Characterizing and Orchestrating NFV-Ready Servers for Efficient Edge Data Processing

Lu Zhang
Shanghai Jiao Tong University
luzhang@sjtu.edu.cn

Chao Li
Shanghai Jiao Tong University
lichao@cs.sjtu.edu.cn

Pengyu Wang
Shanghai Jiao Tong University
wpybtw@sjtu.edu.cn

Yunxin Liu
Microsoft Research
yunxin.liu@microsoft.com

Yang Hu
The University of Texas at Dallas
yang.hu4@utdallas.edu

Quan Chen
Shanghai Jiao Tong University
chen-quan@sjtu.edu.cn

Minyi Guo
Shanghai Jiao Tong University
guo-my@sjtu.edu.cn

ABSTRACT

The fast-growing Internet of Things (IoT) and Artificial intelligence (AI) applications mandate high-performance edge data analytics. This requirement cannot be fully fulfilled by prior works that focus on either small architectures (e.g., accelerators) or large infrastructure (e.g., cloud data centers). Sitting in between the edge and cloud, there have been many server-level designs for augmenting edge data processing. However, they often require specialized hardware resources and lack scalability as well as agility.

Other than reinventing the wheel, we explore tapping into underutilized network infrastructure in the incoming 5G era for augmenting edge data analytics. Specifically, we focus on efficiently deploying edge data processing applications on Network Function Virtualization (NFV) enabled commodity servers. In such a way, we can benefit from the service flexibility of NFV while greatly reducing the cost of many servers deployed in the edge network. We perform extensive experiments to investigate the characteristics of packet processing in a DPDK-based NFV platform and discover the resource under-utilization issue when using the DPDK polling-mode. Then, we propose a framework named EdgeMiner, which can harvest the potentially idle cycles of the cores for data processing purpose. Meanwhile, it can also guarantee the Quality of Service (QoS) of both the Virtualized Network Functions (VNFs) and Edge Data Processing (EDP) applications when they are co-running on the same server.

CCS CONCEPTS

• **General and reference** → **Measurement**; • **Software and its engineering** → *Process management*; • **Networks** → Network servers.

KEYWORDS

Network Function Virtualization, resource under-utilization, edge data processing, QoS

ACM Reference Format:
Lu Zhang, Chao Li, Pengyu Wang, Yunxin Liu, Yang Hu, Quan Chen, and Minyi Guo. 2019. Characterizing and Orchestrating NFV-Ready Servers for Efficient Edge Data Processing. In *IEEE/ACM International Symposium on Quality of Service (IWQoS '19), June 24–25, 2019, Phoenix, AZ, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3326285.3329057

1 INTRODUCTION

Deploying data analytic applications and AI-enabled services near the edge is becoming increasingly popular today since moving stored or in-flight data to the cloud can be problematic. According to Gartner, there will be over 20 billion IoT devices installed by 2020 [9], which will create large quantities of data needing to be analyzed. Integrating domain-specific accelerators in end devices (e.g., smart phones or video cameras) can boost Edge Data Processing (EDP), but it is inadequate due to limited capacity and scalability. Faced with a deluge of edge traffic, it is also critical to tap into auxiliary edge systems for augmented EDP. Recently, many pioneer studies have explored such hierarchical topology, including the Cloud-Fog-Edge three-layer design [7, 38] or four-layer design [34].

However, there are no well-established architectures for the augmented EDP landscape [25]. Even though various edge computing solutions [5, 24, 32, 36, 37] have been explored over the years, they typically assume certain specialized hardware such as smart gateways, mini clusters, or network devices to process the raw data. For example, XPro [36] proposes an energy-efficient architecture for smart body sensing. ParaDrop [37] and Fog Computer [8] highlight smart gateways for processing data stream near users. At the server level, Cloudlet [32] uses virtualized local clusters to augment mobile services and InSURE [24] leverages standalone clusters to pre-process raw data onsite. Recently, Microsoft introduces Data Box Edge/Gateway [5], which is a physical network appliance that uses AI to analyze and transform data before it is uploaded to Azure.

The above systems improve different edge-oriented services for sure, but they can be less cost-efficient and scalable. Oftentimes, enormous mini data centers or fog computing nodes need to be

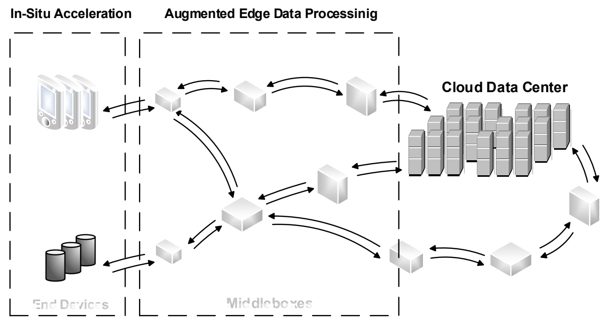


Figure 1: The case for Augmented Edge Data Processing

physically deployed in a specific region. This unavoidably increases capital expenses (Capex) and operational expenses (Opex). Worse, due to the reliance on specialized hardware, one can hardly accommodate edge traffic surge or react to failure scenarios. Without over-provisioning edge resources, one has to offload traffic to remote data centers every now and then.

Moreover, since existing designs introduce various application-specific platforms, it is difficult to coordinate diverse applications across different vendors. While all of these systems may operate under the same open standard in the future, currently there is no universally accepted one. The edge computing space is therefore somewhat fragmented. Many existing technologies only provide a certain degree of functionality. For example, popular suites of proprietary protocols including X10 [1], Z-Wave [2], and ZigBee [4], all of which are incompatible with each other. Unless a single specific kind of equipment is used, one has to find a way to share device data with others.

This work is driven by the observation that currently there is not a strong motivation for provisioning a wide spectrum of specialized appliances for generic EDP tasks. Rather than exploring and redefining a new type of edge architecture, we propose to unleash the performance and capacity potential on edge equipment already in place. Specifically, in this paper we set out to explore leveraging the under-utilized NFV servers at the edge to co-locate the emerging edge computation workloads.

Our argument stands on the recent trend towards Network Function Virtualization (NFV) [10] with the aim of improving the network's manageability and reducing the capital expenditure. NFV implements the network function (NF) (e.g., routing, detection, and load balance, etc.) as software in virtual machines (VMs) or containers, which allows virtual NFs to be deployed on commodity off-the-shelf (COTS) servers. On the other hand, being deployed between the end devices and the data centers, these COTS servers deployed with NFs are naturally formed as a mesh network of computing nodes that can process or store data locally and push all received data to a central cloud, as shown in Figure 1.

In fact, the European Telecommunications Standard Institute (ETSI) advocates reusing existing NFV infrastructure and the management capability of NFV to the largest extent possible in the edge computing era [16]. In recent years, NFV at the network edge (e.g. virtualized Customer Premises Equipment, vCPE) is gaining increasing attention. According to an IDC report [30], the worldwide market for NFV infrastructure at the network edge is expected to grow from a base of $67.8 million in 2016 to $1.16 billion in 2021 at a compound annual growth rate (CAGR) of 76.4%. Such increasing yet under-utilized [40] computing infrastructure makes it attractive for users who want more affordable, deploy-on-demand edge computing power. Doing so also allows one to speed up the adoption and delivery of our edge applications as well.

In this paper, we propose EdgeMiner, an augmented EDP framework that enables NFV-ready COTS servers to efficiently host augmented EDP applications such as the edge video encoding and image processing applications. Towards this goal, we perform extensive experiments to characterize the resource utilization of DPDK-based NFV COTS servers. Intel DPDK [18] is designed to mitigate the overheads of software packet processing by optimizing the kernel network stack and allowing direct data access to bypass the kernel. However, our characterization results show that, in contrast to the core network scenario, DPDK-based NFV platform has poor CPU utilization at the network edge with lower packet receive/sending rate. Motivated by our observation, EdgeMiner smartly mines the idle CPU resources of DPDK-based NFV platforms without impairing the performance and capacity of VNFs. It leverages batch processing and batch-interrupt to discover the idle CPU resource of DPDK-based NFV platforms. We also propose a Dynamic Search Core (DSC) algorithm to guarantee the QoS of EDP applications. We evaluate our framework using the typical network function and popular EDP applications such as image processing and video encoding.

To the best of our knowledge, this work is first to provision EDP applications on the flexible network infrastructures.

This paper makes the following key contributions:

- We conduct extensive experiments to measure the architecture characteristic of a DPDK-based NFV platform under various configurations. We demonstrate the under-utilization issue of servers.
- We propose EdgeMiner, a light-weight edge processing framework that allows VNFs and EDP tasks to co-locate on commodity servers. We design scheduling schemes that can guarantee the QoS of both types of applications.
- We evaluate EdgeMiner using typical network functions and EDP applications such as x264 (a video encoding benchmark) and vips (an image processing benchmark). We show that EdgeMiner can save 13%-90% CPU utilization when there are no EDP applications and guarantee the QoS of EDP applications when deploying them on the NFV platforms.

The rest of this paper is organized as follows. Section 2 provides the background and further motivates our work. Section 3 characterizes the packet processing behavior on DPDK-based NFV platforms and investigates the capacity potential of the COTS servers. Section 4 proposes our EdgeMiner framework. Section 5 evaluates our design with representative applications. Section 6 introduces related work and finally Section 7 concludes the paper.

2 BACKGROUND AND MOTIVATION

In this section we briefly introduce the essential concepts of NFV and EDP. Our driving insight is that traditional DPDK-based NFV platforms are generally underutilized in the edge; one can actually deploy EDP applications as a type of network function (NF) to fully utilize the NFV servers. In such a way, we can provide edge computing services with high cost-efficiency and service agility.

2.1 Network Function Virtualization

NFV proposes to move various network functions from hardware "middleboxes" to software appliances in VMs or containers. Deploying virtualized network functions (VNF) as software on commodity servers has many advantages such as reducing cost through workload consolidation, simplifying resource management, and fast deployment as so forth.

In order to achieve the line rate for packet processing, today's NFV-enabled server adopts several optimization strategies. Many NFV platforms take advantage of the state-of-the-art I/O libraries such as Intel DPDK [18], RP_Ring [28] and Netmap [31]. All the above libraries reduce the performance overheads in the traditional kernel network stack. In order to deeply study NFV platforms, in this paper, we mainly focus on NFV platforms using DPDK for high-performance packet I/O.

2.1.1 Packet Processing of DPDK. Intel DPDK packet I/O library offers a set of primitives that allow users to create efficient user-space NFs on x86 platforms, particularly for high-speed data plane applications. DPDK mainly operates in a polling mode rather than the traditional interrupt packet I/O, which can reduce the time spent for packet traveling in the server. DPDK also uses huge pages to pre-allocate large regions of memory, which can reduce the TLB miss and packet transmission overhead. In this case, applications perform DMA operations and access data directly from the NIC without involving kernel processing and memory copy. Moreover, some architectures can further use DDIO (Direct Data I/O) which directly accesses data from NICs to LLC to reduce the latency of memory access. Additionally, DPDK requires each VNF to occupy one dedicated CPU core to avoid context switches. All of these efforts are made to provide high-performance packet I/O operations and a high-performance NFV system.

While DPDK achieves very high throughput with aggressive optimization, it has sacrificed the efficiency of resource utilization. When the packet transmission rate is low (e.g., below 10 Gigabits/s), DPDK will suffer from rather poor CPU utilization due to its reliance on busy polling. Additionally, since it pre-allocates large regions of memory, it unnecessarily causes memory capacity waste when the packet access rate is low.

2.1.2 Service Function Chaining. Network services often require various NFs. A packet usually traverses a series of sequenced network functions before the final application processes it, which is called Service Function Chain (SFC). SFC implemented in NFV platforms can easily scale and change its locations on demand.

EFSI standards [10] show that different NFs have significantly different processing and performance requirements. Some NFs have a high per-core throughput (Mpps), e.g., switches; and some have a low throughput as a few kilo pps, e.g., encryption applications. In this case, one NF in a service chain that drops packets will waste vast amount of processing resources in earlier NFs of the chain. The imbalanced computing capabilities cause resource waste, and as the length of SFC increases, this situation will be worse.

2.2 Edge Data Processing on NFV Platforms

For traditional IoT systems, raw data generated from edge devices is sent to applications deployed in the Cloud through the network. There are several disadvantages of this Cloud-centric model: 1) large latency overhead: a large amount of data movement through the network will cause severe energy and latency overheads; 2) deployment problems: it is impractical to connect all the edge devices with the Cloud, and; 3) security and privacy issues: sending the edge data to the Cloud may raise security and privacy issues.

Rather than constantly moving a huge amount of data to the Cloud, recent years has witnessed a new trend towards edge-oriented data processing. EDP is a natural extension with the evolution of mobile base stations and the convergence of IT and telecommunications networking. Typically, EDP applications can be implemented in a virtualized environment [16, 32]. In other words, the underlying platform that hosts EDP tasks and NFV workloads is quite similar. EDP emphasizes data processing at the edge network, while the NFV platform is focused on processing network packets. With appropriate design and modification, one can actually deploy computational edge services as network functions. It allows operators to benefit as much as possible from their investment, by hosting both VNFs and EDP applications on the same platform.

Nevertheless, it is important not to aggressively subscribe the NFV-enabled servers as both EDP tasks and VNFs have QoS requirements. VNFs generally pursue high throughput and low latency for packets processing. Similarly, EDP applications face very strict deadline. Moreover, EDP applications are characterized by latency, proximity and real-time running. Thus, intelligent orchestration of computing and network resources is of paramount importance.

3 CHARACTERIZING NFV SERVERS

Traditional NFV platforms mainly focus on high performance but overlook resource efficiency. In this section, we conduct extensive experiments to investigate the resource utilization of a DPDK-based platform in terms of CPU and memory utilization under different configurations. The results of experiments show that there are lots of underutilized resources, which motivate our design of co-running EDP applications on network infrastructure.

3.1 Experiment Setup

3.1.1 Platform Configuration. Our physical platform configuration is shown in Table 1. The server uses 4 Intel I350 1Gigabit Ethernet NICs associated with the single socket. The operating system is Ubuntu Linux 16.04 and the version of DPDK is 17.08. All of our experiments are based on openNetVM [42].

The platform architecture in our experiment is shown in Figure 2. When the system starts, a manager thread pre-allocates memory pools (*rte_mempool_create*) in Huge Pages to store incoming packets. All threads are pinned to dedicated CPU cores in our experiment. Next, we will introduce the flow of packet processing in detail: When packets arrive in the system, the RX thread will fetch the data from the NIC with zero copy (❶). Then the RX thread will look up the flow table to decide which VNF is selected and then send the packet to the corresponding VNF's RX queue (❷). VNF1 reads packets from its RX queue and processes them using user-defined functions (❹). Then, all packets will be sent to VNF1's

Table 1: Platform configuration

Item	Configuration
COTS System	Ubuntu 16.04, single socket
Processor	Intel Xeon E5-2620 v3@2.40GHz 6 physical cores (disabled HyperThreading) 15MB L3 cache for the socket 64KB L1 cache,256KB L2 cache each core
Memory	32GB DDR4 total HugeSize=2MB and 1024 HugePage
NIC	Intel i350, 4 port each 1Gigabit

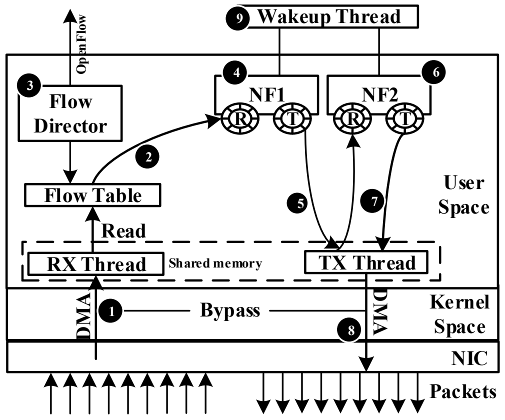


Figure 2: Packet processing on NFV platforms

TX queue. When the polling mode is used, CPU cores will be always busy to wait and process packets while VNF1 is running (i.e. busy polling). If the interrupt mode is used, the core pinned to VNF1 will be sleep when there is no incoming packets or other situations (e.g., backpressure). The wake-up thread aims to monitor each NF's information and decide when to activate the VNF thread (❾). The TX thread polls to read packets from the NF1's TX queue, and then moves the packets into another NF's RX (NF2 RX) or sends them to the NIC based on the destination information of the packets (❺). The packets processed by NF1 will be processed by NF2 (❻). Thus, the TX thread moves the packets from NF1 to NF2, and then NF2 processes the packets in the same manner as step ❹. After NF2 processes the packets, the TX thread checks the packet's destination information and decides to move the packet out of the system (❽).

3.1.2 Network Function Workloads. In our experiments, we choose three typical NFs to explore resource utilization characteristics of DPDK-based NFV platforms:

Forward: This network function is similar to the Ipv4/Ipv6 packet forwarding network function. After receiving packets, *Forward* reads the destination information of packets and then sends packets to the next VNF.

DPI: Deep Packet Inspection (DPI) is an essential security approach in the network. It is applied in network applications including network intrusion detection system (IDS) and Web application firewalls. This network function can locate, identify, classify, reroute and block packets.

Table 2: Different VNFs computation cost

Pkt Size \ NF	Forward	DPI	Aes_encrypt
64bytes	40cycles	300cycles	3.3K cycles
128bytes	40cycles	330cycles	9K cycles
256bytes	40cycles	340cycles	20K cycles
512bytes	40cycles	360cycles	43K cycles
1024bytes	40cycles	370cycles	88K cycles

AES_encryption/DES_encryption: This function aims to encrypt/decrypt UDP packets using specified encryption/decryption algorithm, and then forward them to specific NFs. Since this function needs to encrypt the packet payload using the encryption algorithm, it's a computation-intensive function.

Table 2 shows the computation cost of processing one packet under different packet sizes. *Forward* consumes only about 40 cycles and the computation cost maintains although the packet size increases due to its simple operation. *DPI* needs to read both the header and payload of a packet and inspect packet for security. With packet size increasing, its computation cost increases slightly. As for *AES_encrypt*, it needs not only to read the header and payload of a packet, but also to process the packet using a complex algorithm. Thus, *AES_encrypt* consumes the most cycles to process a packet, and the computation cost is highly correlated to the packet size.

3.1.3 Test Traffic. In this paper, we utilize a DPDK-based packet generator to generate packets with different packet sizes and different transmission rates. Since *AES_encrypt* can only process UDP packets, all the experiments are based on UDP packets. We have 4 ports in our packet generator server, we can generate up to 4 Gb network packets. The packet generator runs on a separate server and connects to the test server directly.

3.2 VNF Memory Bandwidth

Since memory access is an important component when VNFs process packets, it is necessary to study the characteristics of VNFs memory utilization. We use events provided by Performance Monitor Unit [6] to monitor the memory bandwidth. The resource monitor distills crucial information for the system to make informed decisions. We use *libpfm4* library to translate human-readable performance event names to machine-readable event code. The monitor uses the *perf_event* interface of Linux to access hardware performance counters. The performance event name used in the memory bandwidth monitor is *hswep_unc_imcX::UNC_M_CAS_COUNT*. This register counts the number of DRAM CAS commands recorded at integrated memory controller (IMC) at CPU socket X. Additionally, to obtain the peak memory bandwidth of our server, we utilize the STREAM Benchmark [3] to measure the peak memory bandwidth and we get 18.4GB/s.

As shown in Figure 3, we present the memory utilization of three network functions with different packet sizes and different transmission rates. For each VNF, the memory bandwidth usage grows as the packet size and transmission rate increase. It is clear that the memory bandwidth usage is very small in all the experiments, and the highest memory bandwidth usage is only 4.7% of the server

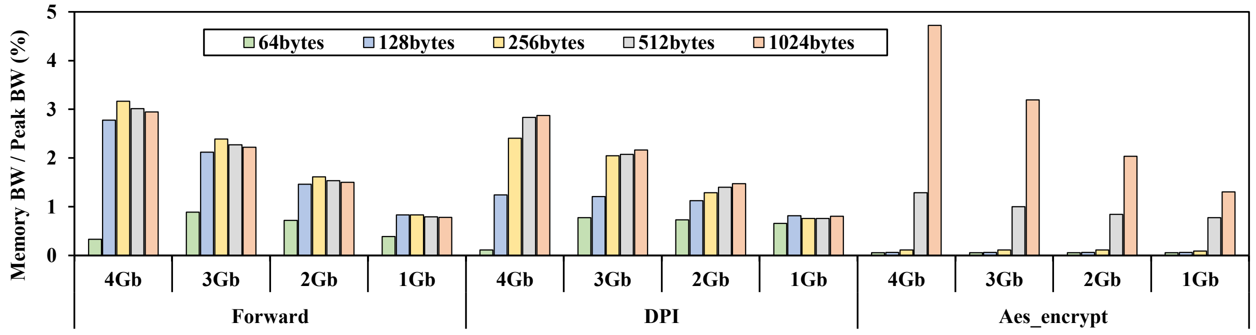


Figure 3: Memory bandwidth usage of different VNFs under different packet rates

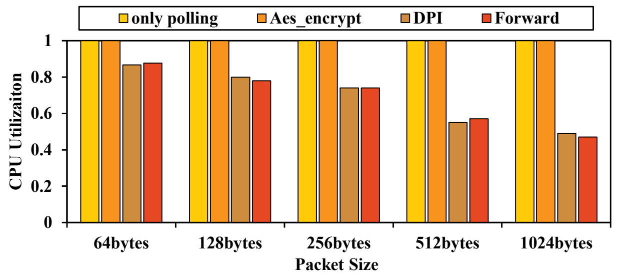


Figure 4: CPU utilization of on-demand interrupt

peak bandwidth usage from VNF *Aes_encrypt* with 1024 packet size under 4Gb speed. In this case, we can consider that VNFs in our experiments are not memory-bandwidth bounded. Thus, we will study the characteristics of CPU utilization in detail and focus on CPU utilization in our framework.

3.3 VNF CPU Utilization

In this subsection, we set up experiments to explore CPU utilization of VNFs. Under the DPDK polling mode, NFs that are busy waiting for packets waste lots of CPU resource in low-speed network. Thus, an interrupt mode would be helpful for saving the resource, as is done in NFV platforms such as Netmap [31] and ClickOS [27].

3.3.1 CPU Utilization of Single VNF. On-demand interrupt is used in Netmap and ClickOS: if there is no packet in the VNF RX queue, the NF Thread will enter sleep status and wait for the wakeup semaphore specifically for it. In the meantime, the wakeup thread monitors the VNF's RX queue. Once the VNF's RX queue is not empty, the wakeup thread will send the semaphore to wake up the VNF thread immediately.

Figure 4 shows the CPU utilization of different applications using this strategy. Since the polling mode of DPDK will be always busy waiting and processing the packets, the CPU utilization of the polling-only mode remains at 100%. The CPU utilization of *Aes_encrypt* is also 100% in both polling and interrupt mode due to its high computation cost. *DPI* and *Forward* reveal important CPU resource-wasting information in DPDK-based platforms. As the packet size grows, the CPU utilization of the NF decreases mainly because the packet rate decreases and the NF has larger probability to become the SLEEP status.

Considering that a VNF needs to switch between the SLEEP and ACTIVE status in the interrupt mode, the context switch and other extra system costs are inevitable. To obtain a detailed CPU utilization breakdown of interrupt mode, we collect two kinds of CPU utilization: User CPU, which is occupied by the user application; and System CPU, which contains CPU time utilized by system interrupts, context switches and other system operations. In this part, we only explore the CPU utilization of *Forward* and *DPI* since *Aes_encrypt* are always busy.

Figure 5 shows a breakdown of CPU utilization of VNFs in the on-demand interrupt mode. Except for 64Bytes packet processing in *DPI*, all the other results show that user-defined functions only account for less than 50% total CPU cycles. When running *Forward*, there is only at most 36% CPU utilization on user applications and the others are wasted by system operations. Both results show huge resource waste and with the packet size increasing, system operation cost becomes more severe.

However, the wasted CPU utilization is beyond what we can see in the above cases. Considering that when the packet size is 64 bytes, the VNF will receive a packet every 1488 cycles with 2.4GHz frequency. Note that *Forward* processes a packet with about 40 cycles. Thus, CPU utilization for processing a packet is $40/1488 \approx 3\%$. Even though this procedure contains other user-defined applications such as reading packets from the RX queue and writing packets into the TX queue, 3% and 82% have too large difference.

To handle the shortcoming of on-demand interrupt, we design a batch interrupt to measure this system. The experiments are as follows: when VNF's RX queue is empty, the NF thread will enter SLEEP status to save the CPU cycles. The Wakeup Thread monitors VNF's RX queue and if the size of the RX queue is larger than the batch size, it will send the ACTIVE semaphore to the VNF Thread.

Figure 6 shows the CPU utilization in the batch interrupt mode. Both ***Forward*** and ***DPI*** save a great amount of CPU utilization. In addition to the total CPU utilization, we explore the breakdown of CPU utilization in this batch situation as well. When it uses batch interrupt manner, user CPU utilization will occupy a large ratio in both VNFs. Even in the worst case, when the packet size is 1024 bytes, the user CPU utilization will occupy 67% and 82% in *Forward* and *DPI* respectively. In this case, using batch interrupt is

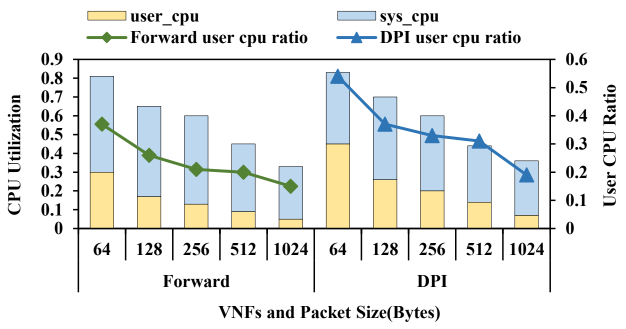


Figure 5: CPU utilization breakdown of on-demand interrupt

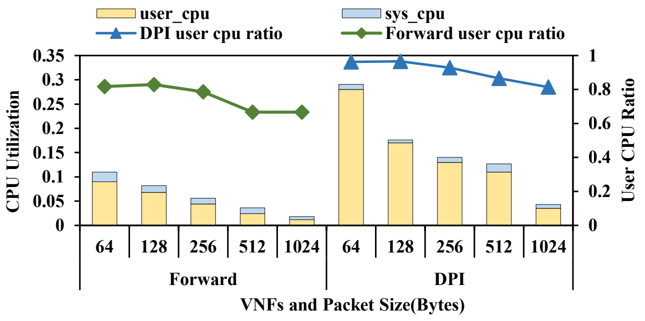


Figure 6: CPU utilization breakdown for batch interrupt

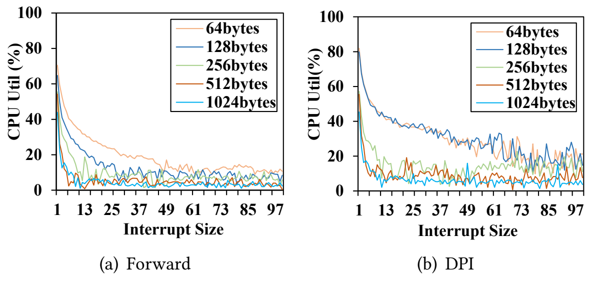


Figure 7: CPU utilization of varying interrupt size

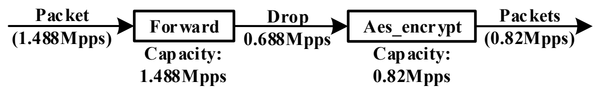


Figure 8: The example of SFC with different cost

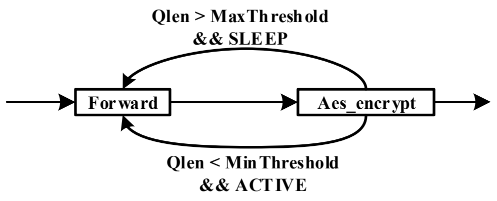


Figure 9: Local backpressure diagram

a good way to save CPU utilization for energy-saving purpose or data processing purpose.

To understand the relationship between batch size and CPU utilization, we design a fine-grained experiment to measure CPU utilization. In our experiment, we change the batch size from 1 to 100. As shown in Figure 7, as interrupt size increases, CPU utilization of VNFs decreases. In our experiment, no experiment drops any packets. In other words, it keeps a desired throughput of VNFs. Our results show that smartly choosing the interrupt size allows us to save more resources.

3.3.2 CPU Utilization in Service Function Chain. In the above subsections, we have discussed the CPU utilization of single VNF in DPDK-based NFV platforms with different configurations. In the real world, network functions will be chained to process the packets assigned by users. The CPU utilization of Service Function Chain (SFC) is also an important design consideration. Due to the imbalanced computation cost in SFC, the downstream VNF will be overloaded and it will drop the packets which have been processed in the upstream NF. The resources used by the upstream NF is useless if the downstream NF has larger computation cost than the upstream VNF[11]. Figure 8 shows such situation when wasted work exists.

In order to avoid wasted work due to the imbalance of VNFs in SFC, we design and implement a local backpressure algorithm as shown in Figure 9. The Wakeup Thread checks the information of the upstream VNF (***Forward/DPI***) Thread and the downstream VNF (***Aes_encrypt***). Once the queue size of ***Aes_encrypt*** is larger than the maximum threshold, the ***Forward/DPI*** Thread will enter SLEEP status, and if the queue size of ***Aes_encrypt*** is less than the min threshold, the Wakeup Thread will immediately wake up the ***Forward/DPI*** Thread and continue processing packets. In this way, we can guarantee the throughput of the whole SFC and reduce CPU utilization waste.

We test the CPU utilization of the first network function as shown in Table 3. In this case, we can also see lots of CPU utilization saved using backpressure.

All above experiments aim to show the characteristic of the CPU utilization on DPDK-based NFV platforms. We mine those free CPU cycles of servers running VNFs and we deploy EDP applications on the NFV platforms to share the precious computing resource. In the next section, we will discuss how EDP can be deployed in the NFV platform with the QoS guarantee of both the VNF and the EDP.

4 DEPLOYING EDGE COMPUTING

Our characterization and evaluation results show that there are still lots of resources wasted in traditional DPDK-based NFV platforms. How to utilize those idle resources presents a great challenge. An intuitive way to improve system utilization is to co-locate some different applications on the same server [19]. Considering that there are a growing amount of EDP applications (e.g., image processing, video processing) actively looking for auxiliary computing resource support at the edge, it would be highly profitable if we

Table 3: Influence of backpressure

	Polling Mode		**Backpresure**			
	Service Rate (Mpps)	Drop (Mpps)	Service Rate (Mpps)	Drop (Mpps)	Forward CPU Util.	DPI CPU Util
64B	1.488	0.688	0.87	0.07	9%	20%
128B	0.844	0.524	0.33	0.04	3%	7.6%
256B	0.453	0.313	0.142	0.012	1.5%	3.3%
512B	0.23	0.163	0.0668	0.008	0.7%	1.7%
1024B	0.12	0.0877	0.0323	0.003	0.3%	1%

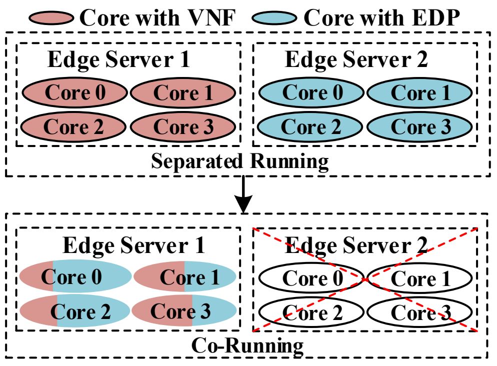


Figure 10: Resource saving with core sharing

can smartly deploy them on NFV platforms. In this case, we can jointly improve network infrastructure utilization and reduce edge server requirement, as shown in Figure 10.

In this paper, we propose EdgeMiner, a resource harvesting strategy for deploying EDP applications on flexible network infrastructure. Some prior works, which focus on the application co-location in a data center, allow the latency-critical and batch applications to share the memory system resources (e.g., Last Level Cache, DRAM bandwidth) [26, 39]. However, very few studies focus on sharing the CPU cores for edge computation augmentation. Prior work [19] tries to enable the sharing of cores in a data center, but it mainly emphasizes sharing-aware dynamic voltage and frequency scaling (DVFS). Different from prior arts, EdgeMiner is a simple but effective method that allows one to co-run packet processing tasks and EDP applications on shared cores.

Co-running EDP applications and VNFs is non-trivial. Due to the transmission latency, unfinished EDP applications cannot be moved to the Cloud in the last minute. These tasks have to respect a strict deadline and be well-managed. We cannot simply schedule EDP applications only during the server's idle period, since there exists limited CPU slack lime that can be exploited on current NFV platforms. Thus, we design a scheduling algorithm to guarantee the QoS of EDP applications.

4.1 Overview of EdgeMiner

EdgeMiner can improve the CPU utilization of edge servers as well as meeting the QoS of EDP applications. The overview of EdgeMiner is shown as Figure 11. This framework mainly contains three modules: EDP profiling, sharing initialization, and dynamic tuning. Importantly, we use a metric called QoS Urgency (QU) to evaluate the difficulty of achieving the QoS.

$$QU = \frac{actual\ runtime}{time\ limit} \tag{1}$$

As an EDP application approaches the deadline, its QU becomes larger. In this case, it often becomes more difficult to meet the target QoS due to limited time and greatly increased computing resource requirement. In the following subsections, we will discuss the three components in detail.

4.1.1 EDP Profiling. In this component, we characterize common EDP applications and record their approximate runtime. Thus, we can estimate the CPU cycles needed for processing incoming edge applications, based on profiling results such as sizes of data. Additionally, each EDP application also has its certain QoS requirement which we can consider it as the time limit. After profiling, we can calculate the QU of each EDP application. This is the most important factor to consider in the following steps.

4.1.2 Sharing Initialization. In this component, we decide how to co-run the NFs and the EDP applications. In Figure 11, we can see that different cores and different EDP applications are labeled by different colors. The core color shown in the figure represents different CPU utilization used by VNFs. Similarly, different color of EDP applications means different QU value. In the initialization phase, we sort the EDP applications by the QU and the CPU utilization of the core. Afterwards, we co-run the least-QU EDP application with the highest CPU utilization core; we also co-run the highest-QU EDP application with the lowest CPU utilization core. If there exists two EDP applications that have the same QU, we sort them based on the time limit. If the number of EDP applications is larger than the number of NFs, we will run the EDP applications on the idle core. If the number of NFs is larger than the number of EDP applications, there will be some cores only host VNFs. According to the above co-running mechanism, VNFs that exhibit the lowest utilization will occupy a single core without sharing. In this case, it will save power due to the low CPU consumption.

4.1.3 Dynamic Tuning. The above co-running mechanism alone may not meet EDP applications' target QoS. In order to meet the QoS requirement, we monitor the information of EDP applications and check if EDP applications should move to the idle core or the core which has more CPU resources. We called this mechanism as **D**ynamic **S**earch **C**ore (DSC), as shown in Algorithm 1. It allows EdgeMiner to efficiently tap into current NFV-ready servers.

5 EVALUATION

In this section, we firstly evaluate the performance of EDP applications when co-running with VNFs using batch-interrupt and the backpressure algorithm. Secondly, we demonstrate the performance of our DSC algorithm.

5.1 Workload

We have introduced VNFs and the DPDK-based platform in Section 3. Since we mainly focus on CPU utilization, we select CPU-bound applications as our benchmarks. We choose two applications from

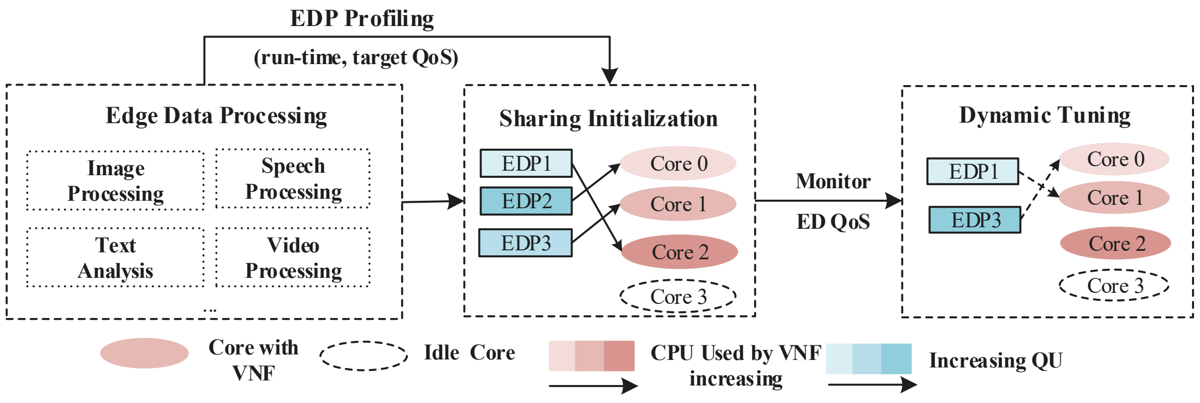


Figure 11: Overview of EdgeMiner

Algorithm 1 Dynamic Search Core Algorithm

```
Definition:
    $T_{real}$ : Time really elapsing
    $T_{target}$ : Longest time of Edge data processing
    $T_{single}$ : Runtime of EDP running in single core
    $T_{trans}$ : Time to change core
    $Ratio_i$ : Ratio of NF using the $i^{th}$ core
1: for EDP is running with $core_j$ do
2:     Get $T_{real}$
3:     Find the CORESET whose $Ratio < (1 - QU)$
4:     for $core_k \in CORESET$ do
5:         $T_{trans} = \frac{(1-Ratio_k)\cdot T_{target} - T_{single}}{Ratio_j - Ratio_k}$
6:         if $T_{trans} > T_{real}$ then
7:             Continue
8:         else if $T_{trans} < T_{real}$ then
9:             remove $core_k$ out of CORESET
10:        else
11:            change EDP from $core_j$ to $core_k$
12:        end if
13:    end for
14: end for
```

Table 4: Edge processing workloads

Name	Information	Workload
vips	Image processing	Image w/ 18K * 18K pixels
x264	video encoding	30fps/640 * 360 pixels

PAR-SEC [35]. The detailed information of workloads is shown in Table 4. Both the two workloads are popular in EDP applications: *vips* is an image processing library and *x264* is an application for encoding video streams into the H.264 compression format.

5.2 Effectiveness of Batch Interrupt

We evaluate the impact of batch packet processing mechanism on EDP applications performance. We run EDP applications on the core that has used by VNFs and record the EDP application's processing time. Figure 12 shows the normalized processing time of different co-running combination. As the figure shows, the batch interrupt scheme can steeply degrade the EDP application's processing time compared with the on-demand interrupt mechanism. It is evident that we can harvest more CPU resources on VNF-ready servers to process EDP applications.

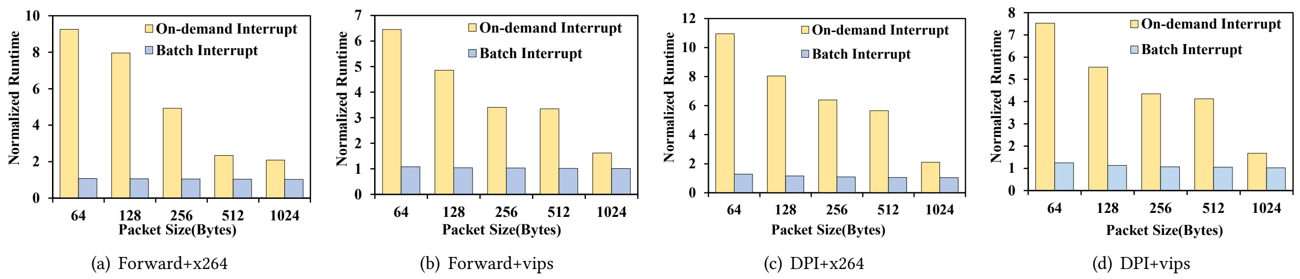
5.3 Effectiveness of Back Pressure Algorithm

As mentioned in section 3.3.2, local backpressure algorithm can save CPU utilization in SFC without affecting the performance of network services. In this experiment we use backpressure algorithm to harvest CPU free cycles and measure the performance of EDP applications. As shown in Figure 13, the normalized performance degradation of co-located EDP applications is less than 15%. Importantly, our results show that if we use the backpressure mechanism, it only slightly delays EDP applications when the packet size is large (e.g., more than 256B). In other words, we may provide high-performance processing of EDP applications depending on the characteristics of the network packets.

5.4 QoS of Edge Data Processing

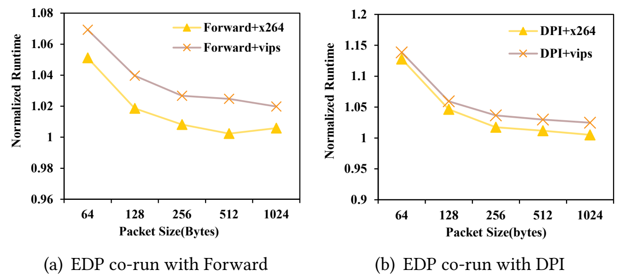
Although batch interrupt and backpressure schemes provide a way to exploit free computing resources, they still cannot provide satisfactory QoS guarantee. To ensure better QoS of EDP applications, it is important to carefully schedule computing resources. We use the 90% QoS policy (the QoS target performance is 90% of solo performance) to evaluate our DSC algorithm. According to the 90% QoS policy, if the run-time degradation is less than 1.11x, there will be no impact on the QoS.

For the above experiment, when the EDP applications (both the x264 and vips) co-run with *Forward*, all the degradation of EDP applications are less than 1.11x (the most is 1.08 when the packet size is 64 and *Forward* co-runs with x264). In this case, using the batch packet processing, we can guarantee the QoS of certain EDP applications even without extra compensation. Moreover, we test the 95% QoS policy as well. The results are shown in Figure 14. From Figure 14, we can see that the DSC can achieve the QoS of the EDP applications (both 90% QoS policy and 95% QoS policy).



(a) Forward+x264 (b) Forward+vips (c) DPI+x264 (d) DPI+vips

Figure 12: The Influence of batch processing when co-running the NFs and the edge date processing. a)-d) represent different co-location between the VNFs and EDP applications. Y-axis represents the time co-running with NF / the time when running the edge data processing in a single core.



(a) EDP co-run with Forward (b) EDP co-run with DPI

Figure 13: Normalized runtime when EDP applications co-run with VNFs using the backpressure algorithm

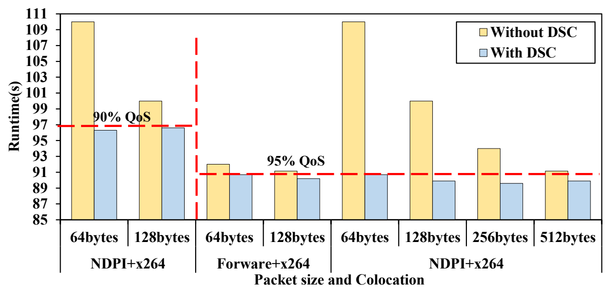


Figure 14: Using DSC algorithm to meet QoS of EDP

6 RELATED WORK

In this section, we discuss representative prior studies in different domains that are most relevant to our work.

6.1 High-Performance NFV System

Many prior works aim to provide high-performance and flexible platforms for NFV. Prior works [23, 29] enhance the individual NF performance to speed up the NFV's performance; Some works [17, 27, 42] focus on the packet delivery between different NFs to reduce the performance degradation; In works [14, 20], they employ heterogeneous platforms such as GPU to accelerate the packet processing. There are also some works use the characteristic of computer architecture to accelerate the packet processing such as threading scheduling [13] and reducing cache miss [15]. All of prior works focus on high performance of packet processing to meet the network line rate. They lack considering the resource utilization. Except for these NFV acceleration techniques, the NFVnice [21] and Flurries [41] benefit from running multiple NFs in a single core to increase the core utilization and make the NFV system more flexible. These works are orthogonal to our work and we mainly focus on the edge server to deploy EDP applications in the core used by the NF to increase the core utilization.

6.2 Edge Data Processing

To reduce the latency, there is a trend to process data on the edge devices close to the users. Li et al. [24] develop a sustainable in-situ server system to pre-process the raw data near where the data is located. However, this work mainly focuses on the energy management in the in-situ data center. Lane et al. [22] use a static model to perform deep learning recognition on IoT devices; Song et al. [33] represent an autonomous and incremental computing framework and architecture for deep learning based IoT applications. All the above work mainly focuses on how to process the edge data in a high-performance or sustainable way. However, they don't consider how to deploy EDP applications in the existed systems such as the NFV-Ready servers.

6.3 Improving Utilization of Data Processing

There have been many prior works on improving the QoS of latency-critical applications [26, 39]. These works leverage different mechanisms to make the latency-critical applications co-located with a batch job on the same server to improve the resource utilization. It is critical to guarantee the QoS of the latency-critical applications at the same time. Bubble-Up [26] and Bubble-Flux [39] bound performance degradation while improving chip multiprocessor utilization. Additionally, HOPE [12] exploits management workloads scheduling and applies graph-based task allocation to improve computation and reduce energy consumption. However, all these works mainly focus on the interference between the latency-critical tasks and batch jobs or performance improving of tasks in a data center. Our work concentrates on the edge computing scenario and the co-running of VNFs and EDP applications.

7 CONCLUSION

The rapid growth of IoT applications places demand on innovative data processing infrastructure that are responsive, efficient, and scalable. In this paper, we study the CPU utilization characteristics of DPDK-based NFV platforms in detail to demonstrate the underutilization issue of servers in edge network. We show that one can effectively harvest 13%-90% free CPU resources on DPDK-based platforms with batch interrupt mechanism and backpressure algorithm. Using the saved resources, we propose EdgeMiner, a light-weight edge processing framework that allows VNFs and EDP tasks to co-run on commodity servers. We also devise scheduling schemes that can guarantee the QoS of both types of applications. The proposed design, which expands the breadth and impact of today's network infrastructure, has the potential to greatly contribute to building a more connected, smarter world.

8 ACKNOWLEDGEMENT

This work is partially sponsored by the National R&D Program of China (No. 2018YFB1004800). Yang Hu is supported by NSF grant 1822985. Corresponding author is Chao Li.

REFERENCES

[1] 2014. X10 Protocol. https://buildyoursmarthome.co/home-automation/protocols/x10/.

[2] 2014. Z-Wave. https://buildyoursmarthome.co/home-automation/protocols/z-wave/.

[3] 2016. STREAM Benchmark. http://www.cs.virginia.edu/stream/FTP/Code/.

[4] 2018. ZigBee Alliance. https://www.zigbee.org.

[5] Azure. 2019. Azure Data Box family. https://azure.microsoft.com/en-us/services/storage/databox/.

[6] Intel. Corporation. 2017. Intel 64 and ia-32 architectures developer's manual,. https://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-software-developer-manual-325462.html.

[7] Amir Vahid Dastjerdi and Rajkumar Buyya. 2016. Fog computing: Helping the Internet of Things realize its potential. *Computer* 49, 8 (2016), 112–116.

[8] Harishchandra Dubey, Jing Yang, Nick Constant, Amir Mohammad Amiri, Qing Yang, and Kunal Makodiya. 2015. Fog data: Enhancing telehealth big data through fog computing. In *Proceedings of ASE BigData & SocialInformatics 2015*. ACM, 14.

[9] Nathan Eddy. 2015. Gartner: 21 Billion IoT Devices To Invade By 2020. https://www.informationweek.com/mobile/mobile-devices/gartner-21-billion-iot-devices-to-invade-by-2020/d/d-id/1323081.

[10] GSNFV ETSI. 2013. Network functions virtualization (nfv): Architectural framework. *ETsI Gs NFV* 2, 2 (2013), V1.

[11] Joel Halpern and Carlos Pignataro. 2015. *Service function chaining (sfc) architecture*. Technical Report.

[12] Yang Hu, Chao Li, Longjun Liu, and Tao Li. 2016. HOPE: Enabling Efficient Service Orchestration in Software-Defined Data Centers. In *Proceedings of the 2016 International Conference on Supercomputing (ICS '16)*. ACM, New York, NY, USA, Article 10, 12 pages. https://doi.org/10.1145/2925426.2926257

[13] Yang Hu and Tao Li. 2016. Towards efficient server architecture for virtualized network function deployment: Implications and implementations. In *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Press, 8.

[14] Yang Hu and Tao Li. 2018. Enabling Efficient Network Service Function Chain Deployment on Heterogeneous Server Platform. In *High Performance Computer Architecture (HPCA), 2018 IEEE International Symposium on*. IEEE, 27–39.

[15] Yang Hu, Mingcong Song, and Tao Li. 2017. Towards full containerization in containerized network function virtualization. *ACM SIGOPS Operating Systems Review* 51, 2 (2017), 467–481.

[16] Yun Chao Hu, Milan Patel, Dario Sabella, Nurit Sprecher, and Valerie Young. 2015. Mobile edge computing - A key technology towards 5G. *ETSI white paper* 11, 11 (2015), 1–16.

[17] Jinho Hwang, K K_ Ramakrishnan, and Timothy Wood. 2015. NetVM: high performance and flexible networking using virtualization on commodity platforms. *IEEE Transactions on Network and Service Management* 12, 1 (2015), 34–47.

[18] Intel. 2012. Data Plane Development Kit. https://www.dpdk.org/.

[19] Harshad Kasture, Davide B Bartolini, Nathan Beckmann, and Daniel Sanchez. 2015. Rubik: Fast analytical power management for latency-critical systems. In *Microarchitecture (MICRO), 2015 48th Annual IEEE/ACM International Symposium on*. IEEE, 598–610.

[20] Joongi Kim, Keon Jang, Keunhong Lee, Sangwook Ma, Junhyun Shim, and Sue Moon. 2015. NBA (network balancing act): A high-performance packet processing framework for heterogeneous processors. In *Proceedings of the Tenth European Conference on Computer Systems*. ACM, 22.

[21] Sameer G Kulkarni, Wei Zhang, Jinho Hwang, Shriram Rajagopalan, KK Ramakrishnan, Timothy Wood, Mayutan Arumaithurai, and Xiaoming Fu. 2017. Nfvnice: Dynamic backpressure and scheduling for nfv service chains. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 71–84.

[22] Nicholas D Lane, Sourav Bhattacharya, Petko Georgiev, Claudio Forlivesi, and Fahim Kawsar. 2015. An early resource characterization of deep learning on wearables, smartphones and internet-of-things devices. In *Proceedings of the 2015 international workshop on internet of things towards applications*. ACM, 7–12.

[23] Bojie Li, Kun Tan, Layong Larry Luo, Yanqing Peng, Renqian Luo, Ningyi Xu, Yongqiang Xiong, Peng Cheng, and Enhong Chen. 2016. Clicknp: Highly flexible and high performance network processing with reconfigurable hardware. In *Proceedings of the 2016 ACM SIGCOMM Conference*. ACM, 1–14.

[24] Chao Li, Yang Hu, Longjun Liu, Juncheng Gu, Mingcong Song, Xiaoyao Liang, Jingling Yuan, and Tao Li. 2015. Towards sustainable in-situ server systems in the big data era. In *ACM SIGARCH Computer Architecture News*, Vol. 43. ACM, 14–26.

[25] Chao Li, Yushu Xue, Jing Wang, Weigong Zhang, and Tao Li. 2018. Edge-Oriented Computing Paradigms: A Survey on Architecture Design and System Management. *ACM Computing Surveys (CSUR)* 51, 2 (2018), 39.

[26] Jason Mars, Lingjia Tang, Robert Hundt, Kevin Skadron, and Mary Lou Soffa. 2011. 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*. ACM, 248–259.

[27] Joao Martins, Mohamed Ahmed, Costin Raiciu, Vladimir Olteanu, Michio Honda, Roberto Bifulco, and Felipe Huici. 2014. ClickOS and the art of network function virtualization. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*. USENIX Association, 459–473.

[28] ntop. 2015. PF_RING ZC. http://www.ntop.org/products/pf_ring/pf_ring-zc-zero-copy/.

[29] Aurojit Panda, Sangjin Han, Keon Jang, Melvin Walls, Sylvia Ratnasamy, and Scott Shenker. 2016. NetBricks: Taking the V out of NFV.. In *OSDI*. 203–216.

[30] Rohit Mehra Rajesh Ghai, Petr Jirovsky. [n. d.]. Worldwide vCPE/uCPE Forecast, 2017âĂŞ2021: NFV at the Network Edge. https://www.idc.com/getdoc.jsp?containerId=US41429616.

[31] Luigi Rizzo. 2012. Netmap: a novel framework for fast packet I/O. In *21st USENIX Security Symposium (USENIX Security 12)*. 101–112.

[32] Mahadev Satyanarayanan, Victor Bahl, Ramón Caceres, and Nigel Davies. 2009. The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing* (2009).

[33] Mingcong Song, Kan Zhong, Jiaqi Zhang, Yang Hu, Duo Liu, Weigong Zhang, Jing Wang, and Tao Li. 2018. In-Situ AI: Towards Autonomous and Incremental Deep Learning for IoT Systems. In *2018 IEEE InternatiOnal SympOsium On High PerfOrmance COmputer Architecture (HPCA)*. IEEE, 92–103.

[34] Bo Tang, Zhen Chen, Gerald Hefferman, Tao Wei, Haibo He, and Qing Yang. 2015. A hierarchical distributed fog computing architecture for big data analysis in smart cities. In *Proceedings of ASE BigData & SocialInformatics 2015*. ACM, 28.

[35] Princeton University. 2010. PARSEC. http://parsec.cs.princeton.edu/.

[36] Aosen Wang, Lizhong Chen, and Wenyao Xu. 2017. XPro: A cross-end processing architecture for data analytics in wearables. In *ACM SIGARCH Computer Architecture News*, Vol. 45. ACM, 69–80.

[37] Dale Willis, Arkodeb Dasgupta, and Suman Banerjee. 2014. ParaDrop: a multi-tenant platform to dynamically install third party services on wireless gateways. In *Proceedings of the 9th ACM workshop on Mobility in the evolving internet architecture*. ACM, 43–48.

[38] Yi Xu and Sumi Helal. 2014. Application caching for cloud-sensor systems. In *Proceedings of the 17th ACM international conference on Modeling, analysis and simulation of wireless and mobile systems*. ACM, 303–306.

[39] Hailong Yang, Alex Breslow, Jason Mars, and Lingjia Tang. 2013. Bubble-flux: Precise online qos management for increased utilization in warehouse scale computers. In *SIGARCH Computer Architecture News*, Vol. 41. ACM, 607–618.

[40] Yishay Yovel. 2017. Why NFV is Long on Hype, Short on Value. https://www.catonetworks.com/blog/why-nfv-is-long-on-hype-short-on-value/.

[41] Wei Zhang, Jinho Hwang, Shriram Rajagopalan, KK Ramakrishnan, and Timothy Wood. 2016. Flurries: Countless fine-grained nfs for flexible per-flow customization. In *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*. ACM, 3–17.

[42] Wei Zhang, Guyue Liu, Wenhui Zhang, Neel Shah, Phillip Lopreiato, Gregoire Todeschi, KK Ramakrishnan, and Timothy Wood. 2016. OpenNetVM: A platform for high performance network service chains. In *Proceedings of the 2016 workshop on Hot topics in Middleboxes and Network Function Virtualization*. ACM, 26–31.