive cache writes back every modification or eviction that happens in L2 cache to the LLC

How does prediction error change when composition is used? To evaluate the accuracy of composition, Figure 16 compares SLOMO's prediction error in the following scenarios; (1) when contentiousness is directly measured from the server (*i.e.,* no composition is needed), (2) when the contentiousness vector is composed with the PCM metrics of each competitor when run solo and (3) when contentiousness is composed using SLOMO's methodology. We find that our composition mechanism introduces an overhead that the worst case does not introduce more than 5% of additional error. On the other hand, using the solo contentiousness of the competitors to compose the aggregate contentiousness vector can increase the error by up to 20%. We note that composition does not introduce additional time overheads during online prediction.

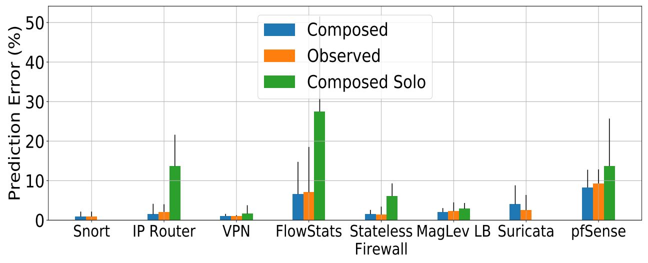

Figure 16: SLOMO*'s prediction error when contentiousness vector is observed versus composed.*

7.3 SLOMO use cases

To demonstrate SLOMO's practical benefits, we first show that SLOMO can enable efficient online scheduling in an NFV cluster. Then, we demonstrate how SLOMO can enable better resource partitioning using CAT[6].

Scheduling with SLOMO**:** We consider an online scheduling scenario where the operator periodically receives NF scheduling requests, containing the NF's description and an SLA (*i.e.,* maximum throughput drop relative to solo). The operator's goal is to maximize resource utilization while minimizing SLA violations. Given that the optimal algorithm for this task is NP-complete [37], we opt for a greedy incremental algorithm that evaluates for every node whether the addition of the NF will lead to SLA violations [51]. If there's no feasible schedule then we provision an additional server.

We run 10 simulations of 1000 scheduling requests, where an NF is picked randomly from the pool of profiled NFs and is given a throughput SLA in the 5 – 30% range. We exhaustively run all possible combinations of requests in our deployment to determine the feasible schedules and those result in SLO violations. We compare SLOMO's schedules with those derived using Dobrescu's CAR-based model and ResQ, a contention aware scheduler by Tootoonchian *et al.* [51]. Table 8 presents the simulation results in terms of how many additional machines (%) with respect to the optimal schedule each approach requires and the associated SLO violations (% of requests).

System	Resource overhead (%)	SLO violations
SLOMO	1.5%	0
ResQ [51]	6%	3%
Dobrescu [24]	14%	0

Table 8: *Resource requirements and SLO violations in online NF scheduling.*

SLOMO **and CAT:** We test our NFs on the Broadwell architecture (Skylake does not support CAT) and allocate 25% of the LLC (5MB) to the target NF. The competitors contend for the remaining LLC space. Table 9 shows the average absolute mean prediction error for the target NF using SLOMO against the percentage difference in the target's throughput from its solo performance in the same partition of the LLC. We categorize our target NFs as sensitive vs. insensitive to contention. We observe that SLOMO's prediction error is ~ 3× lower that the "error" an operator would make assuming that CAR achieves perfect isolation and linear performance scaling.

NF	SLOMO error	Observed drop
Not sensitive	<1%	<1%
Sensitive	4.5%	13.8%

Table 9: SLOMO *Prediction error in the presence of CAT*

7.4 Extrapolation

Finally, we show the promise of extrapolation. We use 5 NF types (IP Router, FlowStats, VPN, Snort and Maglev LB) and at each experiment we change either the number of unique traffic flows or the ruleset size by a factor of up to ±20%. For example, for Flowstats, the initial NF processed 400K of unique flows. Our new configurations were in the [320K-480K] range. That is because, intuitively, one can only extrapolate when the NF's behavior is not expected to change drastically. Considering the discussion in §7.1 about the behavior of FlowStats being radically different for 40K and 400K flows, extrapolating this change would be unwise.

Figure 17 shows the absolute mean prediction error broken down across NFs that are sensitive to contention (IPRouter, FlowStats, Snort) with that of non-sensitive NFs. For contention sensitive NFs, using the extrapolated sensitivity adds on average 3% to the prediction error compared to the average error when the sensitivity function is already known. Without extrapolation *i.e.,* when we use the old sensitivity function, the prediction error on average doubles. For non-sensitive NFs, the difference in error is marginal.

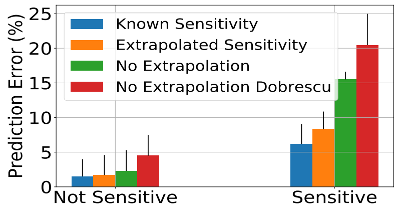

Figure 17: *Prediction error of extrapolated sensitivity functions*

8 DISCUSSION

What if NF_i **does not use packet acceleration?** In cases of NFs that do not use packet acceleration, we find that the kernel-based network stack will become the major performance bottleneck and determine NFs' sensitivity [35]. However, as competing NFs run on dedicated cores and with appropriate core-interrupt affinity, NF_i's contentiousness can still be characterized using SLOMO.

Can SLOMO **account for other sources of contention?** Additional sources of contention fall within SLOMO's scope as long as PCM exposes utilization metrics about these resources. For instance, relaxing the one NF per core assumption might lead to contention in the L2 cache, breaking NUMA affinity might result in contention at the QPI interconnect, and multiple traffic flows might contend for NIC resources [46]. Because PCM exposes data about L2 hit rates and QPI bandwidth, we would expect SLOMO to perform well in this scenario. On the other hand, SLOMO cannot currently address contention at the NIC because PCM does not measure resources at the NIC. We leave an exploration of what metrics to collect in scenarios beyond the scope of PCM to future work.

9 RELATED WORK

Performance prediction: Prior work in the architecture community identifies performance degradation due to shared-resource contention [26, 47, 48]. The focus here is to inform processor designs that can dynamically reallocate their shared resources during runtime without offline profiling of each process. These works are complimentary to ours as the focus is on the architectural design that enables insights similar to SLOMO's.

NF management: Work in NFV management acknowledges the degradation problem and suggested workarounds via NF placement; *e.g.,* E2 [41] and CoMB [44] consolidate NFs to avoid cross-switch traffic. However, these works do not model contention-related performance degradation. Other works have also looked at scaling NFs based on observable triggers such as congestion, long tail latency or packet drops [28, 31]. Using SLOMO for performance prediction can improve the utilization and SLAs for these efforts.

NF isolation: Recent efforts explore ways to provide performance isolation between NFs on the same host; *e.g.,* Netbricks [42] and ResQ [51]. These efforts are complementary to SLOMO which can be used to inform the design of isolation policies.

Prediction and verification via symbolic execution: Recent work by Pedrosa *et al.* [43] uses symbolic execution to understand the execution paths a NF can take and uses a cache simulator to identify adversarial workloads. This requires access to NF source code, whereas SLOMO can work with blackbox NF realizations. Similar works rely on symbolic execution either to provide performance contracts or to verify the correctness of Network Functions [30, 53].

10 CONCLUSIONS

Providing performance guarantees when NFs share hardware remains an elusive goal in NFV. The ability to accurately predict the potential performance for a future colocation configuration can inform the provisioning and placement decisions in today's NFV orchestration frameworks. While prior work identified the memory contention problem in NFV, it treated memory as monolithic whole and cannot provide sufficient accuracy in today's NFV landscape. In this work we systematically investigate the memory subsystem and the sources behind contention-related slowdown. Our insights enable the development of SLOMO, a performance prediction framework for NFV. We show that relative to prior work SLOMO reduces prediction error by 2-5× and enables 6-14% more efficient cluster utilization.

Ethics: *This work does not raise any ethical issues.*

Acknowledgments: We thank our shepherd and the reviewers for their feedback. This work was supported in part by the CONIX Research Center, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA, and NSF awards 1440056, 1440065, and 1700521.

REFERENCES

[1] Aggregate PCM Metrics. https://software.intel.com/en-us/forums/software-tuning-performance-optimization-platform-monitoring/topic/277497.

[2] AT&T. Domain2. https://www.att.com/Common/about_us/pdf/AT&T%20Domain%202.0%20Vision%20White%20Paper.pdf.

[3] AT&T. ECOMP. https://policyforum.att.com/wp-content/uploads/2017/03/ecomp-architecture-whitepaper-att.pdf.

[4] Dpdk-performance tuning guide. https://doc.dpdk.org/guides-16.11/linux_gsg/nic_perf_intel_platform.html.

[5] ETSI. Network Functions Virtualization. https://portal.etsi.org/NFV/NFV_White_Paper.pdf.

[6] Intel cache allocation technology. https://www.intel.com/content/www/us/en/communications/cache-monitoring-cache-allocation-technologies.html.

[7] Intel direct data i/o technology. https://www.intel.com/content/www/us/en/io/data-direct-i-o-technology.html.

[8] Intel PCM. https://github.com/opcm/pcm.

[9] Intel skylake-x review: Core i9 7900x, i7 7820x and i7 7800x tested. https://www.anandtech.com/show/11550/the-intel-skylakex-review-core-i9-7900x-i7-7820x-and-i7-7800x-tested/4.

[10] Linux Foundation. OPNFV. https://www.opnfv.org/.

[11] Mellanox-performance tuning guide. https://community.mellanox.com/s/article/performance-tuning-for-mellanox-adapters.

[12] Reducing os jitter due to per-cpu kthreads. https://www.kernel.org/doc/html/latest/admin-guide/kernel-per-CPU-kthreads.html.

[13] Snort: Network Intrusion Detection & Detection System. https://www.snort.org.

[14] SR-IOV. https://github.com/intel/sriov-network-device-plugin.

[15] Suricata: Open source ids, ips, nsm engine. https://suricata-ids.org.

[16] A. Abel, F. Benz, J. Doerfert, B. Dörr, S. Hahn, F. Haupenthal, M. Jacobs, A. H. Moin, J. Reineke, B. Schommer, et al. Impact of resource sharing on performance and performance prediction: A survey. In *International Conference on Concurrency Theory*, pages 25–43. Springer, 2013.

[17] L. A. Barroso, J. Clidaras, and U. Hölzle. The datacenter as a computer: An introduction to the design of warehouse-scale machines. *Synthesis lectures on computer architecture*, 8(3):1–154, 2013.

[18] J. Benesty, J. Chen, Y. Huang, and I. Cohen. Pearson correlation coefficient. In *Noise reduction in speech processing*, pages 1–4. Springer, 2009.

[19] C. M. Buechler and J. Pingle. pfsense: The definitive guide. *Reed Media Services*, 2009.

[20] C. Delimitrou and C. Kozyrakis. Paragon: Qos-aware scheduling for heterogeneous datacenters. In *ACM SIGPLAN Notices*, volume 48, pages 77–88. ACM, 2013.

[21] C. Delimitrou and C. Kozyrakis. Quasar: resource-efficient and qos-aware cluster management. *ACM SIGPLAN Notices*, 49(4):127–144, 2014.

[22] T. G. Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer, 2000.

[23] I. D. Direct. I/o technology (intel ddio) a primer, 2012.

[24] M. Dobrescu, K. Argyraki, and S. Ratnasamy. Toward predictable performance in software packet-processing platforms. In *Proc. NSDI 12*, pages 141–154, San Jose, CA, 2012. USENIX.

[25] M. Dobrescu, N. Egi, K. Argyraki, B. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy. Routebricks: Exploiting parallelism to scale software routers. In *Proc. SOSP 2009*, SOSP '09, pages 15–28, New York, NY, USA, 2009. ACM.

[26] E. Ebrahimi, C. J. Lee, O. Mutlu, and Y. N. Patt. Fairness via source throttling: a configurable and high-performance fairness substrate for multi-core memory systems. In *ACM Sigplan Notices*, volume 45, pages 335–346. ACM, 2010.

[27] J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[28] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella. Opennf: Enabling innovation in network function control. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 163–174. ACM, 2014.

[29] D. M. Hawkins. The problem of overfitting. *Journal of chemical information and computer sciences*, 44(1):1–12, 2004.

[30] R. Iyer, L. Pedrosa, A. Zaostrovnykh, S. Pirelli, K. Argyraki, and G. Candea. Performance contracts for software network functions. In *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*, pages 517–530, 2019.

[31] M. Kablan, A. Alsudais, E. Keller, and F. Le. Stateless network functions: Breaking the tight coupling of state and processing. In *NSDI*, pages 97–112, 2017.

[32] E. Kohler, R. Morris, B. Chen, J. Jannotti, and F. Kaashoek. The click modular router. In *Proc. TOCS 2000*, volume 18, pages 263–297. ACM, 2000.

[33] C. Kozyrakis, A. Kansal, S. Sankar, and K. Vaid. Server engineering insights for large-scale online services. *IEEE micro*, 30(4):8–19, 2010.

[34] M. Kurth, B. Gras, D. Andriesse, C. Giuffrida, H. Bos, and K. Razavi. Netcat: Practical cache attacks from the network, 2020.

[35] J. Li, N. K. Sharma, D. R. K. Ports, and S. D. Gribble. Tales of the tail: Hardware, os, and application-level sources of tail latency. In *Proceedings of the ACM Symposium on Cloud Computing*, SOCC '14, page 1–14, New York, NY, USA, 2014. Association for Computing Machinery.

[36] Y. Li and M. Chen. Software-defined network function virtualization: A survey. *IEEE Access*, 3:2542–2553, 2015.

[37] E. C. man Jr, M. Garey, and D. Johnson. Approximation algorithms for bin packing: A survey. *Approximation algorithms for NP-hard problems*, pages 46–93, 1996.

[38] J. Mars, L. Tang, and R. Hundt. Heterogeneity in "homogeneous" warehouse-scale computers: A performance opportunity. *IEEE Computer Architecture Letters*, 10(2):29–32, 2011.

[39] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa. Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations. In *Proceedings of the 44th annual IEEE/ACM International Symposium on Microarchitecture*, pages 248–259. ACM, 2011.

[40] L. A. Mauricio, M. G. Rubinstein, and O. C. Duarte. Proposing and evaluating the performance of a firewall implemented as a virtualized network function. In *2016 7th International Conference on the Network of the Future (NOF)*, pages 1–3. IEEE, 2016.

[41] S. Palkar, C. Lan, S.Han, K. Jang, A. Panda, S. Ratnasamy, L. Rizzo, and S. Shenker. E2: A framework for nfv applications. In *Proc. SOSP 2015*, SOSP '15, pages 121–136, New York, NY, USA, 2015. ACM.

[42] A. Panda, S. Han, K. Jang, M. Walls, S. Ratnasamy, and S. Shenker. Netbricks: Taking the v out of nfv. In *OSDI*, pages 203–216, 2016.

[43] L. Pedrosa, R. Iyer, A. Zaostrovnykh, J. Fietz, and K. Argyraki. Automated synthesis of adversarial workloads for network functions. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, pages 372–385, New York, NY, USA, 2018. ACM.

[44] V. Sekar, N. Egi, S. Ratnasamy, M. Reiter, and G. Shi. Design and implementation of a consolidated middlebox architecture. In *Proc. of NSDI 2012*, NSDI'12, pages 24–24, Berkeley, CA, USA, 2012. USENIX Association.

[45] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar. Making middleboxes someone else's problem: network processing as a cloud service. *ACM SIGCOMM Computer Communication Review*, 42(4):13–24, 2012.

[46] B. Stephens, A. Akella, and M. Swift. Loom: Flexible and efficient {NIC} packet scheduling. In *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*, pages 33–46, 2019.

[47] L. Subramanian, V. Seshadri, A. Ghosh, S. Khan, and O. Mutlu. The application slowdown model: Quantifying and controlling the impact of inter-application interference at shared caches and main memory. In *Proc. ACM MICRO 2015*, pages 62–75. ACM, 2015.

[48] L. Subramanian, V. Seshadri, Y. Kim, B. Jaiyen, and O. Mutlu. Mise: Providing performance predictability and improving fairness in shared main memory systems. In *High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on*, pages 639–650. IEEE, 2013.

[49] L. Tang, J. Mars, and M. L. Soffa. Contentiousness vs. sensitivity: Improving contention aware runtime systems on multicore architectures. In *Proceedings of the 1st International Workshop on Adaptive Self-Tuning Computing Systems for the Exaflop Era*, EXADAPT '11, pages 12–21, New York, NY, USA, 2011. ACM.

[50] L. Tang, J. Mars, N. Vachharajani, R. Hundt, and M. L. Soffa. The impact of memory subsystem resource sharing on datacenter applications. In *ACM SIGARCH Computer Architecture News*, volume 39, pages 283–294. ACM, 2011.

[51] 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*. USENIX, 2018.

[52] Y. Yang and J. O. Pedersen. A comparative study on feature selection in text categorization. In *Icml*, volume 97, page 35, 1997.

[53] A. Zaostrovnykh, S. Pirelli, R. Iyer, M. Rizzo, L. Pedrosa, K. Argyraki, and G. Candea. Verifying software network functions with no verification expertise. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 275–290, 2019.

[54] Z.-H. Zhou. *Ensemble methods: foundations and algorithms*. Chapman and Hall/CRC, 2012.