NFVPermit: Toward Ensuring Performance Isolation in NFV-Based Systems

Venkatarami Reddy Chintapalli, *Graduate Student Member, IEEE*, Sai Balaram Korrapati, Madhura Adeppady, Bheemarjuna Reddy Tamma, *Senior Member, IEEE*, Antony Franklin A, *Senior Member, IEEE*, and Balaprakasa Rao Killi, *Senior Member, IEEE*

***Abstract*—Network Functions Virtualization (NFV) promises programmability and cost savings by replacing hardware middleboxes with more flexible Virtual Network Functions (VNFs) on commodity servers. But, the current virtualization technologies do not fully isolate the system resources like Last Level Cache (LLC) and Memory Bandwidth (MB); therefore, co-location of VNFs on the same commodity server causes interference effects which might severely impact the performance of VNFs in terms of throughput, latency, etc. Contention at LLC is one of the root causes of this performance degradation and it is addressed by LLC resource partitioning. But, it remains unexplored the impact of both LLC and MB on VNF performance. In this work, we investigate the importance of MB partitioning along with LLC partitioning to achieve performance isolation in NFV-based systems. Allocating these system resources among co-located VNFs to meet Service Level Agreements (SLAs) is challenging due to the dynamic nature of traffic and varying functionality of VNFs. In this work, we formulate the resource allocation problem as an Integer Linear Programming (ILP) for maximizing the number of accepted VNF requests with SLA guarantees. Since the problem is NP-hard, we present a polynomial time ϵ-approximation scheme. Further, we propose a heuristic approach named *NFVPermit*, a resource manager for NFV-based systems that tries to ensure performance isolation among co-located VNFs based on their current traffic rates and SLA requirements. Through extensive experiments, we show how *NFVPermit* outperforms state-of-the-art and baseline approaches.**

***Index Terms*—NFV, VNF, resource contention, performance isolation, last level cache partitioning, memory bandwidth partitioning, resource allocation.**

Manuscript received 15 March 2022; revised 11 December 2022; accepted 28 April 2023. Date of publication 22 May 2023; date of current version 6 July 2023. The associate editor coordinating the review of this article and approving it for publication was M. Shojafar. *(Corresponding author: Venkatarami Reddy Chintapalli.)*

Venkatarami Reddy Chintapalli was with the Department of Computer Science and Engineering, Indian Institute of Technology Hyderabad, Hyderabad 502285, India. He is now with the Department of CSE, National Institute of Technology Calicut, Kozhikode 673601, India (e-mail: cs17resch01007@iith.ac.in).

Sai Balaram Korrapati was with the Department of Computer Science and Engineering, Indian Institute of Technology Hyderabad, Hyderabad 502285, India (e-mail: es18btech11011@iith.ac.in).

Madhura Adeppady was with the Department of Computer Science and Engineering, Indian Institute of Technology Hyderabad, Hyderabad 502285, India. She is now with the Department of Electronics and Telecommunications, Politecnico di Torino, 10129 Turin, Italy (e-mail: cs17mtech11013@iith.ac.in).

Bheemarjuna Reddy Tamma and Antony Franklin A are with the Department of Computer Science and Engineering, Indian Institute of Technology Hyderabad, Hyderabad 502285, India (e-mail: tbr@iith.ac.in; antony.franklin@iith.ac.in).

Balaprakasa Rao Killi is with the Department of Computer Science and Engineering, National Institute of Technology Warangal, Warangal 506001, India (e-mail: bsprao@nitw.ac.in).

Digital Object Identifier 10.1109/TNSM.2023.3278731

I. Introduction

IN LEGACY communication networks, Network Functions (NFs) like Deep Packet Inspection (DPI), firewall, etc are typically realized using dedicated network devices, which are commonly known as middleboxes. Although such middleboxes are quite popular and able to handle heavy traffic loads, they are expensive to buy and inflexible to modify or reprogram their functionality to meet ever-changing network requirements. Network Functions Virtualization (NFV) addresses these limitations of (proprietary) middleboxes by separating hardware and software. In NFV, NFs are deployed on commodity servers with software-based implementations known as Virtual Network Functions (VNFs), which bring easier operation, lower costs, programmability, and resource sharing between VNFs [1]. These VNFs are usually deployed in Virtual Machines (VMs), containers, or as native processes, with one or more dedicated CPU cores [2] of commodity servers. Despite their benefits, the virtualization layer's additional overhead results in performance penalties in terms of latency and throughput. To alleviate these performance penalties, commodity servers offer numerous optimization solutions such as Data Plane Development Kit (DPDK) [3], Direct Data I/O (DDIO) technology [4], etc. Yet, VNFs co-located in the same server compete for various system resources, making it challenging to achieve predictable and deterministic performance. Addressing this challenge is crucial for successful deployments of different network services (deployed in VNFs) with stringent Service Level Agreement (SLA) requirements on commodity servers.

Network operators often pack many VNFs in the same commodity server to improve its resource utilization [5], [6], [7], [8], [9] and reduce their CAPEX/OPEX. But, these co-located VNFs may contend for some or all of the shared processor resources like CPU cores, Last Level Cache (LLC), Memory Bandwidth (MB), etc., which leads to performance interference that can cause up to 50% of throughput degradation compared to when a VNF runs alone [10], [11]. In the cloud computing jargon, the performance interference problem is also known as the noisy neighbor problem and the co-located VNFs responsible for performance degradation

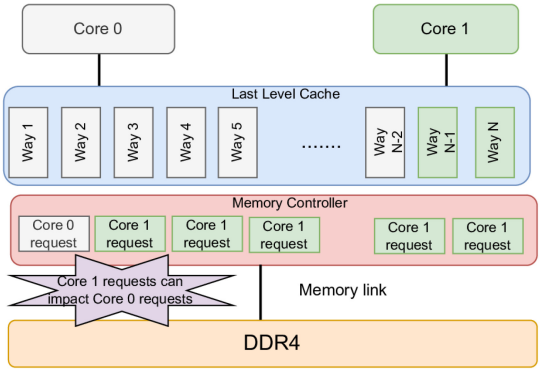


Fig. 1. The problem of allocating only the LLC ways to the co-located VNFs is demonstrated here. The name "way" is given to a partitioned fraction of the full cache. The NF service running on *core* 0 has high priority, and more cache ways are given to it. The other service deployed on *core* 1 gets relatively fewer cache ways, resulting in frequent cache misses and thus saturating the memory link. This might cause priority inversion and impact the performance of the NF service running in *core* 0.

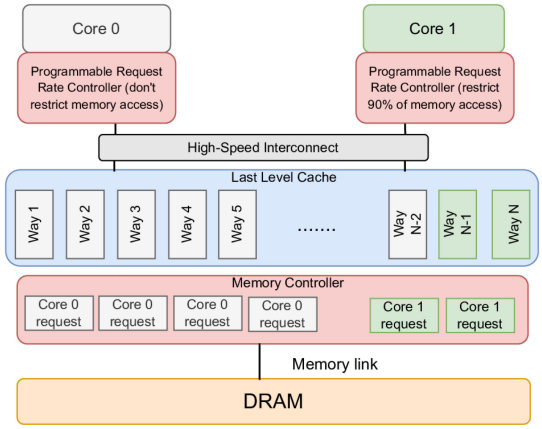


Fig. 2. Controlling LLC and memory bandwidth. Using MBA, MB link access for the NF service running on *core* 1 is restricted to assure that the high priority NF service running on *core* 1 indeed gets high priority.

are regarded as the noisy neighbors. It has been identified that contention at LLC is one of the primary causes for performance degradation in NFV-based systems [12], [13], [14], [15] and it is addressed by LLC resource partitioning in case of x86 servers using Intel's Cache Allocation Technology (CAT). CAT allows partitioning the LLC into cache ways and allocating them exclusively to individual cores [16]. However, we observe that a network service, application process, or function with fewer LLC ways experiences frequent cache misses and thereby saturates the MB, indirectly impacting the co-located high-priority services with more LLC ways, *i.e.,* causing priority inversion issue (refer Fig. 1). To avoid the problem of MB saturation, the network operators should also partition MB along with LLC to achieve performance isolation among the co-located services/VNFs. Such performance isolation helps in achieving SLA (e.g., throughput, latency, etc) guarantees in NFV-based systems.

Intel recently introduced Memory Bandwidth Allocation (MBA) technology, which provides indirect and approximate control over MB given to each core [17]. Using MBA, one can change the amount of MB allocated to different co-located services/VNFs based on their SLA requirements and thus improve the performance of high-priority VNFs as shown in Fig. 2.

The combination of bare minimum LLC ways and MB resources that need to be allocated to achieve the guaranteed SLA (e.g., throughput) is distinctive for different VNFs. Furthermore, it also varies with the incoming traffic rate of the VNF. So, instead of over-provisioning for its peak resource demand (max allocation), an ideal solution could be to dynamically adjust the amount of resources allocated to a VNF based on its current demand (dynamic allocation). But it is challenging to find the optimal resource combination for a given set of co-located VNFs due to the dynamic and fluctuating nature of incoming traffic rates, different functionality of VNFs, and availability of multiple combinations of LLC ways and MB to satisfy their SLA requirements.

For making the dynamic resource allocation very efficient, the network operators must somehow map the VNF's performance specification given in terms of SLA to an adequate resource allocation. Towards this, we propose the usage of a VNF profiler for profiling various VNFs available in [18] to understand variations in their performance as a result of different combinations of LLC and MB resources for different incoming traffic rates. We then formulate dynamic resource allocation as an Integer Linear Programming (ILP) with the objective of maximizing the number of SLA guaranteed VNFs. Since the problem is NP-hard, we present a polynomial time ϵ-approximation scheme. Further, we propose a low complexity heuristic approach named *NFVPermit*, which acts as a resource manager for NFV-based systems and tries to ensure performance isolation (which helps to guarantee SLAs) among co-located VNFs. The *NFVPermit* makes use of CAT and MBA mechanisms to dynamically control the allocation of LLC ways and MB among co-located VNFs based on their current incoming traffic rates and SLA requirements. To the best of our knowledge, none of the existing works have studied the importance of joint LLC and MB allocation to improve the SLA guarantee for VNFs while ensuring performance isolation between the VNFs in an NFV-based system.

In summary, the main contributions of this work are as follows:

- We demonstrate the importance of effectively allocating LLC and MB resources for ensuring performance isolation among the co-located services/VNFs deployed on commodity servers.
- We propose a method to build a profiling table that lists out the bare minimum LLC ways and MB resources needed for the target VNFs to ensure their SLA guarantees.
- We formulate dynamic resource allocation problem as an ILP-based optimization problem with the objective of maximizing the number of SLA-guaranteed VNFs.
- Since this ILP model is NP-hard, we propose a heuristic algorithm named *NFVPermit* for the proposed dynamic resource allocation problem that can run in an online fashion even for a large number of VNFs in the system.

Paper organization: The rest of the paper is organized as follows. The related work is presented in Section II. Section III provides a brief overview of CAT and MBA and motivation behind this work. Section IV defines the system architecture

of NFV-based systems and presents an ILP-based optimization model for dynamic resource allocation. Next, Section V proposes a heuristic algorithm named *NFVPermit* for dynamic resource allocation. Section VI evaluates the proposed ILP model and *NFVPermit* solution and provides a comparison with some baseline and state-of-art approaches. Finally, Section VII concludes the paper.

II. Related Work

Prior work [10] investigated the performance interference problem when multiple types of VNFs are consolidated on a single commodity server. They have shown that consolidated VNFs compete for various shared resources causing up to 50% performance degradation. Recent years have witnessed an emerging number of packet processing frameworks such as E2 [19], Netbricks [20], NFVnice [21], ClickOS [22], OpenNetVM [18], and NFP [23]. These frameworks provide various features to the VNF developers and service providers such as packet steering, load balancing, network I/O, flow management, scalability, etc. Although performance interference causes severe performance degradation, none of these frameworks solve the resource contention problem among consolidated VNFs.

Recent works on solving performance interference problem can be broadly classified into supply-demand models, prediction models, and resource partitioning solutions.

Supply-demand models: Recent works [24], [25] quantified VNF interference as a mismatch between the resource supply provided by the server and the resource demand of the competing VNFs consolidated on the same server. These works have considered only CPU and memory as sources of contention, which limits their applicability in addressing performance interference. Moreover, competition for one type of resource negatively affects a different kind of resource(s), which cannot be measured using supply-demand models.

Prediction models: Prior works [26], [27], [28], [29], [30] have developed models to predict the observed performance degradation of a target VNF due to consolidation. The effect of shared LLC on software-based packet processing performance on multi-core processors was investigated in [31], [32]. They have developed linear models for predicting the VNF performance based on Cache Access Rate (CAR) [31] and working set size of the competitors [32]. However, as shown in [26], these works suffer from poor performance in modern memory architecture, as resource contention cannot be modeled using a single-metric. In SLOMO [26], authors have investigated memory subsystem and various resource contentions that are responsible for performance degradation due to co-location. Based on the observation, they have developed a multi-variable performance prediction framework for NFV. Although SLOMO provides a nearly accurate model for predicting performance, it does not guarantee performance isolation among the co-located VNFs. Furthermore, the effect of fluctuating input traffic rates to the co-located VNFs which is responsible for further deteriorating the performance is not considered in SLOMO. In several recent works [33], [34], [35], [36], [37], observed client-side performance is used to detect, locate, and mitigate performance interference using profiling information. However, the primary focus of these works is on detecting performance interference rather than resource partitioning among the competing co-located services. The authors are of [38], [39], [40], [41] studied the impact of core selection in a non-uniform memory access (NUMA) system on the performance of VNFs and Service Function Chains (SFCs).

Resource partitioning solutions: A large number of prior works focused on resource partitioning among the competing generic workloads [42], [43], [44], [45], [46]. These solutions are not suitable for the NFV environment for two reasons. The first reason is, generic workloads compete for a wide variety of system resources such as memory capacity, storage, network bandwidth, and CPU resources. On the contrary, VNF performance depends mainly on the competition for memory subsystem resources. Secondly, VNFs exhibit idiosyncratic and extreme interactions with the cache hierarchy in comparison with generic workload. Therefore, solutions proposed for generic workload are not suitable for VNFs. In the NFV context, only a few works focus on LLC partitioning to provide performance isolation. ResQ [11] is one such work that leverages CAT to partition the LLC among co-located VNFs. However, ResQ does not provide complete isolation among the competing VNFs as they still compete for MB. Furthermore, ResQ does not support dynamic resource allocation in response to varying incoming traffic rates to the VNFs.

Recently, the importance of the dynamic allocation of LLC and MB to the competing VNFs to avoid performance interference is explored in RESTRAIN [47]. It provides a naive heuristic named RESTRAIN for allocating resources to the running VNFs to guarantee performance isolation among the co-located services. RESTRAIN chooses resource combinations in a greedy manner based on available free resources, while also disregarding effective resource utilization in the system. Our work is an enhancement to RESTRAIN by allocating resources for multiple competing VNFs and maximizing the number of SLA guaranteed VNFs. The proposed NFVPermit scheme penalizes the use of resources depending on the current resource state and iteratively improves the solution by gradually replacing the chosen resource combination with another as long the solution remains feasible. We also mathematically formulate this dynamic LLC and MB allocation problem and provide a polynomial time ϵ-approximation scheme in this article.

III. Background and Motivation

This section provides the necessary background, followed by a motivational experiment that showcases the importance of allocating both LLC and MB to the co-located VNFs to achieve performance isolation.

A. Background

1) Cache Allocation Technology (CAT): Intel's CAT helps to address noisy neighbor problem by providing a software control over the amount of LLC cache that can be utilized by a core. By means of specialized resource tag called Class of

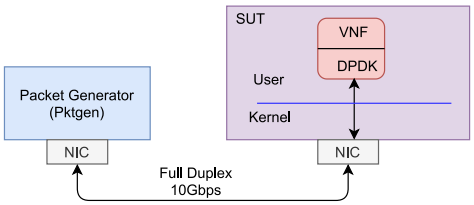


Fig. 3. The measurement setup used for profiling VNFs.

Service (CLOS), CAT can assign partitions of the LLC cache to a particular core, thereby limiting the amount of LLC cache a core can use. The available cache ways/partitions can be dynamically assigned to each CLOS using Capacity Bit Mask (CBM), which indicates how much cache can be used by them. Bits within the CBMs indicate relative amount of cache space allocated and priority level of the CLOS. It is possible to create disjoint or overlapping cache ways using CLOS. Note that, the number of cache partitions is architecture-dependent.

2) Memory Bandwidth Allocation (MBA): MBA is an important tool introduced by Intel to control noisy neighbor problem by enabling prioritization and bandwidth management among competing applications. MBA proposes a Programmable Request Rate Controller (PRRC) between each core and the shared high-speed interconnect to provide approximate and indirect per-core control over MB [17]. Using MBA, it is possible to improve the performance of high priority VNFs by controlling those VNFs that over-utilize the bandwidth relative to their priorities. Like CAT, MBA also exploits CLOS to throttle the MB available for different cores dynamically. Typically, MBA levels in terms of percentage can be assigned to the required CLOS dynamically up to 90% in steps of 10%. MBA injects a delay into each outgoing request to efficiently control traffic from the L2 cache to the LLC without impacting other per-core activities. Note that, upper-bound and granularity of MBA are dependent on the system architecture. We have used CAT and MBA tools for the dynamic allocation of LLC ways and MB resources, respectively to different CPU cores of the commodity servers from Intel onto which the VNFs are pinned to.

B. Motivation

A fair question to ask is the importance of both MB and LLC allocation in the NFV environment, i.e., does the joint allocation of LLC and MB give any performance improvement? To address this question, we experimented with a simple VNF, more specifically, a firewall (also referred to as the target VNF) with *stress-ng* [48] as a noisy neighbor, which causes a lot of LLC misses and MB usage to induce sufficient stress on the memory system. We configured firewall and noisy neighbor to run on isolated server cores to avoid contention for CPU resources.

Experimental setup: Fig. 3 shows the testbed setup consisting of System Under Test (SUT) and packet generator. These components are running on similar type of servers having two Intel Xeon Gold 6126 with 48-core CPUs at 2.60 GHz, 98 GB memory, running Ubuntu 18.04 with kernel 4.4.0-131-generic. The SUT server has 19.7 MB of LLC which is partitioned into

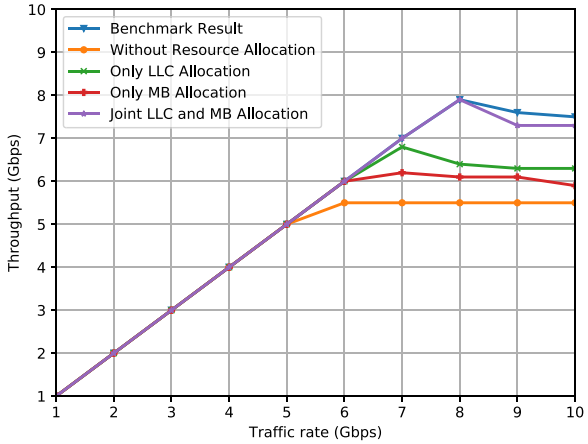


Fig. 4. Variation in throughput of firewall VNF with input traffic rate for various experimental scenarios.

11 cache ways. The *pqos* software package[1] available in Linux is used to control the allocation of LLC ways and MB. Servers are connected in back-to-back manner with 10 Gbps DPDK compatible NICs to reduce switch overheads. Pktgen [49], a DPDK-based high-speed traffic generator, is used for generating input traffic load. On the SUT server, we run a specific DPDK enabled VNF (called the target VNF) which receives the traffic, processes it, and sends it back to the sender. Our study involved the following experimental scenarios, where for each scenario, we measure the throughput of the target VNF by varying traffic rates (64-byte packets) from 1 *Gbps* to 10 *Gbps*. Experiments for each scenario are repeated 10 times and the average results are presented. Each experiment has 60 *seconds* duration.

1) *Scenario 1 (Running VNF alone):* In this scenario, we run the firewall VNF alone and observe the maximum performance obtained. It is considered as the benchmarking result. We also find that with the minimum combination of five LLC ways and 50% MB allocation, the firewall VNF exhibits the same performance as that of the benchmarking result where no such limits are imposed on the usage of LLC and MB by the target VNF.
2) *Scenario 2 (Without resource allocation):* We run the firewall VNF along with *stress-ng* without any resource reservation. The cache hit rate of a VNF depends on the LLC ways allocated to it [47]. Since we have no control over the resources in this scenario, *stress-ng* is aggressively trying to use more LLC ways; as a result, the VNF gets fewer LLC ways, resulting in a reduction in performance.
3) *Scenario 3 (LLC allocation only):* In this scenario, we allocated five LLC ways (from scenario 1) to the firewall VNF and remaining all LLC ways are used by stress-ng. Although we allocated sufficient LLC, performance degradation is observed for the traffic rates above 6 *Gbps* as shown in Fig. 4. This is due to the fact that we are not controlling MB for which both VNFs compete with each other, and the target VNF is not getting sufficient MB.

[1] https://github.com/intel/intel-cmt-cat/tree/master/pqos

TABLE I
MINIMUM RESOURCE COMBINATION REQUIRED FOR DIFFERENT VNFS TO ACHIEVE REQUIRED THROUGHPUT

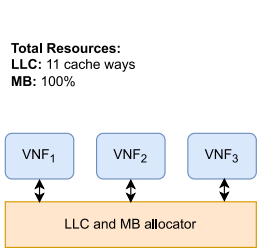
	Throughput (Gbps)								
VNF	**1-2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Router	(2,10)	(2,10)	(2,10)	(2,20)	(2,30) (3,20)	(2,30) (3,20)	(3,20)	(4,30)	(5,40)
Flow Tracker	(2,10)	(2,10)	(2,10)	(3,50) (4,20)	(5,30)	-	-	-	-
Basic Monitor	(2,10)	(2,10)	(2,10)	(2,20)	(2,20)	(2,20)	(3,30)	(3,30)	(4,40)
Simple Forward	(2,10)	(2,10)	(2,10)	(2,10)	(2,10)	(2,20) (3,10)	(3,30) (2,40)	(3,40)	(4,20)
Firewall	(2,10)	(2,10)	(2,20) (3,10)	(2,20) (3,10)	(2,30) (3,10)	(4,40) (5,30)	(5,50)	-	-

Hence, the allocation of only LLC resources to the target VNF is not sufficient to get the required performance.
Observation 1: Allocating only LLC always does not result in performance isolation due to bandwidth saturation, i.e., the available MB for the target VNF is limited by how frequently other co-located services contend for it.

4) *Scenario 4 (MB allocation only):* In this scenario, 50% of MB (from scenario 1) is allocated to the target VNF and the remaining is used by *stress-ng*. Both VNFs are not allocated any LLC ways exclusively and therefore contend for it. Although the target VNF has been given sufficient MB, performance degradation is observed for the traffic rates above 5 *Gbps* as shown in Fig. 4. This is due to the fact that we are not controlling LLC for which both VNFs compete with each other, and the target VNF is not getting sufficient LLC. Hence, the allocation of only MB resources to VNFs is not sufficient to get the required performance.
Observation 2: Since co-located VNFs contend for LLC, allocating only MB does not guarantee performance isolation.
5) *Scenario 5 (Joint LLC and MB allocation):* In this scenario, the target VNF is allocated five LLC ways with 50% of MB. It achieves similar performance as that in Scenario 1, i.e., as if the target VNF is running alone in the system.
Observation 3: The performance of a VNF is highly dependent on both LLC and MB partitioning. Hence, to avoid performance interference, the allocation of MB along with LLC is of paramount importance in NFV-based systems realized using commodity servers.

We have examined the relationship between the allocated resources and VNF's performance and observed that multiple combinations of resource allocation would produce the same performance (as shown in Table I). Different VNFs exhibit different kinds of performance for the same combination of allocated resources (see more details in Section IV-A2). Hence, it is very important to allocate these resources by choosing the best possible combination in order to meet SLAs of the VNFs deployed.

An illustrative example: Fig. 5 shows an example scenario comparing three different resource allocation mechanisms when multiple choices (x, y) are available to guarantee SLAs of the deployed VNFs. Each entry (x, y) in the table of Fig. 5 represents the resource combination, where x and y represent the number of LLC ways and % of MB, respectively. Each VNF has been allocated a certain number of LLC ways and % of MB resources, which are listed in the row *Allocated*, and



	VNF_1	VNF_2	VNF_3	#SLA guaranteed VNFs
Allocated	(5,30)	(2,20)	(2,40)	-
Demand	(3,50) (4,20)	(4,40) (5,30)	(3,40) (5,30)	-
LLC-greedy	(3,50)	(4,40)	(3,40)	2
MB-greedy	(4,20)	(5,30)	(5,30)	2
NFVPermit	(4,20)	(4,40)	(3,40)	3

Fig. 5. An example showing allocation of LLC and MB resources among the co-located VNFs on a commodity server for different resource allocation schemes.

the current resource demand is given in the row *Demand*. The last column in the table (number of SLA-guaranteed VNFs) indicates the number of VNFs that achieved the required throughput for different resource allocation mechanisms. It is important to note that a VNF is said to be SLA guaranteed when it is given sufficient resources for its ingress traffic rate. The naive schemes like LLC-greedy and MB-greedy guarantee SLA for two of the VNFs because they picked entries greedily based on LLC and MB, respectively. In these greedy approaches, either LLC or MB exceeds the maximum resource limit of the system and therefore they fail to guarantee SLA for one of the three VNFs deployed on the system. For example, in the case of LLC-greedy, VNF_1, VNF_2, and VNF_3 select entries (3, 50), (4, 40), and (3, 40), respectively where the aggregate of required MB exceeds 100%. That is, sufficient resources are allocated only to two of the VNFs, and hence the number of SLA-guaranteed VNFs is only two. Similarly, in the case of MB-greedy, the aggregate of required LLC ways is more than the total LLC ways (11) available in the system. In contrast, the proposed method *NFVPermit* (presented later in Section V) penalizes the use of resources depending on the currently available resources and iteratively improves the solution by gradually replacing the chosen resource combination with another as long as the solution remains feasible, which leads to guaranteeing SLA for all of the VNFs deployed.

In a nutshell, allocating MB and LLC among the co-located VNFs leads to the effective utilization of system resources and avoids performance interference, which we are exploring further in this article.

IV. NFV SYSTEM ARCHITECTURE AND PROBLEM FORMULATION

In this section, we describe the proposed NFV system architecture and then formulate the resource allocation problem faced in NFV-based systems.

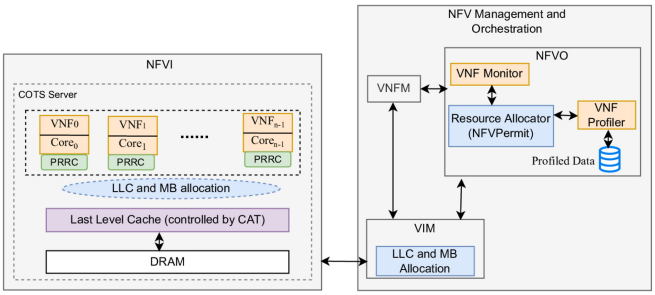


Fig. 6. System architecture.

A. System Architecture and Work Flow

Fig. 6 shows the main components of the proposed NFV system architecture, which consists of three new modules namely VNF monitor and resource allocator (NFVPermit), and VNF profiler. These modules are part of NFV Orchestrator (NFVO). The NFV MANO component manages the physical infrastructure's available resources in compliance with ETSI [50]. The Virtual Infrastructure Manager (VIM) takes care of the allocation of the LLC and MB resources to the competing VNFs on the commodity server with the help of NFVPermit (proposed approach). To prevent contention for CPU resources, dedicated core(s) is allocated solely to each target VNF, which also reaps in the benefits of exclusive L1 and L2 cache [51]. The functionalities of each module are described in the following.

1) VNF Monitor: The VNF monitor module continuously tracks each co-located VNF's traffic rate and sends it to the resource allocator. There are a number of tools available for realizing this, one of which is *collectd* [52]. The granularity of reporting tracked data will differ depending on the usecase. Since QoS-sensitive services demand higher levels of SLA, the granularity of interval for them is very low relative to other types of services.

2) Resource Allocator: Resource allocator gets the current traffic rate of each VNF from the monitor module and consults the VNF profiler to determine the best resource combination to satisfy the SLA. Once this information is collected from all of the co-located VNFs, it effectively allocates resources to the VNFs with the objective of maximizing the SLA guaranteed VNFs. It is important to note that we assume co-located VNFs and services are independent and allocated system resources to each target VNF separately. It remains challenging for the resource allocator to allocate resources with deterministic performance. The network operator must somehow map the performance specification in the SLA to an adequate amount of resource allocation in the virtualized infrastructure. The VNF profiler module can be used for this purpose.

3) VNF Profiler: The network service providers can reduce the amount of allocated system resources as long as the VNFs do not experience performance degradation that causes SLA violations. But to ensure SLAs, the impact of allocated resources on the achievable performance of target VNFs should be known. We learned this relationship by profiling VNFs in this work. If such profiling does not exist, the network service providers end up over-provisioning by allocating all

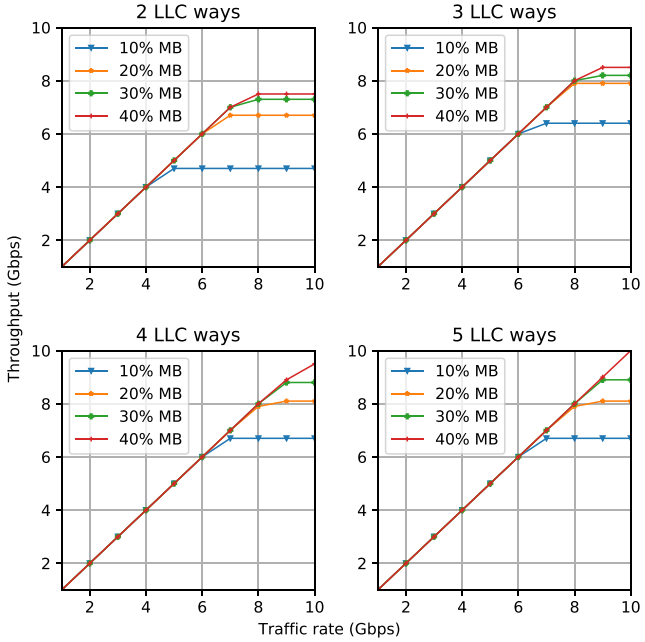


Fig. 7. Observed throughput as a function of resources assigned for router (VNF).

of the system resources to the target VNF as SLA violations are not acceptable. VNF profiler helps the network service providers to determine the optimal combinations of system resources to guarantee SLAs for different traffic loads in advance and therefore it does not suffer from SLA violation issues time-to-time. It profiles each of the VNFs, understands their performance characteristics, and builds a lookup table. The lookup table provides information about the resources required to satisfy specified SLA to the network operator. In this work, we have profiled the performance of five representative VNFs[2] (a virtual router, basic monitor, flow tracker, simple forward, and firewall) under varying workloads and resource configurations. The experimental setup described in Section III-B is used here as well for the profiling. We run each VNF on the SUT (profiling server) which receives the input traffic from *pktgen* (runs on another server).

The VNF profiler begins by constructing the traffic rate-throughput curve as a function of allocated LLC and MB. To build this curve for a target VNF, the profiler runs the VNF alone on the profiling server by pinning it on to the dedicated core(s) and increasing the amount of allocated LLC and MB resources at each step. From the constructed curves, the maximum achievable throughput for a given combination of LLC and MB can be derivable. For scalability, we use a minimal benchmarking procedure given in [53].

Fig. 7 shows the router's traffic rate-throughput curve as a function of LLC and MB allocation. We started the experiment by allocating two LLC ways and 10% of MB to the router. From the curve, it is noticeable that the router's performance is sensitive to both allocated LLC and MB. An example lookup table constructed for some of the VNFs is shown in Table I. For a given input traffic rate, the lookup table

[2]https://github.com/sdnfv/openNetVM/tree/master/examples

provides minimum resources in terms of LLC and MB to achieve the maximum throughput. An important observation is that multiple combinations of LLC and MB are possible for a given input traffic rate, and the lookup table will return all of them. The resource allocator is responsible to decide which combination to choose based on the available resources and needs of the co-located VNFs. As the VNFs face varying traffic conditions during their lifespan, it is critical to address the dynamic allocation of LLC and MB to maximize the SLA guaranteed VNFs. We call this problem as Dynamic Resource Allocation Problem (DRAP) in NFV-based systems. In the following subsection, we formally define the DRAP problem.

B. Problem Formulation

The DRAP is formulated as an Integer Linear Programming (ILP) model. The aim of DRAP is to maximize the number of accepted/satisfied services while ensuring their SLAs with efficient resource utilization. In a given time frame, the inputs include either the new VNF requests and/or existing VNF requests with changes in traffic rates and thus requiring new resource allocation to the VNFs. Due to changes in the input traffic rate of an existing VNF, its resource requirements may increase or decrease with time. Since the VNF resource requirements are strongly related to VNF's characteristics, we adopt a profiling approach to learn these for every type of target VNF. Let the allocated resources of type k to an existing VNF i at time $t-1$ is $C_{i,k}^{t-1}$ and the requirement of resources of type k at time t is $G_{i,k}^{t}$. The current demand of type k resources of VNF i at time t is $G_{i,k}^{t} - C_{i,k}^{t-1}$. We represent the demand of type k resources for j^{th} choice of VNF i at time t with v_{ijk}. We introduce a binary variable $x_{ij} \in \{0,1\}$ to denote whether VNF $i \in F$ is given resource choice j or not for meeting its SLA. All the notations used in this section are given in Table II. The DRAP is formulated as follows:

$$DRAP: \quad \max \sum_{i=1}^{n} \sum_{j=1}^{|R_i|} x_{ij} \tag{1}$$

$$s.t \quad \sum_{i=1}^{n} \sum_{j=1}^{|R_i|} x_{ij} * v_{ijk} \leq L_k \quad \forall k \in [1,m] \tag{2}$$

$$\sum_{j=1}^{|R_i|} x_{ij} \leq 1 \quad \forall i \in [1,n] \tag{3}$$

$$x_{ij} \in \{0,1\}, \quad \forall i \in [1,n], j \in [1,|R_i|] \tag{4}$$

Eqn. (1) shows that the objective of ILP model is to maximize the total number of accepted/satisfied VNF requests, so that they meet their SLA requirements. For each resource type k, Eqn. (2) ensures that the sum of allocated resources at time t does not exceed the residual resources, L_k. Eqn. (3) ensures that for each VNF at most one choice (or entry) is selected. Eqn. (4) indicates whether we can allocate resources or not to a VNF for guaranteeing its SLA.

Theorem 1: DRAP is NP-hard.

Proof: We start the proof by introducing the Multidimensional Multiple-choice Knapsack Problem (MMKP). Assume n groups are given with l_i items in group i. Each item in the group has a specific value and requires m resources. MMKP aims to select exactly one item from each group to maximize the total value of the collected items, subject to the knapsack's m resource constraints. Akbar et al. demonstrated that the MMKP problem is NP-hard [54]. We can transform an instance of MMKP into an instance of DRAP as follows: i) consider each group in the MMKP as a VNF in *DRAP*. ii) consider each item in group i as a resource requirement choice of VNF i. iii) consider that exactly one item can pick from each group as selecting at most one resource combination of VNF i, and iv) set the resource constraints of knapsack to be equal to the resource availability constraints of the server. The transformation procedure can be completed in polynomial time of the input size. As a consequence, MMKP is polynomially reducible to DRAP, hence DRAP is also an NP-hard problem. ■

TABLE II
GLOSSARY FOR THE PROPOSED ILP MODEL

Notation	Description
n	Number of VNF requests
T	Total number of time intervals
R_i	Set of available choices for VNF i
x_{ij}	1: represents SLA guaranteed for VNF i 0: otherwise If a VNF gets sufficient resources then it is considered as guaranteeing its SLA.
L_k	Available resources of type k (such as LLC and MB)
S_k	Total system resources of type k (such as LLC and MB)
m	Number of resource types
$G_{i,k}^{t}$	Requirement of resources of type k to a VNF i at time instance t
$C_{i,k}^{t-1}$	Allocated resources of type k to a VNF i at time instance $t-1$
v_{ijk}	Demand of type k resources for j^{th} choice of VNF i at time t

C. Approximation Scheme for DRAP

The proposed approximation scheme considers linear program for DRAP by relaxing integer variables. The linear program $DRAP_{lp}$ is defined as follows:

$$DRAP_{lp}: \max \sum_{i=1}^{n} \sum_{j=1}^{|R_i|} x_{ij} \tag{5}$$

subject to (2), (3) and the following

$$0 \leq x_{ij} \leq 1, \quad \forall i \in [1,n], j \in [1,|R_i|] \tag{6}$$

Let $q = min\{n, m/\epsilon\}$ where m denotes the number of resource types. Let H be a subset of F with size at most q and h is a function that maps each element $i \in H$ to a choice in R_i. Given a set $H \subseteq F$ with $|H| \leq q$ and a function $h: H \rightarrow R_i$, we can assign values to some variables x_{ij} in $DRAP_{lp}$ using Eqn. (7). Let the resulting linear program is represented by $DRAP_{lp}(H,h)$.

$$x_{ij} = \begin{cases} 1, \ i \in H \ \ and \ \ j = h(i) \\ 0, \ i \in H \ \ and \ \ j \neq h(i) \\ 0, \ i \notin H \end{cases} \tag{7}$$

For a fixed number of VNF requests n and approximation factor $\epsilon > 0$, we present a polynomial time ϵ-approximation

scheme (PTAS) for DRAP in Algorithm 1. Let $X^f(H,h)$ be an optimal solution to $DRAP_{lp}(H,h)$. The PTAS scheme solves $DRAP_{lp}(H,h)$ for all $H \subseteq F$, $H \leq q$ and then rounding down $X^f(H,h)$ to an integer solution $X^I(H,h)$. Let z^f be the objective value of $DRAP_{lp}(H,h)$ and z^I be the objective value after rounding down the solution of $DRAP_{lp}(H,h)$.

$$z^I = \sum_{i=1}^{n} \sum_{j=1}^{|R_i|} x_{ij}^I \tag{8}$$

$$z^f = \sum_{i=1}^{n} \sum_{j=1}^{|R_i|} x_{ij}^f \tag{9}$$

Let $\hat{x}$ be the rounded down integral solution to the $DRAP_{lp}(H,h)$ with the maximum objective value.

$$\hat{z} = max\left\{z^I\right\} \tag{10}$$

$$\hat{z} \geq z^I \ \ \forall H \subseteq F, H \leq q \tag{11}$$

If $\hat{x}$ accepts at most q VNF requests, then we are done. Otherwise, let $\hat{H}$ be the set of VNF requests selected by $\hat{x}$. For $i \in \hat{H}$, let $\hat{h}(i)$ be the choice of VNF requests selected by $\hat{x}$. Consider the iteration of Algorithm 1 in which $H = \hat{H}$ and $h = \hat{h}$. Let n' be the number of VNF requests that are not there in $\hat{H}$. Let $g = \sum_{i=1}^{n} |R_i|$ be the number of x_{ij} variables in $DRAP_{lp}(\hat{H},\hat{h})$. Slack variables are added to transform the inequality constraints into equality constraints in standard form. Therefore, $m+n'$ additional slack variables are added while converting the inequality constraints into the following equality constraints.

$$\sum_{i=1}^{n} \sum_{j=1}^{|R_i|} v_{ijk} x_{ij} + s_k = 1, \forall 1 \leq k \leq m \tag{12}$$

$$\sum_{j=1}^{|R_i|} x_{ij} + s_i' = 1, \forall 1 \leq i \leq n' \tag{13}$$

Hence, there are $g + m + n'$ variables in $DRAP_{lp}(\hat{H},\hat{h})$ and $m+n'$ constraints (from Eqns. (12) and (13)). Therefore, there can be at most $m + n'$ positive variables in any basic solution of $DRAP_{lp}(\hat{H},\hat{h})$. Consequently, $x^f(H^I,h^I)$ has at most m non-integral entries. Therefore, we lose at most m choices of items while rounding down $x^f(H^I,h^I)$. We lose at most $\frac{|H^I|}{q}$ for each choice due to rounding. Let x^* and z^* be an optimal (integral) solution and an optimal value to the *DRAP*, respectively.

$$z^* = \sum_{i=1}^{n} \sum_{j=1}^{|R_i|} x_{ij}^* \tag{14}$$

From (11) and (14), we conclude that

$$\hat{z} \geq z^f - m.\frac{|H|}{q} \geq z^f\left(1 - \frac{m}{q}\right) \tag{15}$$

$$z^f\left(1 - \frac{m}{q}\right) \geq z^*\left(1 - \frac{m}{q}\right) \tag{16}$$

Algorithm 1: ϵ-Approximation Scheme for DRAP

```
1  begin
2      ẑ = 0
3      for all H ⊆ F with |H| ≤ q do
4          for every h:H → R do
5              for k = 1, 2, ..., m do
6                  // Compute residual capacity
                       b_k(H,h) = L_k − Σ_{i∈H} v_{ih(i)k};
7              end
8              Feasible=True;
9              for k = 1, 2, ..., m do
10                 if (b_k(H,h) < 0) then
11                     Feasible=False;
12                 end
13             end
14             if (Feasible==True) then
15                 Solve DRAP^f(H,h);
16                 Let x^f(H,h) be the solution of DRAP_lp(H,h);
17                 Round x^f(H,h) to an integer solution;
18                 for i = 1, 2, ..., n do
19                     for j = 1, 2, ..., |R_i| do
20                         x^I_ij(H,h) = ⌊x^f_ij(H,h)⌋;
21                     end
22                 end
23                 Compute z^I(H,h) = Σ_{i=1}^{n} Σ_{j=1}^{|R_i|} x^I(H,h);
24                 if (ẑ < z^I(H,h)) then
25                     ẑ = z^I(H,h);
26                     x̂ = x^I(H,h);
27                 end
28             end
29         end
30     end
31 end
```

From (15) and (16), we conclude that

$$\hat{z} \geq z^*\left(1 - \frac{m}{q}\right) \tag{17}$$

$$z^* - \hat{z} \leq \epsilon.z^*, \epsilon = \frac{m}{q} \tag{18}$$

Therefore, Algorithm 1 is an ϵ-approximation algorithm. The outermost *for loop* of the approximation scheme (Algorithm 1) considers all possible subsets H of F whose size does not exceed q and the possible subsets of size at most q are $O(n^q)$. The second *for loop* considers all possible mappings $h : H \rightarrow R$ and the possible mappings h for a given H are $O(|R|^q)$. Hence, Algorithm 1 runs for $O((n|R|)^q) = O((n|R|)^{m/\epsilon})$ iterations. The approximation scheme is polynomial in input length for a fixed number of resource types m with ϵ approximation factor.

V. NFVPermit: An Adaptive Resource Allocation Scheme

Since the proposed ILP model is NP-hard, it cannot scale well especially when the problem size increases. The proposed approximation scheme for DRAP is polynomial with input length, *i.e.,* runs for $O((n|R|)^{m/\epsilon})$ iterations. However, for a fixed number of resource types m with ϵ approximation factor, $O((n|R|)^{m/\epsilon})$ results in a higher order polynomial. Hence, we propose a heuristic algorithm for DRAP namely *NFVPermit* whose running time is quadratic with the inputs. As the input traffic rate of VNF changes over time, the required

Algorithm 2: NFVPermit Scheme

```
Input: : Resource allocation matrix: C;
         Ingress traffic rate of VNFs: λ_i, ∀ i ∈ F;
         Available resources: L_k, ∀ k ∈ m;
Output:: Allocation of system resources to VNFs;
1  begin
2    for t = 1, 2, ..., T do
3      for i = 1, 2, ..., n do
4        // Measure input traffic rate of each VNF
           λ_i^(t) = findTraffic();
5      end
6      G= Compute required resources using profiled data
         based on λ^(t) = {λ_1^(t), λ_2^(t), ..., λ_|F|^(t)}
7      for i = 1, 2, ..., n do
8        for j = 1, 2, ..., |R_i| do
9          for k = 1, 2, ..., m do
10           v_ijk ← G_ijk - C_ik;
11         end
12       end
13     end
14     for i = 1, 2, ..., n do
15       for j = 1, 2, ..., |R_i| do
16         for k = 1, 2, ..., m do
17           if (v_ijk < 0) then
18             Rel.add(i);
19           else if (v_ijk > 0) then
20             Acq.add(i);
21           else if (v_ijk = 0) then
22             noChange;
23           else
24             Rest.add(i);
25           end
26         end
27       end
28     end
29     AFraction_k ← 0, ∀k = {1, 2, ..., m}
30     for i = 1, 2, ..., |Acq| do
31       for j = 1, 2, ..., |R_i| do
32         for k = 1, 2, ..., m do
33           AFraction_k ← AFraction_k + v_ijk / S_k;
34         end
35       end
36     end
37     Sort AFraction in descending order;
38     AcqRank ← Assign ranks to the resources based on
         AFraction;
39     release(Rel, AcqRank);   (Refer Algorithm 3)
40     Sort VNFs in Acquire category in non-descending order based
         on the least available system resource
41     acquire(Acq);   (Refer Algorithm 4)
42     mixed(Rest);   (Refer Algorithm 5)
43     Distribute remaining resources to the background or
         best-effort services running on the server
44   end
45 end
```

system resources for those VNFs also need to be reconfigured quickly to meet their SLA requirements. The goal of *NFVPermit* is to determine and allocate the required system resources (i.e., LLC and MB) among the co-located VNFs running in the server at each time interval for improving the overall resource utilization while ensuring performance isolation to the co-located VNFs.

The procedure followed by *NFVPermit* is given in Algorithm 2. Ingress traffic rate of each VNF is measured using *findTraffic()* procedure with the help of *VNF monitor* module. Based on the ingress traffic rate of VNFs at time interval $t \in [1, T]$, step 6 computes the required resources from the profiling table and stores them in matrix *G*, where *T* is the total number of time intervals. The resources allocated to the currently running VNFs are stored in matrix *C*. We compute the demand *v* in steps 7-13, which stores the deficit or surplus resource information of each VNF in the current interval. If the value of v_{ijk} is positive, then more resources are needed than are currently allocated. If v_{ijk} is negative, then some of the allocated resources can be released. Based on v_{ijk}, VNFs can be grouped into four categories as mentioned in steps 14-28.

- *Release:* VNFs which need fewer LLC ways and/or MB than their current allocations in a particular time interval are classified into *Release* category. Such VNFs can donate extra resources to the resource pool.
- *Acquire:* VNFs which need more LLC ways and/or MB than their current allocations are considered in the *Acquire* category. Such VNFs can claim extra resources from the resource pool.
- *Mixed:* VNFs acting as *Release* for one resource type and as *Acquire* for other resource type come under the *Mixed* category.
- *None:* VNFs which neither require any additional resources nor donate any of their current resources are categorized as *None*.

After categorizing each VNF into one of the aforementioned categories, resources need to be allocated to each VNF with the goal of maximizing the SLA guaranteed VNFs. To accomplish this, steps 29-38 compute the demand of each resource while considering VNFs in *Acquire* category and ranking the resource types. Then, it releases resources from VNFs in the *Release* category, taking *rank* into account as described in Algorithm 3. The available resources are then assigned to VNFs in the *Acquire* and *Mixed* categories using Algorithm 4 and Algorithm 5, respectively. Consequently, if any free resources are available, they are allocated equally to background or best-effort services running on the same server.

Algorithm 3 describes the procedure for releasing resources from VNFs belonging to the *Release* category. Steps 3-11 consider VNFs in *Rel* category and compute the amount of resource that can be released. Let the ranking of resources based on the amount released be *RelRank*. The variable ch_{ij} is used to maintain the difference between *RelRank* and *AcqRank* (computed in Algorithm 2). Step 19 selects the choice with the lowest value of ch_{ij}. Then, the residual system resources and the resources allocated to the VNFs are adjusted in step 20 and step 21, respectively.

NFVPermit sorts the VNFs in the *Acquire* category in non-decreasing order based on the least available resource in the system. This improves the number of VNFs whose system resource requirements are met (which leads to guaranteeing SLA). The method to acquire resources for VNFs belonging to the *Acquire* category is described in Algorithm 4 which is based on the approach defined in [55]. Steps 3-10 compute the initial allocation of resources for each VNF in the *Acquire* category. This is done by first selecting a feasible choice from the list of choices available to the VNF. The feasibility of a choice is determined by calling the procedure *feasible*(). It checks all available choices for a VNF iteratively and returns

Algorithm 3: Release($Rel, AcqRank$)

```
1  begin
2      Rfraction_k ← 0, ∀k = {1, 2, ..., m}
3      for i = 1, 2, ..., |Rel| do
4          for j = 1, 2, ..., |R_i| do
5              for k = 1, 2, ..., m do
6                  Rfraction_k ← Rfraction_k + v_ijk / S_k;
7              end
8          end
9      end
10     Sort Rfraction in descending order;
11     RelRank_k = Rank resources based on Rfraction;
12     for i = 1, 2, ..., |Rel| do
13         for j = 1, 2, ..., |R_i| do
14             ch_ij ← 0;                                   ▷ choice
15             for k = 1, 2, ..., m do
16                 ch_ij ← ch_ij + RelRank_k − AcqRank_k;
17             end
18         end
19         j* ← arg min_j ch_ij;
20         L ← L − v_ij*;
21         C_i ← C_i + v_ij*;
22     end
23 end
```

Algorithm 4: Acquire(Acq)

```
1  begin
2      /* Initial allocation */
3      for i = 1, 2, ..., |Acq| do
4          for j = 1, 2, ..., |R_i| do
5              if (feasible(i, j) == 1) then
6                  allocate(i, j);
7                  break;
8              end
9          end
10     end
11     /* Checking for better allocation */
12     while true do
13         Δr_max ← 0;
14         for i = 1, 2, ..., |Acq| do
15             if pick_i > 0 then
16                 for j = 1, 2, ..., |R_i| do
17                     if (pick_i != j and feasible(i, j) == 1) then
18                         Δr = (v_ipick_i − v_ij) * Current / ||Current||;
19                         if (Δr > Δr_max) then
20                             Δr_max ← Δr;
21                             i^1 = i; j^1 = j;
22                         end
23                     end
24                 end
25             else
26                 for j = 1, 2, ..., |R_i| do
27                     if (feasible(i, j) == 1) then
28                         allocate(i, j);
29                         i ← i − 1;
30                         break;
31                     end
32                 end
33             end
34         end
35         if (Δr_max ≤ 0) then
36             Return;
37         end
38         L ← L − v_i^1 pick_i^1 + v_i^1 j^1;
39         Current ← Current + v_i^1 j^1 − v_i^1 pick_i^1;
40         C_i^1 ← C_i^1 + v_i^1 j^1 − v_i^1 pick_i^1;
41         pick_i^1 ← j^1;
42     end
43 end

44 begin
45     procedure allocate(i,j)
46     L ← L − v_ij;
47     C_i ← C_i + v_ij;
48     Current ← Current + v_ij;
49     pick_i ← j;
50     end procedure
51 end

52 begin
53     procedure feasible(i,j)
54     for k = 1, 2, ..., m do
55         if (v_ijk > L_k) then
56             Return 0;
57         end
58     end
59     Return 1;
60     end procedure
61 end
```

the first feasible choice. That is, if a choice is not feasible, then it checks for feasibility of next entry from the list of choices. Note that a choice is feasible if each resource type's demand does not exceed the corresponding residual resources. After finding the feasible choice of a VNF, the residual and allocated resources are updated by calling *allocate*() procedure. It also maintains a variable *Current*, which is used to keep track of the sum of allocated resources for the VNFs in *Acquire* category. Further, for each VNF $i \in Acquire$ a variable *pick* is used to keep track of choice selected in the initial allocation. Note that, $pick_i$ is zero if resources are not allocated to the VNF i in the initial allocation.

After selecting initial entries and the initial allocation of resources to VNFs, steps 12-42 select better choices for the VNFs with *pick* greater than zero such that the saved resources can be used to satisfy the requirements of VNFs with *pick* equal to zero. For a VNF *i* with $pick_i > 0$ and a choice *j* with $pick_i \neq j$, let Δr be the aggregate resource difference between the existing entry, $pick_i$, and the new entry *j*. Step 13 initializes the maximum aggregate resource saving Δr_{max} to zero. For VNFs with $pick$ greater than zero, steps 16-24 iterate over feasible entries other than the current entry and compute the VNF and its choice with maximum Δr. Further, steps 38-41 update current choice, residual and allocated resources to VNF with maximum Δr. For VNFs with *pick* equal to zero, steps 26-32 recheck feasibility of entries and allocate resources if there is a feasible choice. Then, the process of selecting a better entry is repeated until deviating from the existing choice of VNFs does not result in saving resources, i.e., $\Delta r_{max} < 0$.

For better understanding, we illustrate *acquire* process by taking an example. We assume three VNFs are in the *Acquire* category. Allocated and demand of resources to each VNF, as well as available resources, are shown in Table III. In the first iteration, we pick the initial feasible demand (3, 10) and (1, 10) for VNF_1 and VNF_2, respectively. The free resources available right now are (1, 40). VNF_3 is not able to get the resources in the first iteration. In order to guarantee the SLA for more VNFs, we iteratively replace the initially picked choices with other choices as long as the solution remains

TABLE III
AN EXAMPLE SCENARIO TO ILLUSTRATE *acquire* PROCESS

Available free resources (L_k): (5,60)

	VNF_1	VNF_2	VNF_3
Allocated	(2,20)	(0,0)	(2,10)
Requirement	(5,30) or (4,40) or (3,50)	(1,10)	(6,20) or (4,30)
Demand	(3,10) or (2,20) or (1,30)	(1,10)	(4,10) or (2,20)

Algorithm 5: Mixed($Rest$)

```
1  begin
2      for i = 1, 2, ..., |Rest| do
3          for j = 1, 2, ..., |R_i| do
4              ch_ij ← 0;                                   ▷ choice
5              for k = 1, 2, ..., m do
6                  if (v_ijk < L_k) then
7                      ch_ij ← ch_ij − v_ijk / S_k;
8                  else
9                      break;                    ▷ Do not consider choice
10                 end
11             end
12         end
13         j* ← arg max_j ch_ij;
14         L ← L − v_ij*;                  ▷ Change resources accordingly
15         C_i ← C_i + v_ij*;
16     end
17 end
```

feasible. In the second iteration, we check for other choices for VNF_1 and calculate Δr. Δr is calculated using *Current* variable and difference between the already allocated choice and the new choice in the current iteration. Δr value for demand (2, 20) of VNF_1 is 0.009 and for another demand (1, 30) its value is 0.019 (both are greater than 0). (1, 30) is the demand for which maximum Δr value is chosen and the residual resources are (3, 20). Now, VNF_3 is also able to get the resources needed. The feasible demand (2, 20) is chosen and the residual resources are (1, 0). This process leads to allocation of sufficient resources to the third VNF also.

Algorithm 5 describes the procedure for accommodating VNFs which can both acquire and release resources. For a VNF $i \in Mixed$ and one of its choices j, the variable ch_{ij} is used to maintain the difference between fraction of released and acquired resource. For each VNF, steps 3-12 compute ch_{ij} for all choices. Step 13 selects the choice with the maximum value of ch_{ij}. Then, the residual system resources and resources allocated to VNFs are adjusted in step 14 and step 15, respectively.

If any of the VNFs in the *Release* and *Mixed* categories run out of resources, there is a need for VNF migration to another NUMA node on the same server or to another server altogether, which is beyond the scope of this article.

Theorem 2: Time complexity of NFVPermit is $O((np)^2m + mlogm)$.

Steps 3-25 of Algorithm 2 compute the required resources of each VNF from the lookup table based on its ingress traffic rate and categorize it into one of the four categories. Steps 29-38 enumerate all VNFs from acquire list and assign ranks to resources based on the demand. The time complexity of this whole process is $O(npm + mlogm)$, where n is the number of VNFs, p is the maximum number of choices for any VNF, and m is the number of resource types.

Now we briefly discuss the time complexity of the procedures invoked by Algorithm 2. The *release*() procedure (Algorithm 3) also assigns ranks to resources, sorts them, and releases them to the free pool. It also takes $O(npm + mlogm)$ time. Following that, we call *acquire*() procedure to allocate resources to VNFs from the resource pool. In steps 12-43 of Algorithm 4, we are checking for choices that might be a better pick and at most np number of upgrades. Since each iteration takes O(npm) time, the overall time complexity of Algorithm 4 is O($(np)^2m$). Next, we allocate resources for VNFs in the mixed category by calling Algorithm 5 whose time complexity is O(npm). Therefore, the time complexity of NFVPermit heuristic is $O((np)^2m + mlogm)$.

VI. PERFORMANCE EVALUATION

In this section, we evaluate the performance of *NFVPermit* by implementing it on a prototype system using OpenNetVM, a high-performance container-based platform for NFV with necessary changes. Since the system configuration used to perform real-time experiments is limited to 20 *Gbps* traffic, we also conducted extensive simulation experiments to demonstrate the efficacy of *NFVPermit* when running more number of VNFs. VNFs are selected from OpenetVM and each VNF receives ingress traffic for 120 *seconds*. To simulate a time-varying traffic demand, the ingress traffic rate of each VNF is taken from the real traces available from [56] and normalized to 10 *Gbps*. All VNFs are assumed to run on the local NUMA node (i.e., NICs are directly connected to it) of the server. In the servers used in experimentation (Intel Xeon Gold 6126 CPU@2.60 GHz), there are 11 LLC ways and MB levels could be varied from 10% to 90% in steps of 10% [17], [57]. Simulation experiments are carried out on a Linux machine with four 3.5 *GHz* CPU cores and 16 *GB* memory. For simulation experiments, we have generated synthetic data of resource requirements for each VNF based on the profiling data collected in Section IV-A2. In all experiments, we used certain traffic rate distribution and thus we know the actual (i.e., future) traffic rate of each VNF at each time interval in advance. The VNF monitor has been configured with a one-second time interval for this study.

A. Comparison Schemes

Proposed *NFVPermit* is compared with the following state-of-the-art and baseline schemes:

- *Static:* In this scheme, resources will be assigned to VNFs based on the peak traffic rate. Maximum amount of resources are allocated irrespective of the input traffic rate.
- *ResQ [11]:* Depending upon the incoming traffic rate to the VNFs, this scheme allocates minimum number of LLC ways to them without considering MB allocation.
- *LLC-greedy (ResQ with MB consideration):* This scheme allocates resource combinations having least number of

LLC ways to the VNFs in correspondence to their traffic rates. Unlike ResQ, this scheme allocates both LLC ways and MB to the VNFs.

- *MB-greedy:* In this scheme, a resource combination entry that takes least amount of MB gets allocated to a VNF. Along with MB, this scheme also allocates certain LLC ways.
- *RESTRAIN [47]:* In this scheme, each VNF greedily chooses a resource combination entry based on the scarcity of available free resources until the system has enough remaining resources to accommodate. For example, consider the scenario where out of 11 LLC ways in the system, suppose 40% of LLC ways and 60% of MB are currently available. It considers LLC as the resource under scarcity, and while allocating resources for, it chooses the allocation which has the least LLC requirement.

The resource allocation schemes *static* and *MB-greedy* are baseline schemes, while *ResQ* and *RESTRAIN* are the state-of-the-art schemes. We also consider *LLC-greedy* which is a variation of ResQ to compare the performance of *NFVPermit*.

Performance metrics: The following performance metrics are used:

1) *Number of SLA guaranteed VNFs over time:* This metric represents the number of SLA guaranteed VNFs for each time instance. A VNF is said to be SLA guaranteed when it has given sufficient resources based on its ingress traffic rate.
2) *Average Normalized SLA guaranteed VNFs (ANSV):* Normalized SLA guaranteed VNFs (NSV) is defined as the ratio of the number of SLA guaranteed VNFs over the total number of VNFs running. NSV is calculated for each time interval. The average of NSV for all time instances is termed as Average NSV (ANSV).
3) *Average resource utilization:* It represents the ratio of the average percentage of resources (i.e., LLC and MB) allocated to the running VNFs over the total resources.
4) *Execution time (measured in CPU time):* We use time elapsed to obtain resource allocations for a given set of VNFs for comparing the running time of the different schemes.

B. Simulation Results

We consider 30 VNFs which are randomly picked from the profiled VNFs listed in Table I. We consider six homogeneous servers, with five randomly selected VNFs deployed on each of them. To simulate time-varying traffic demands, the ingress traffic rate of each VNF is taken from the real traces available from [56] and normalized to a traffic rate based on its maximum achievable throughput. Fig. 8 shows variation in ANSV for different schemes with 95% confidence intervals. As plotted, *NFVPermit* achieves approximately 43%, 14%, 25%, and 8% more ANSV than that of *static*, *LLC-greedy*, *MB-greedy*, and *RESTRAIN* schemes, respectively. The *static* scheme allocates resources in advance based on peak traffic rates of VNFs. As a result, it allocates resources to the least number of VNFs. Greedy schemes, such as *LLC-greedy* and *MB-greedy*, make

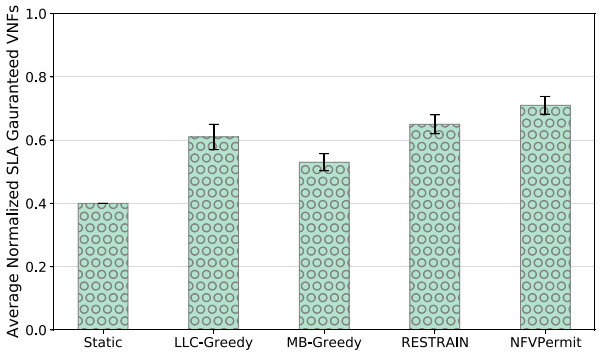


Fig. 8. Variation in ANSV for various schemes.

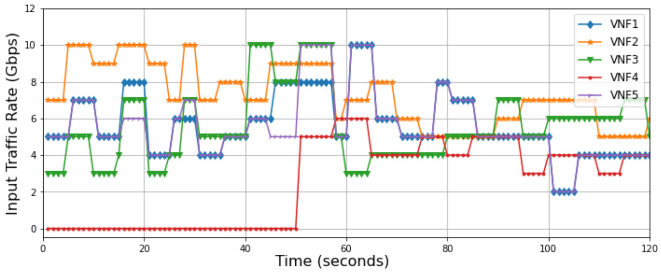


Fig. 9. Variation in traffic demands of five VNFs over time.

choices by acting greedily on conserving one resource, causing another resource to be depleted. *RESTRAIN*, on the other hand, assigns resource combinations based on the scarcity of free resources. Therefore, these schemes result in SLA guarantees for fewer VNFs even though another resource type is sufficiently available. Unlike greedy schemes, the proposed *NFVPermit* balances LLC and MB resources, thereby meeting SLA for more number of VNFs. When sufficient resources are not available for allocating to a VNF, none of the schemes guarantee SLA for that particular VNF.

To get better insights, we present how the resource allocation and the number of SLA guaranteed VNFs vary over time based on traffic rates of different VNFs running on a server using different mechanisms. Let us consider, a server running one instance of Router (VNF1), Flow Tracker (VNF2), Basic Monitor (VNF3), and two instances of Simple Forward (VNF4 and VNF5) VNFs for a time duration of 120 *seconds*. We assume that the traffic of one instance of Simple Forward VNF starts from the time interval 50, while the remaining VNFs' traffic begins at time interval 1. Fig. 9 depicts the variation in traffic demands of each VNF. Results obtained for this scenario are shown in Figs. 10-14.

1) Average Resource Utilization: Fig. 10 variation in average resource utilization for different schemes. The *static* scheme allocates resources to only two VNFs as these VNFs consume most of the resources based on their peak traffic rates. The remaining resources are insufficient to assign to other VNFs. Except when the ingress traffic rate of VNFs reaches its peak traffic, the resources allocated in the *static* scheme are underutilized at all other times. *LLC-greedy* and *MB-greedy* methods allocate resources greedily on one resource, leading to increased consumption of the other resource. Unlike greedy

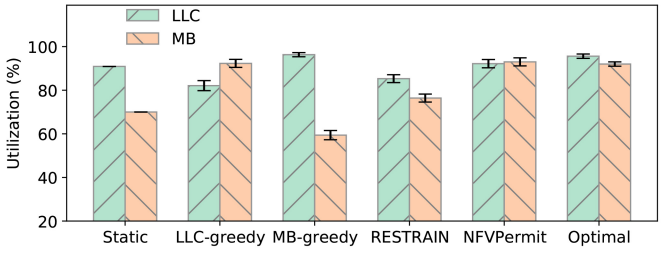


Fig. 10. Resource utilization for various schemes.

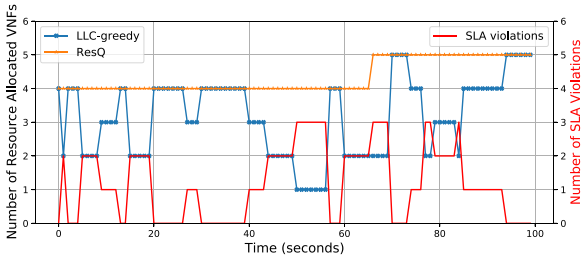


Fig. 11. Number of VNFs to which resources are allocated by ResQ and LLC-greedy and number of SLA violations in ResQ over time.

and *RESTRAIN* schemes, the proposed *NFVPermit* balances LLC and MB resources among the VNFs, allowing for a higher number of VNFs to meet their SLAs.

2) ResQ vs LLC-Greedy: ResQ scheme allocates LLC ways to VNFs greedily without considering MB allocation. In doing so, the total MB requirements of all the VNFs to satisfy their SLAs exceed the maximum limit of the system, i.e., 100%. If a VNF does not get the required resource combination, we consider the SLA of that VNF to be violated. To calculate the number of SLA violations incurred for *ResQ*, we remove the VNFs one by one until the total MB requirement of existing VNFs does not exceed the maximum limit, such that the number of SLA violations is minimized. Fig. 11 shows a combined plot of the number of VNFs to which resources are allocated (Y1 axis) by *ResQ* and *LLC-greedy* schemes and the number of VNFs that experience SLA violations (Y2 axis) by *ResQ* scheme over time. When *LLC-greedy* is used, the number of SLA guaranteed VNFs is the same as the number of VNFs to which *ResQ* allocated the resources subtracted by the number of VNFs which experienced SLA violations from those, therefore we did not present the plots of *ResQ* separately in this article.

3) Number of SLA Guaranteed VNFs: Fig. 12 depicts the number of SLA guaranteed VNFs over time for different schemes including the optimal solution, i.e., DRAP given in Eqn. (1). If a VNF gets sufficient resources based on its ingress rate then it is assumed that its SLA will be guaranteed for that particular time interval. It is worth noting that the proposed *NFVPermit* approach guarantees SLAs for either more or the same number of VNFs as other schemes except the optimal solution in all time instances. Further, it is to be noted that the performance of *NFVPermit* scheme is almost close to that of the optimal for most of the time intervals. The number of SLA guaranteed VNFs in the static scheme is always two because it allocates resources in advance based on the peak traffic rates of VNFs. Greedy schemes, such as *LLC-greedy* and *MB-greedy* make choices by acting greedily on conserving one resource, causing another resource to be depleted. Therefore, these schemes result in SLA guarantees for fewer VNFs even though another resource type is available. For most of the time intervals, the proposed *NFVPermit* approach outperforms LLC-greedy, MB-greedy, and *RESTRAIN* schemes, while the performance of LLC-greedy is better than that of MB-greedy for most of the time intervals.

4) Average Normalized SLA Guaranteed VNFs: We also calculated the ANSV for different schemes and reported them in Fig. 13. All the baseline schemes are either greedy on one resource type or based on peak traffic, therefore they provide guaranteed SLA only for a fewer number of VNFs. The proposed *NFVPermit* scheme accepts 49%, 16%, 28%, and 9% more ANSV than *static*, *LLC-greedy*, *MB-greedy*, and *RESTRAIN* schemes, respectively. It is worth noting that *NFVPermit* achieves results very close to the optimal results (approximately 5% difference).

Fig. 14 shows the percentage of remaining resources after allocating the required amount to the running VNFs. These surplus resources could be assigned to the background or other services co-located with the VNFs. Greedy schemes make choices by acting greedily on one resource, causing another resource to be depleted as shown in Fig. 14. Unlike greedy schemes, the proposed *NFVPermit* scheme balances the resource utilization which leads to guaranteeing SLAs for more number of VNFs. From these results, we conclude that the proposed *NFVPermit* scheme is more effective in improving resource allocation while also guaranteeing performance isolation among the co-located services deployed in NFV-based systems.

5) Execution Time: Fig. 15 shows the average execution times of different resource allocation schemes. It is worth highlighting that, *NFVPermit* and the optimal (ILP model) solution differ drastically in terms of execution times incurred. The average execution time of the optimal solution increases exponentially with the number of VNFs. But the proposed *NFVPermit* provides an efficient and feasible solution in very less time therefore it is quite apt for real-world deployments with almost the same performance as that of the optimal solution. In terms of efficiency, as shown in Fig. 15, the execution time of *NFVPermit* increases slightly as we increase the number of VNFs from three to seven. The time taken by the baseline and state-of-the-art schemes is also close to that of the proposed *NFVPermit* scheme.

In summary, these results show that the proposed *NFVPermit* offers better performance with efficient resource utilization over other schemes at low computational cost even when we need to deploy higher number of VNFs on a commodity server.

C. Testbed Results

Since the system configuration (described earlier in Section III-B) used to perform real-time experiments is limited to 20 *Gbps* traffic, we run two VNFs namely Simple Forward (SF) and Firewall (FW) which receive traffic ranging from

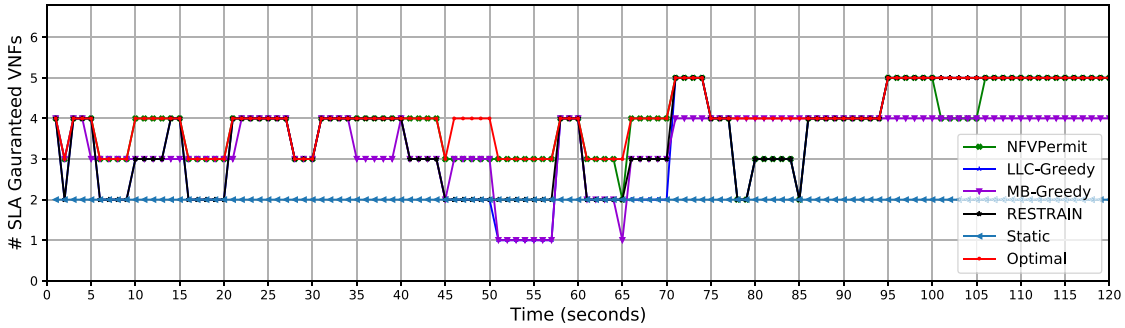


Fig. 12. Variation in SLA guaranteed VNFs in case of single server scenario for various schemes over time.

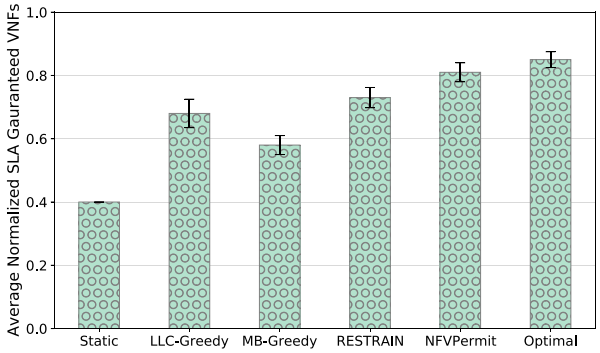


Fig. 13. Variation in ANSV in case of single server scenario for various schemes.

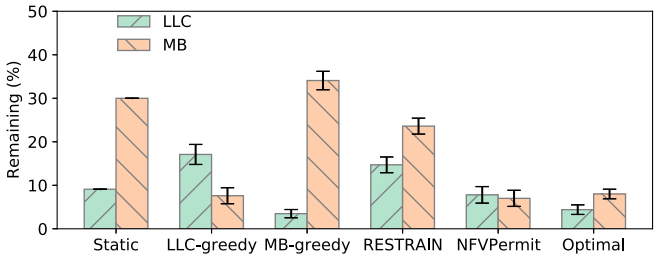


Fig. 14. Percentage of remaining resources for various schemes.

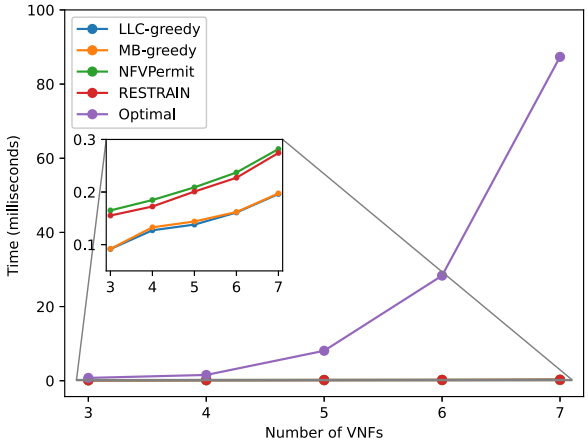


Fig. 15. Comparison of execution times of different schemes.

1 *Gbps* to 10 *Gbps* independently. We consider the scenario where the total available resource pool is (5, 50). The resources are allocated for each VNF based on its ingress traffic rate

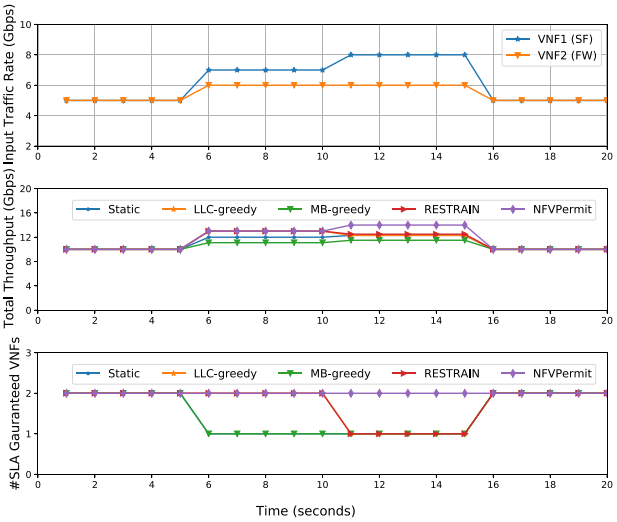


Fig. 16. When two VNFs are running, we depict how the total achieved throughput and the number of SLA guaranteed VNFs change over time as the ingress traffic rate of the VNFs changes in the experimental testbed scenario.

by referring to the profiling data given in Table III. Fig. 16 depicts the number of SLA guaranteed VNFs over time for different schemes. The top plot reflects the ingress traffic rate of each VNF over time. The total aggregate throughput achieved and the number of SLA guaranteed VNFs are depicted in the middle and bottom plots, respectively. The static scheme allocates resources (3, 30) to Simple Forward VNF based on its peak traffic rate (i.e., 8 *Gbps*), while the remaining resources (2, 20) are allocated to the other VNF running in the same system. As a result, it guarantees SLA for only one VNF most of the time. Greedy schemes, such as *LLC-greedy* and *MB-greedy* result in SLA guarantees for a single VNF most of the time since they act greedily on conserving one resource. All schemes, including *RESTRAIN*, guarantee SLA for a single VNF, especially from 11 to 15 seconds whereas *NFVPermit* guarantees SLA for both the VNFs throughout the experimental run due to dynamic balancing of the system resources LLC ways and MB efficiently. The proposed *NFVPermit* scheme achieves either equal or higher total aggregate throughput than other schemes.

D. Discussion

The proposed *NFVPermit* approach relies on profiling that characterizes the VNFs. Profiling helps the network service providers to determine the optimal combinations of system

resources to guarantee SLAs for different traffic loads in advance and therefore it does not suffer from SLA violation issues time-to-time. Moreover, we target services that require performance guarantees, which is a very essential requirement for VNFs. We feel that the task of profiling is not an overhead as it takes place only once for each VNF in offline. VNF profiling has limitations like it has to be performed for each update of the target VNF and hardware-level changes to the underlying commodity server. But, hardware-level changes are sporadic in data centers, and even software updates to the VNFs happen less frequently. However, by adopting the methodology given in [53], the time required for profiling could be reduced significantly.

Leveraging Machine Learning (ML) techniques for dynamically allocating resources like LLC and MB to the target VNFs without profiling is an interesting research direction. Reinforcement Learning (RL) could be suitable for this type of problem, which is beyond the scope of this article.

VII. Conclusion and Future Directions

Ensuring performance isolation in an NFV environment is challenging when multiple VNFs share system resources, especially LLC and MB. Using Intel RDT tools, we demonstrated the necessity of dynamic LLC and MB to the competing VNFs. Further, we formulated the dynamic allocation of LLC and MB to co-located VNFs as an ILP to maximize the number of SLA guaranteed VNFs. Due to NP-hardness of the problem, we presented a polynomial-time approximation scheme by relaxing the integer constraints. As the approximation scheme results in a higher order polynomial, we also proposed *NFVPermit*, a lightweight heuristic *NFVPermit* which ensures performance isolation between co-located VNFs running on a server. Our evaluation results demonstrate the effective utilization of resources over baseline approaches. Through the simulations, we demonstrated that our proposal could efficiently improve the number of SLA guaranteed VNFs compared to the state-of-the-art and baseline approaches. *NFVPermit* has improved the performance compared to two state-of-the-art resource allocation schemes, LLC-greedy (modified version of *ResQ*) and *RESTRAIN*, by 16% and 9%, respectively.

Future directions include extending the proposed solution to support the migration of VNFs across NUMA nodes within the same server or across servers when the resources are not available in other NUMA nodes. We also like to use traffic predictions that can help in improving the proposed resource allocation algorithm based on future traffic. In addition, we would like to investigate ML-based solutions for dynamically allocating system resources like LLC and MB to the target VNFs.

References

[1] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 236–262, 1st Quart., 2016.

[2] "Network functions virtualization: An introduction, benefits, enablers, challenges and call for action," ETSI NFV ISG, Sophia Antipolis, France, Oct. 2012.

[3] "DPDK: Data plane development kit: Programmer's guide," Intel Corp., Santa Clara, CA, USA, Feb. 2017.

[4] "DDIO: Intel data direct I/O technology overview," Intel White Paper, Santa Clara, CA, USA, 2012.

[5] T. Kuo, B. Liou, K. C. Lin, and M. Tsai, "Deploying chains of virtual network functions: On the relation between link and server usage," *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1562–1576, Aug. 2018.

[6] Q. Zhang, Y. Xiao, F. Liu, J. C. S. Lui, J. Guo, and T. Wang, "Joint optimization of chain placement and request scheduling for network function virtualization," in *Proc. IEEE ICDCS*, 2017, pp. 731–741.

[7] Y. Sang, B. Ji, G. R. Gupta, X. Du, and L. Ye, "Provably efficient algorithms for joint placement and allocation of virtual network functions," in *Proc. IEEE INFOCOM*, 2017, pp. 1–9.

[8] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *Proc. IEEE INFOCOM*, 2015, pp. 1346–1354.

[9] P. Jin, X. Fei, Q. Zhang, F. Liu, and B. Li, "Latency-aware VNF chain deployment with efficient resource reuse at network edge," in *Proc. IEEE INFOCOM*, 2020, pp. 267–276.

[10] C. Zeng, F. Liu, S. Chen, W. Jiang, and M. Li, "Demystifying the performance interference of co-located virtual network functions," in *Proc. IEEE Conf. INFOCOM*, 2018, pp. 765–773.

[11] A. Tootoonchian et al., "ResQ: Enabling SLOs in network function virtualization," in *Proc. USENIX NSDI*, 2018, pp. 283–297.

[12] J. Nam, J. Seo, and S. Shin, "Probius: Automated approach for VNF and service chain analysis in software-defined NFV," in *Proc. ACM SOSR*, 2018, p. 14.

[13] R. S. Kannan, A. Jain, M. A. Laurenzano, L. Tang, and J. Mars, "Proctor: Detecting and investigating interference in shared datacenters," in *Proc. IEEE ISPASS*, 2018, pp. 76–86.

[14] P. Veitch, E. Curley, and T. Kantecki, "Performance evaluation of cache allocation technology for NFV noisy neighbor mitigation," in *Proc. IEEE NetSoft*, 2017, pp. 1–5.

[15] N. Schramm, T. M. Runge, and B. E. Wolfinger, "The impact of cache partitioning on software-based packet processing," in *Proc. IEEE NetSys*, 2019, pp. 1–6.

[16] "Introduction to cache allocation technology in the Intel Xeon processor E5 v4 family," Intel Corp., Santa Clara, CA, USA, 2016.

[17] Y. Xiang, C. Ye, X. Wang, Y. Luo, and Z. Wang, "EMBA: Efficient memory bandwidth allocation to improve performance on intel commodity processor," in *Proc. ACM ICPP*, 2019, p. 16.

[18] W. Zhang et al., "OpenNetVM: A platform for high performance network service chains," in *Proc. ACM HotMiddlebox*, 2016, pp. 26–31.

[19] S. Palkar et al., "E2: A framework for NFV applications," in *Proc. ACM SOSP*, 2015, pp. 121–136.

[20] A. Panda, S. Han, K. Jang, M. Walls, S. Ratnasamy, and S. Shenker, "NetBricks: Taking the V out of NFV," in *Proc. USENIX OSDI*, 2016, pp. 203–216.

[21] S. G. Kulkarni et al., "NFVnice: Dynamic Backpressure and scheduling for NFV service chains," in *Proc. ACM SIGCOMM*, 2017, pp. 71–84.

[22] J. Martins et al., "ClickOS and the art of network function virtualization," in *Proc. USENIX NSDI*, 2014, pp. 459–473.

[23] C. Sun, J. Bi, Z. Zheng, H. Yu, and H. Hu, "NFP: Enabling network function parallelism in NFV," in *Proc. ACM SIGCOMM*, 2017, pp. 43–56.

[24] Q. Zhang, F. Liu, and C. Zeng, "Online adaptive interference-aware VNF deployment and migration for 5G network slice," *IEEE/ACM Trans. Netw.*, vol. 29, no. 5, pp. 2115–2128, Oct. 2021.

[25] Y. Mu, L. Wang, and J. Zhao, "Energy-efficient and interference-aware VNF placement with deep reinforcement learning," in *Proc. IEEE IFIP Netw.*, 2021, pp. 1–9.

[26] A. Manousis, R. A. Sharma, V. Sekar, and J. Sherry, "Contention-aware performance prediction for virtualized network functions," in *Proc. ACM SIGCOMM*, 2020, pp. 270–282.

[27] J. Li, Y. Qi, W. Wei, J. Lin, M. Wozniak, and R. Damasevicius, "dCCPI-predictor: A state-aware approach for effectively predicting cross-core performance interference," *Future Gener. Comput. Syst.*, vol. 105, no. 1, pp. 184–195, Apr. 2020.

[28] X. Xu, N. Zhang, M. Cui, M. He, and R. Surana, "Characterization and prediction of performance interference on mediated passthrough gpus for interference-aware scheduler," in *Proc. USENIX HotCloud*, 2019, p. 14.

[29] Y. Li et al., "GAugur: Quantifying performance interference of colocated games for improving resource utilization in cloud gaming," in *Proc. ACM HPDC*, 2019, pp. 231–242.

[30] F. Romero and C. Delimitrou, "Mage: Online and interference-aware scheduling for multi-scale heterogeneous systems," in *Proc. ACM PACT*, 2018, p. 19.

[31] M. Dobrescu, K. Argyraki, and S. Ratnasamy, "Toward predictable performance in software packet-processing platforms," in *Proc. USENIX NSDI*, 2012, p. 11.
[32] 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 *Proc. IEEE/ACM MICRO*, 2011, pp. 248–259.
[33] S. Javadi and A. Gandhi, "User-centric interference-aware load balancing for cloud-deployed applications," *IEEE Trans. Cloud Comput.*, vol. 10, no, 1, pp. 736–748, Jan.-Mar. 2022.
[34] M. Savić, M. Ljubojević, and S. Gajin, "A novel approach to client-side monitoring of shared infrastructures," *IEEE Access*, vol. 8, pp. 44175–44189, 2020.
[35] J. Mukherjee, D. Krishnamurthy, and J. Rolia, "Resource contention detection in virtualized environments," *IEEE Trans. Netw. Service Manag.*, vol. 12, no. 2, pp. 217–231, Jun. 2015.
[36] J. Mukherjee, D. Krishnamurthy, and M. Wang, "Subscriber-driven interference detection for cloud-based Web services," *IEEE Trans. Netw. Service Manag.*, vol. 14, no. 1, pp. 48–62, Mar. 2017.
[37] J. Mukherjee and D. Krishnamurthy, "PRIMA: Subscriber-driven interference mitigation for cloud services," *IEEE Trans. Netw. Service Manag.*, vol. 17, no. 2, pp. 958–971, Jun. 2020.
[38] H. Yu et al., "Octans: Optimal placement of service function chains in many-core systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 9, pp. 2202–2215, Sep. 2021.
[39] Y. Zhong et al., "BPA: The optimal placement of interdependent VNFs in many-core system," in *Proc. Int. Conf. Collaborative Comput. Netw. Appl. Worksharing*, 2020, pp. 305–319.
[40] G. Papathanail, A. Pentelas, and P. Papadimitriou, "Towards fine-grained resource allocation in NFV infrastructures," in *Proc. GLOBECOM*, 2021, pp. 1–6.
[41] V. R. Chintapalli, S. B. Korrapati, B. R. Tamma, and A. Franklin A, "NUMASFP: NUMA-aware dynamic service function chain placement in multi-core servers," in *Proc. COMSNETS*, 2022, pp. 181–189.
[42] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis, "Heracles: Improving resource efficiency at scale," in *Proc. ACM ISCA*, 2015, pp. 450–462.
[43] C. Xu, K. Rajamani, A. Ferreira, W. Felter, J. Rubio, and Y. Li, "DCat: Dynamic cache management for efficient, performance-sensitive infrastructure-as-a-service," in *Proc. ACM EuroSys*, 2018, p. 14.
[44] K. Nikas, N. Papadopoulou, D. Giantsidi, V. Karakostas, G. Goumas, and N. Koziris, "DICER: Diligent cache partitioning for efficient workload consolidation," in *Proc. ACM ICPP*, 2019, p. 15.
[45] S. Chen, C. Delimitrou, and J. F. Martínez, "PARTIES: QoS-aware resource partitioning for multiple interactive services," in *Proc. ACM ASPLOS*, 2019, pp. 107–120.
[46] J. Fried, Z. Ruan, A. Ousterhout, and A. Belay, "Caladan: Mitigating interference at microsecond timescales," in *Proc. USENIX OSDI*, 2020, pp. 281–297.
[47] V. R. Chintapalli, M. Adeppady, B. R. Tamma, and A. Franklin A., "RESTRAIN: A dynamic and cost-efficient resource management scheme for addressing performance interference in NFV-based systems," *J. Netw. Comput. Appl.*, vol. 201, May 2022, Art. no. 103312.
[48] "Stress-ng." Accessed: Mar. 10, 2022. [Online]. Available: https://wiki.ubuntu.com/Kernel/Reference/stress-ng
[49] I. GitHub. "Pktgen-traffic generator powered by DPDK." 2019. [Online]. Available: https://github.com/Pktgen/Pktgen-DPDK/
[50] "Zero-touch network and service management (ZSM); reference architecture," ETSI GS ZSM 002, Sophia Antipolis, France, vol. V1.1.1, 2019.
[51] P. Zheng, W. Feng, A. Narayanan, and Z.-L. Zhang, "NFV performance profiling on multi-core servers," in *Proc. IFIP Netw. Conf. (Netw.)*, 2020, pp. 91–99.
[52] I. GitHub. "The system statistics collection deamon." 2021. [Online]. Available: https://collectd.org/
[53] M. G. Khan et al., "A performance Modelling approach for SLA-aware resource recommendation in cloud native network functions," in *Proc. IEEE NetSoft*, 2020, pp. 292–300.
[54] M. M. Akbar, M. S. Rahman, M. Kaykobad, E. G. Manning, and G. C. Shoja, "Solving the multidimensional multiple-choice knapsack problem by constructing convex hulls," *Comput. Oper. Res.*, vol. 33, no. 5, pp. 1259–1273, 2006.
[55] M. Khan et al., "Quality adaptation in a multisession multimedia system: Model, algorithms and architecture," Ph.D. dissertation, Dept. Elect. Comput. Eng., Univ. Victoria, Victoria, BC, Canada, 1998.
[56] J. Martín-Pérez et al., "Dimensioning V2N services in 5G networks through forecast-based scaling," *IEEE Access*, vol. 10, pp. 9587–9602 2022.
[57] A. Jeatsa, B. Teabe, and D. Hagimont, "CASY: A CPU cache allocation system for FaaS platform," in *Proc. IEEE CCGrid*, 2022, pp. 494–503.

Venkatarami Reddy Chintapalli (Graduate Student Member, IEEE) received the Ph.D. degree in computer science and engineering from the Indian Institute of Technology Hyderabad in 2022. He is currently working as an Assistant Professor with the National Institute of Technology Calicut, India.

Sai Balaram Korrapati received the B.Tech. degree in computer science and engineering from the Indian Institute of Technology Hyderabad in 2022. He is currently working as a Software Developer Engineer with Amazon India.

Madhura Adeppady received the M.Tech. degree in computer science and engineering from the Indian Institute of Technology Hyderabad in 2020. She is currently pursuing the Ph.D. degree with the Politecnico di Torino, Italy.

Bheemarjuna Reddy Tamma (Senior Member, IEEE) received the Ph.D. degree from the Indian Institute of Technology Madras, India, and then worked as a Postdoctoral Fellow with the University of California at San Diego prior to taking up faculty position with the Indian Institute of Technology Hyderabad. He is a Professor with the Department of CSE, Indian Institute of Technology Hyderabad. His research interests are in the areas of converged cloud radio access networks, SDN/NFV for 5G, network security, and green ICT.

Antony Franklin A (Senior Member, IEEE) received the Ph.D. degree from the Indian Institute of Technology Madras. He is an Associate Professor with the Department of CSE, Indian Institute of Technology Hyderabad (IITH). Before joining IITH, he worked as a Senior Engineer with the DMC R&D Center, Samsung Electronics, South Korea, where he was involved in the development of 5G networking technologies. He also worked as a Research Engineer with Electronics and Telecommunications Research Institute, South Korea, where he was involved in cognitive radio technology research. His current research interests include mobile networks (5G and Beyond 5G), C-RAN, MEC, IoT, and SDN/NFV.

Balaprakasa Rao Killi (Senior Member, IEEE) received the Ph.D. degree in computer science and engineering from the Indian Institute of Technology Guwahati. He is currently working as an Assistant Professor with the National Institute of Technology Warangal, India.