MTDA: Efficient and Fair DPU Offloading Method for Multiple Tenants

Zhaoyang Huang, Yanjie Tan, Yifu Zhu, *Graduate Student Member, IEEE*, Huailiang Tan, and Keqin Li, *Fellow, IEEE*

***Abstract*—In modern cloud computing environment, the offloading potential of DPU must be fully exploited for multiple tenants. Existing DPU offloading techniques lack the capability to perform the fair allocation of a DPU domain's internal resources among tenants with various performance requirements. In this article, we propose a virtual multi-channel DPU offloading architecture for multiple tenants (MTDA) and implement it on a BlueField-2 DPU platform to achieve stability and fairness in resource allocation for generic datacenter tasks. MTDA provides an independent virtual channel for each tenant before their requests are submitted to avoid competition among tenants. Considering the diverse requirements of tenants, MTDA constructs a credit-based resource allocation model and a traffic-aware scheduling algorithm to fully utilize the rich computing resources of DPU and improve the fairness of DPU resource allocation. Experimental results show that MTDA increases the throughput by up to 101.2%, 143.2%, 36.1%, and 41.7%, lowers the latency by up to 50.3%, 58.9%, 26.6%, and 29.4%, improves the fairness by up to 98.8%, 99.0%, 98.3%, and 98.4%, and provides more stable performance for multi-tenants, compared with DPDK, iPipe, FairNIC, and LogNIC.**

***Index Terms*—Credit model, DPU, fair resource allocation, multi-tenant.**

I. Introduction

With the marvelous development of cloud computing and network, the explosive growth of data in datacenters has driven the leap of network bandwidth from 10 Gbps to 400 Gbps [1], [2]. However, the stagnation of CPU computing power in recent years has caused traditional data centers, which rely on CPUs to process data, to be overwhelmed, i.e., there exist too many data center taxes in datacenters that prevent precious CPU resources from being released.

The emergence of Data Processing Unit (DPU) bridges the gap between the growth of network bandwidth and the stagnation of CPU computing power in the data center [3]. The rich computing resources in DPU, including various hardware accelerators for packet processing and specialized functions (such as crypto, compression, and hash, etc.), on-board DRAM, and multi-core processors (for some multicore SoC DPU), endow it the potential to offload the generic datacenter tasks for multiple tenants, which can release CPU resources to improve the efficiency of the entire computing system and lower the total cost of the overall system. In recent years, several researchers have explored the offloading ability of DPU from various aspects [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13]. Detailed comparisons are summarized in Table I. AccelTCP [4], Lynx [5] and UNO [6] focus on offloading network tasks or services to reduce the network overhead. LineFS [7], Xenic [8], and LeapIO [9] aim to offload various storage protocols into DPU to accelerate the distributed storage systems. Unlike the fronted two categories, Floem [10] and iPipe [3] explore the design space to offload computation tasks and applications into DPU to enhance the performance and lower the latency. However, most of them are engaged in offloading various applications or services into DPU and ignore the competition and fair resource allocation among tenants since all the tenants share the same physical device and will compete for the DPU's internal resources.

TABLE I
Related Work

Methods	Multi-tenant Isolation	Dynamic Scheduling	Fairness Control
AccelTCP [4], LineFS [7], Xenic [8], Floem [10]	×	×	×
iPipe [3], UNO [6]	×	✓	×
FairNIC [11], λ-NIC [13]	✓	×	✓
LogNIC [12], Lynx [5], LeapIO [9]	✓	×	×
MTDA (Our work)	✓	✓	✓

Optimally managing and fairly allocating DPU resources among multiple tenants is an effective technique to improve the throughput of offloading applications. There are several discussions and methods for allocating and scheduling DPU resources [3], [11], [12], [13], [14]. Liu et al [14] also conclude that the dynamic load conditions of DPU should be taken into consideration for improving the performance of data centers, which is one of the research challenges that must be solved. However, there are no further solutions in their research. iPipe [3]

Manuscript received 7 November 2023; revised 15 July 2024; accepted 15 July 2024. Date of publication 25 July 2024; date of current version 30 December 2024. This work was supported in part by the special funding for the construction of innovative provinces in Hunan Province under Grant 2021GK4012, in part by the National Natural Science Foundation of China under Grant 62302158, in part by the Hunan Provincial Natural Science Foundation of China under Grant 2023JJ40175. *(Zhaoyang Huang and Yanjie Tan are co-first authors.) (Corresponding author: Huailiang Tan.)*

Zhaoyang Huang, Yanjie Tan, Yifu Zhu, and Huailiang Tan are with the College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China (e-mail: huangzhaoyang@hnu.edu.cn; tanhuailiang@hnu.edu.cn).

Keqin Li is with the College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China, and also with the Department of Computer Science, State University of New York, New Paltz, NY 12561 USA (e-mail: lik@newpaltz.edu).

Digital Object Identifier 10.1109/TSC.2024.3433588

proposes an actor-based hybrid scheduler, which combines First Come First Serve (FCFS) and Deficit Round Robin (DRR) based processor sharing, to promote the offloading benefits. Although iPipe achieves dynamic resource scheduling through the hybrid scheduler, it fails to consider the interference among tenants and fair DPU resource allocation is also ignored. λ-NIC [13] maps lambdas across different memory hierarchies to isolate memory access and ensure fair allocation of resources, but it is only suitable for serverless computing scenarios. FairNIC [11] provides strict core partitioning, cache and memory striping, and rate limiting of access to the fixed-function hardware accelerator unit to achieve isolation and fair allocation for DPU resources. However, this simple static partition method does not consider the fairness of DPU resource allocation under diverse applications of multiple tenants, each with varying performance requirements. LogNIC [12] develops an optimizer to guide fine-grained resource allocation and computation placement. However, it requires an offline characterization phase to acquire essential parameters as model input, rendering it inadequate for adapting to the dynamic multi-tenant environment.

Existing DPU offloading architectures fail to satisfy the complex and varying multi-tenant environments where all the co-located tenants compete for the shared DPU's internal resources. This competition and interference will inevitably result in degraded and fluctuating performance [15], [16]. In this paper, we focus on DPU resource isolation and fair allocation under complex and varying multi-tenant environments. We propose a virtual multi-channel DPU offloading architecture for multiple tenants (MTDA) and implement it on a typical multicore SoC-based DPU platform Nvidia BlueField-2. MTDA maximizes the offloading benefits by fairly assigning DPU resources to respond to each tenant's requests, according to the requirements of application workloads. First, MTDA establishes an independent virtual channel for individual tenants to ensure isolation and mitigate the potential mutual interference among them. Offloading requests will be inserted into corresponding virtual channels based on the $Tenant\ ID$ before they are submitted to the hardware for execution. Second, for fair resource allocation, MTDA abstracts DPU computing resources into credits and quantifies the hardware resources to be allocated or transferred as the number of credits changes. Based on their specific requirements, MTDA allocates corresponding credits to each tenant. Third, we design a traffic-aware scheduling algorithm that can adapt to varying traffic characteristics and dynamic traffic behaviors by assigning appropriate time slices.

The main contributions of our work are as follows.

- We propose a virtual multi-channel DPU offloading architecture, called MTDA, which provides stable and fair allocation of the DPU domain's internal resources for each tenant. MTDA separates the unified management of all offloading requests in DPU and builds an independent virtual channel for each tenant to submit its offloading requests, which prevents internal competition and interference of requests among tenants.
- We develop a credit-based resource allocation model that abstracts DPU resources to credit values and allocates credits for each tenant according to its actual requirements. Additionally, we also design a traffic-aware scheduling algorithm capable of adapting to diverse traffic characteristics by monitoring the entire credit allocation process and reallocating credits dynamically.
- We implement MTDA on the BlueField-2 DPU platform and conduct a series of experiments with various benchmarks (including both balanced and unbalanced workloads) to compare our method with four DPU offloading frameworks (DPDK, iPipe, FairNIC, and LogNIC) from four aspects, i.e., throughput, latency, fairness, and stability. For a fair comparison, we also implement them on the BlueField-2 DPU platform. The evaluation results demonstrate that MTDA enhances the throughput, reduces the latency, and improves the fairness and stability of DPU resource allocation.

TABLE II
COMPARISON AMONG DIFFERENT DPU DESIGNS

Processor	FPGA	ASIC	SoC
Cost	High	Low	Moderate
Programmable	Difficult	Limited	Straightforward
Usability	Hard	Moderate	Easy
Flexibility	Moderate	Low	High
Products	e.g. Mellanox Innova [20]	e.g. Intel Mount Evans [21]	e.g. Nvidia BlueField [19]

The rest of this paper is organized as follows. Section II describes the background and motivation. Section III illustrates the design of MTDA in detail. Section IV describes the implementation and portability of MTDA. Experimental results are demonstrated in Section V. Section VI introduces related work. Finally, Section VII concludes the paper and presents our future work.

II. Background and Motivation

A. Data Processing Unit

Since the concept and technical standards of DPU are not unified at present, the hardware design architecture of DPU is diverse. From the perspective of core processors, DPU can be categorized into three types which are FPGA-based, ASIC-based, and SoC-based designs. Table II shows the comparison between different DPU architectures.

FPGA-based DPUs usually combine hardware programmable FPGAs with ASIC network controllers. They offer flexibility and enhanced performance through parallel data flow processing. However, these systems tend to be costly and pose challenges in terms of programming [9]. Specifically, they require dedicated programmers with sufficient skills in Hardware Description Language (HDL) to fully leverage their capabilities. ASIC-based DPUs achieve higher efficiency by sacrificing the flexibility present in FPGA [13]. They are capable of running parallel workloads with minimal latency and executing specific tasks with highly optimized hardware structures. Nevertheless, their programmability is limited, making them less suitable for handling complex application scenarios [17]. Compared with the aforementioned two categories, multi-core SoC-based DPUs obtain the highest performance by adopting designs that mingle dedicated hardware accelerators with programmable processors [18]. They usually hold rich computing resources including

hardware accelerators, PCIe interface for host communication, multicore processors, and onboard memory. Separate from the host system, SoC-based DPUs are able to run their own operating systems (commonly Linux), making them easier to program [13] and obtain the maximum flexibility. Given these considerations, we select Nvidia BlueField-2 DPU [19] as the experimental platform and implement our virtual multi-channel DPU offloading architecture on it.

B. Multi-Tenant Cloud

Multi-tenant cloud is a cloud computing service model where multiple organizations or users collectively share a unified pool of infrastructure and computing resources. Tenants within this environment can leverage the resources provided by Cloud Service Providers (CSPs) to deploy and execute their applications or services. The emergence of multi-tenant cloud models not only streamlines management processes and reduces operational costs but also offers flexible and scalable cloud computing services. Cloud resources for multi-tenant environments are characterized by several key features, including 1) Resource sharing: computing resources are shared among diverse tenants, encouraging collaborative utilization while also fostering competition for access to shared hardware infrastructure [22]. 2) Pay-as-you-go billing: cloud computing system charges based on the actual resource consumption of tenants [23]. This pay-as-you-go billing model ensures users are billed for the precise quantity of resources they consume, facilitating elastic resource scaling and aligning costs with usage levels. 3) Dynamicity: the computing demands of cloud data centers undergo constant real-time fluctuations [24]. For example, during peak periods such as year-end promotions, the server load experienced by e-commerce platforms may escalate significantly in comparison to standard operating periods. Consequently, the system design should offer resource isolation to mitigate tenant competition and provide dynamic configuration and traffic-aware scheduling to accommodate fluctuations in tenant resource demands, thereby ensuring optimal performance and fair resource allocation within the system.

C. Motivation

Although current DPU offloading frameworks have gained decent performance, several research challenges need to be addressed in dynamically offloading computation applications of a multi-tenant system into DPU such as Bluefield-2. We will demonstrate these challenges from two aspects as follows.

1) Competition among tenants: Considering the investment and cost, today's cloud providers usually don't allocate physical hardware to a tenant individually. The offloaded tasks of tenants will share and compete with the internal resources of the DPU domain [11], which results in performance degradation of the whole system. Fig. 1 shows the experimental results of using native DPDK to offload four typical applications in datacenters (real-time analytics [10], flow monitor [25], IPv4 router [26] and firewall [27]), the detailed experimental setup is described in Section V-A. It can be seen that the throughput of the whole system plummets with the increase in the number of tenants.

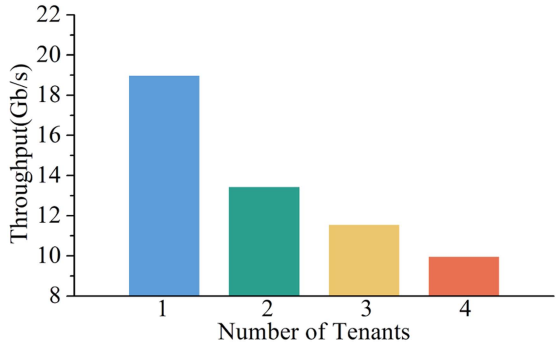


Fig. 1. Performance declines as tenants increase.

iPipe [3] alleviates this trend by scheduling requests with a hybrid scheduler, but it does not fundamentally solve the problem of request competition among tenants (as shown in Fig. 4). FairNIC [11] achieves fairness through the static isolation and allocation of resources, but it is not suitable for unbalanced load situations or scenarios with dynamic traffic changes (as shown in Fig. 5). LogNIC [12] partitions computing resources into multiple virtual instances and independently characterizes the bandwidth to handle resource contention issues. However, as an offline strategy, it also fails to address the dynamicity of tenant resource demands (as shown in Fig. 13). To eliminate the request interference among tenants, we first build an independent virtual channel for each tenant in DPU to differentiate between the offloading requests from different tenants, which can be directed to their separate virtual channel, and then allocate appropriate DPU computing resources for each tenant according to its resource demands to process the offloaded requests.

2) Fair allocation of DPU computing resources for multi-tenants: Emerging DPUs enclose rich computing resources including a variety of hardware accelerators. It is a challenge to fairly allocate these hardware resources for various offload requests of tenants to achieve high and stable performance. Fig. 2 depicts the performance distribution of DPDK, iPipe, FairNIC, and LogNIC when offloading four applications simultaneously. We can observe that the throughput and latency of native DPDK and iPipe fluctuate drastically over time, which means that they cannot maintain stable performance in a multi-tenant environment. Although FairNIC and LogNIC guarantee small performance fluctuations through strict isolation, they fail to achieve fair resource allocation, leading to significant performance differences among tenants. Moreover, their unfairness exhibits a notable upward trend with the increase in the number of tenants (experimental results are shown in Fig. 10). Therefore, we build a credit-based resource allocation model to abstract the DPU computing resources and assign credits for each tenant according to its actual demands. A traffic-aware scheduling algorithm is also proposed to adapt to the varying traffic behaviors.

III. Design

In this section, we first formalize the definition of fairness to pinpoint the root cause of unfairness and clarify our design

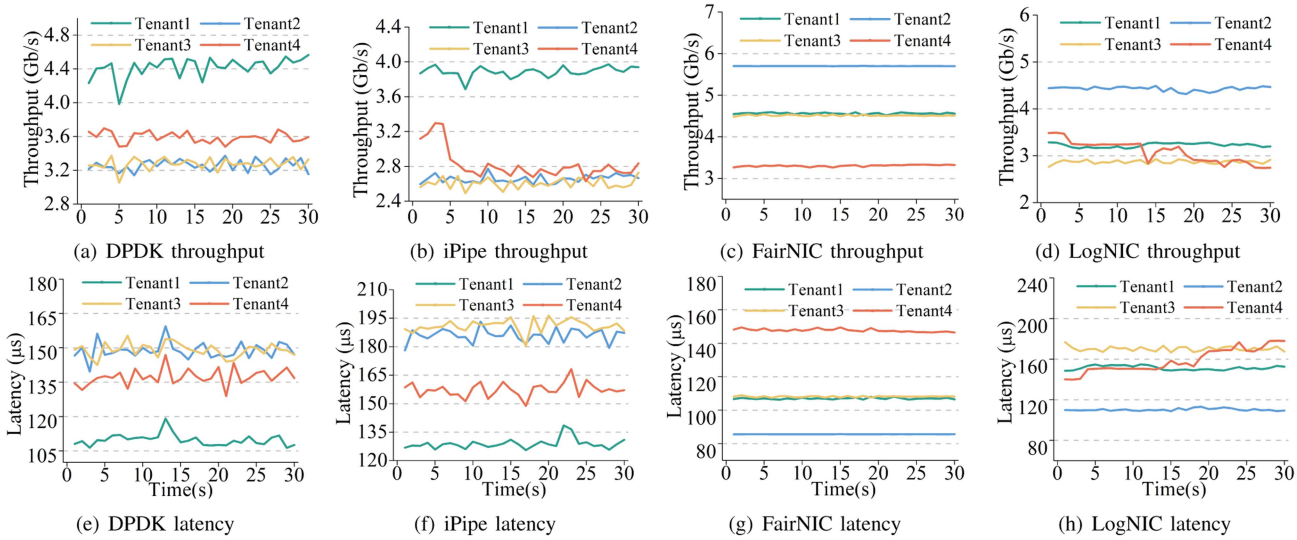


Fig. 2. Performance of each tenant fluctuates over time.

TABLE III
LIST OF PARAMETERS

	Notation	Description
Definition of Fairness	N	Number of tenants
	$Fair_1$	Variance between expected and actual weight
	w_i^{ep}	Expected weight of $Tenant_i$
	w_i^{real}	Actual weight of $Tenant_i$
	T_i	Throughput during interval $[t_1, t_2]$
	T	The overall system throughput
	$Fair_2$	Ratio of min and max slowdowns
	RL_i^{Alone}	Latency when executed individually
	RL_i^{Shared}	Latency during concurrent execution
	$Slow_i$	Slowdown caused by interference
System Model	CM	The quantity of programmable processor
	CM_i	Multicore processor occupancy of $Tenant_i$
	M	The number of hardware accelerators
	ACC_t^i	Whether $Tenant_i$ relies on accelerator t
	Req_i	The requests number from virtual channels
	$CREDIT$	The total amount of credit
	$Credit_i$	Credit value of $Tenant_i$

goals. Then we introduce the design specifics of our fair DPU offloading architecture, namely MTDA, a system that effectively separates the mixed offloading requests of tenants and ensures stable performance and fair resource allocation.

A. Definition of Fairness

The assessment of fairness in resource allocation can be defined in various ways, with different definitions being suitable for different scenarios. In this subsection, we introduce two typical definitions of fairness to clarify the underlying reasons for unfairness and establish theoretical foundations to guide the subsequent design of fair resource allocation strategy. The detailed descriptions of each notation used in this paper are summarized in Table III.

First, we follow the definition of fairness in research [28] and [29], which is defined as the variance between the expected weight and the actual weight of $Tenant_i$. We denote $Fair_1$ as the fairness index of DPU resource allocation and assume that the throughput of $Tenant_i$ during the interval $[t_1, t_2]$ is T_i. Consequently, the overall system throughput is denoted as $T = \sum_{i=1}^{N} T_i$ and the measured weight of $Tenant_i$ is computed as $w_i^{real} = T_i/T$. Therefore, the fairness index $Fair_1$ can be calculated as follows:

$$Fair_1 = \sum_{i=1}^{N} \left| w_i^{ep} - w_i^{real} \right|, \tag{1}$$

where N is the number of tenants, w_i^{ep} means the expected weight of $Tenant_i$, w_i^{real} represents the actual weight. A smaller value of $Fair_1$ indicates better fairness because it means that the difference between the actual throughput and the fairly allocated throughput is smaller.

Second, when multiple tenant requests run simultaneously, they can negatively interfere with each other. To further assess the impact of resource contention on performance and fairness, and understand how the scheduler influences the interference, we follow the definition of [30], [31] to redefine the fairness index $Fair_2$. The average slowdown $Slow_i$ for each tenant $Tenant_i$ is defined as follows:

$$Slow_i = \frac{RL_i^{Shared}}{RL_i^{Alone}}, \tag{2}$$

where RL_i^{Alone} represents the request latency while running the tenant $Tenant_i$ by itself and RL_i^{Shared} means the request latency while $Tenant_i$ runs concurrently with other tenants. The slowdown values, denoted as $Slow_i$, express the difference between these two scenarios. We define fairness as the ratio between the minimum slowdown value and the maximum slowdown value, which is formulated as follows:

$$Fair_2 = \frac{\min_i\{Slow_i\}}{\max_i\{Slow_i\}}. \tag{3}$$

The fairness index $Fair_2$ varies from 0 to 1, where a higher value indicates better fairness. If co-located tenants experience

the same slowdown due to interference (i.e., the value of $Fair_2$ equals 1), we consider the resource allocation strategy to be completely fair.

B. System Model and Problem Formulation

Considering a multi-tenant cloud environment with N tenants, where co-located tenants share and compete for the DPU domain's internal computing resources, including multicore processors and hardware accelerators. This resource contention and interference will inevitably lead to degraded and fluctuating performance. To alleviate the competition and ensure fairness among tenants, it is essential to implement an efficient allocation strategy.

Denote the number of programmable processors as CM, and CM_i represents the multicore processor occupancy of $Tenant_i$. Since various applications need specific hardware accelerators to perform, we denote ACC as the tenant dependency on them, and the number of accelerators is M. Thus, ACC_t^i represents whether $Tenant_i$ relies on accelerator ACC_t (1 indicates true, otherwise the value is 0). Our objective is to achieve optimal computing resource allocation among tenants, minimizing fairness index $Fair_1$ (lower is better), and maximizing $Fair_2$ (higher is better). Similar to research [32], we construct an optimization problem model, which can be formulated as follows:

$$\max \left[Fair_2\left(CM_i, ACC_t^i\right) - Fair_1\left(CM_i, ACC_t^i\right)\right] \quad (4)$$

$$\text{s.t.} \quad \sum_{i=1}^{N} CM_i \leq CM, \quad \forall i, \quad (5)$$

$$\sum_{t=1}^{M} ACC_t^i \leq M, \quad \forall i, t. \quad (6)$$

Equations (5) and (6) impose constraints on resource allocation policy, ensuring that allocated hardware resources should not exceed the total resource capacity.

Based on the discussion on the definition of fairness presented in the preceding subsection, we can deduce two critical factors contributed to unfairness: 1) the deviation between the ideal and actual weights results from improper resource allocation, and 2) performance slowdown caused by mutual interference among co-located requests. Therefore, we can transform the fairness-enhancing problem into two specific objectives: 1) considering the actual resource demands of each tenant ($min[Fair_1]$), and 2) alleviating the mutual competition among tenants for shared resources ($max[Fair_2]$). To address this, we propose MTDA, an innovative virtual multi-channel DPU offloading architecture designed to facilitate stable and fair allocation of the DPU domain's internal resources. The architecture of MTDA is illustrated in Fig. 3.

First of all, we implement an independent virtual channel for each tenant to ensure performance isolation and stability. Offloading requests from multiple tenants are divided into independent virtual channels to avoid competition and interference with each other before they are submitted to the hardware. Subsequently, for fair resource allocation, MTDA incorporates

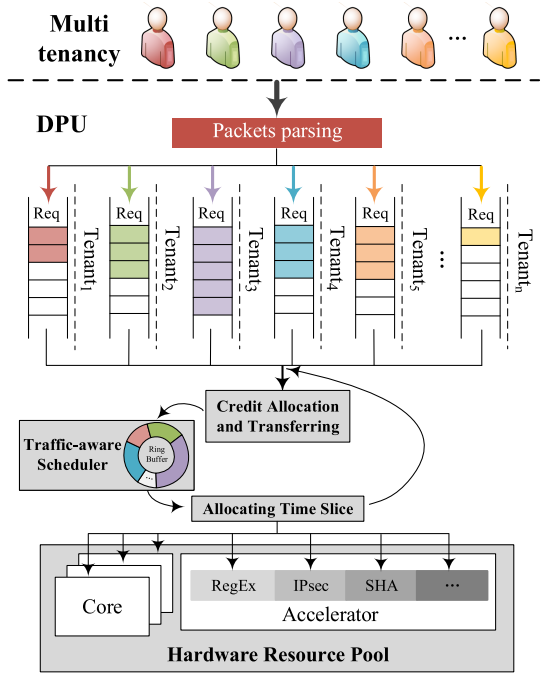


Fig. 3. MTDA architecture.

a credit-based resource allocation model that abstracts DPU resources as credits and assigns credits to each tenant according to their actual requirements. The credit value of each tenant, which presents the abstract of hardware resources, is allocated and bound to the corresponding virtual channel according to the credit allocation model. Finally, requests within the virtual channels are dispatched to the physical hardware accelerator via a traffic-aware scheduler, which allocates time slices according to the actual demand of each tenant. When tenants contend for the same accelerator during a period, the requests of each tenant are sequentially sent to the ring buffer we establish. Then the accelerator will consume these requests by fetching them from the ring buffer. Further details will be provided in the subsequent subsections.

C. Fair Resource Allocation Based on Credit Model

1) Independent virtual channels: In today's data centers, servers often host applications from multiple tenants on shared physical DPU resources, leading to potential competition and resource contention among them [33]. For instance, consider a scenario where two tenants are co-located on the same server. If Tenant A operates a compute-intensive application, it might dominate the available resources, causing Tenant B to experience head-of-line blocking. Therefore, it is essential to implement robust performance isolation mechanisms to effectively manage tenant competition and ensure fair resource allocation among tenants.

In order to ensure the stability and security of multi-tenant systems in complex shared resource environments and to mitigate potential interference among tenants, we implement an independent virtual channel for each individual tenant to achieve

resource isolation. For clear identification, MTDA assigns a unique $Tenant\ ID$ to each tenant and associates tenant requests with their corresponding virtual channels through tenant ID allocation. Virtual channels are isolated from each other, effectively preventing unnecessary interference or interaction among tenants. The offloading requests will be received through dedicated Virtual Functions (VFs) and inserted into the appropriate virtual channel based on the assigned $Tenant\ ID$. Subsequently, these requests are submitted to suitable hardware computing units, such as various hardware accelerators or multicore processors, for execution.

2) Credit model: Based on the principles of credit-based scheduling, a technique widely employed to ensure fairness in CPU resource allocation within the Xen hypervisor [34], we extend the concept to facilitate fair DPU resource allocation in MTDA. We introduce a credit-based resource allocation model that abstracts DPU computing resources into credits. These credits are distributed fairly among tenants, taking factors such as the number of requests, the quantity of hardware accelerators, and the number of programmable processors into consideration. Inspired by the findings in study like [35], which indicate that hardware resources allocated to virtual machines (VMs) tend to follow an exponential distribution, we express the credit value assigned to $Tenant_i$ as follows:

$$Credit_i = \frac{e^{\sum_{t=1}^{M} ACC_i^i} \times e^{CM_i} + Req_i}{e^{\sum_{t=1}^{M} ACC_t} \times e^{CM} + \sum_{j=1}^{N} Req_j} \times CREDIT, \tag{7}$$

where Req_i represents the number of requests from $Tenant_i$'s virtual channel, and $CREDIT$ means the total amount of credits, i.e., the abstraction of the overall hardware resources in DPU.

According to (7), we quantify the hardware resources to be allocated or transferred as the number of credits changes (i.e., the amount of credits increases or decreases). By predicting the credit weight of each virtual channel, we assign proper time slices to each tenant and achieve efficient time division multiplexing of physical hardware resources. However, the required context switches might result in increased latency. Therefore, if the available multicore processors can satisfy the tenant demands, we allocate certain amounts of cores to each tenant to avoid frequent context switching. Through credit allocation, DPU resources can be fairly exploited by all tenants, effectively alleviating DPU resource competition among tenants.

D. Traffic-Aware Scheduling

The credit-based resource allocation model effectively assigns the necessary hardware resources to each tenant based on their requirements. However, system performance can be further improved with an appropriate scheduling algorithm. Considering different tenants have various traffic characteristics, and even the traffic of the tenant itself will change dynamically over time in different periods, we design a traffic-aware scheduling algorithm to apply to this situation.

The traffic-aware scheduling framework encompasses two principal components: the Monitoring Unit and the Reallocating Unit. The monitoring unit monitors the entire credit allocation process and records the status of virtual channels. Tenant queues will become inactive when one of the following conditions are met: 1) EXPIRED: the allocated credits will expire by the end of a scheduling period, 2) EXHAUSTED: all credits in the tenant's virtual channel are exhausted and new requests arrive. These indicators will be provided as the input of the second Reallocating Unit, as shown in Algorithm 1. EXPIRED and EXHAUSTED denote the two conditions for deactivating a queue, and $currentCredit[i]$ means the current credit value of $Tenant_i$. If the reason for queue deactivation is EXPIRED, we consider there are remaining credits for each tenant and renew the value of current credits as the last scheduling period (Step 1 to 3). However, if the reason for deactivation is EXHAUSTED, we regard the tenant application to have been too high-intensity, i.e., the request generation rate becomes higher and triggers the credit reallocation process according to (7) (Step 4 to 10).

Algorithm 1: Credit Reallocation Algorithm.

EXPIRED: The current scheduling period ends and there are remaining credits
EXHAUSTED: All credits are exhausted and new requests arrive
1) **int** CreditReallocation(int $^*req$, int $^*acc$, int $^*core$)
2) **if** EXPIRED **then**
3) $currentCredit = credit$;
4) **else if** EXHAUSTED **then**
5) /* Trigger the credit reallocation process based on (7) */
6) $credit =$ creditAllocation(req, acc, $core$);
7) $currentCredit = credit$;
8) **end if**
9) **return** $currentCredit$;
10) **End**

The procedure of traffic-aware fair scheduling is depicted in Algorithm 2. Overall, in order to adapt to the traffic dynamicity and achieve fair DPU resource allocation, MTDA takes the actual resource requirements of each tenant as crucial inputs and generates optimal fair resource allocation results. The time complexity of Algorithm 2 is $O(N \cdot max(Req_i))$, where N is the number of tenants, and $\max(Req_i)$ represents the maximum number of tenant requests Req_i. The main steps are as follows. MTDA traverses the virtual channel of each tenant, if spare credits and pending requests exist, the algorithm dispatches requests from the virtual channel and decreases the corresponding value of $req[i]$ and $currentCredit[i]$ (Step 1 to 6). Otherwise, if all the credits of $Tenants_i$ are exhausted or the current scheduling period ends, MTDA sets the corresponding indicators as 1, triggers the credit reallocation process, and continues to the next scheduling period (Step 7 to 21).

IV. Implementation and Discussion

In this section, we present the implementation details of the MTDA framework on the BlueField-2 DPU platform and discuss the architecture independence and portability of MTDA.

Algorithm 2: Traffic-Aware Fair Scheduling Algorithm.

Input: Request number of each tenant $req[N]$, accelerator dependency $acc[N][M]$, and multicore processor occupancy $core[N]$
Output: Fair resource allocation results $credit[N]$

```
1) for i = 1 to N do
2)   while req[i] > 0
3)     if currentCredit[i] > 0 then
4)       /* Dispatch requests from the virtual channel */
5)       req[i] − −;
6)       currentCredit[i] − −;
7)     else
8)     /* All the credits of Tenant[i] are exhausted */
9)       EXHAUSTED = 1;
10)      credit = CreditReallocation(req, acc, core);
11)      /* Continue the next scheduling round */
12)      return credit;
13)    end if
14)  end while
15) end for
16) /* By the end of the scheduling period */
17) EXPIRED = 1;
18) credit = CreditReallocation(req, acc, core);
19) /* Continue to the next scheduling period */
20) return credit;
21) End
```

MTDA implementation on BlueField-2 DPU: BlueField-2 DPU is the latest generation DPU product of NVIDIA, which belongs to the SoC-based designs. It is equipped with eight 64-bit ARMv8 A72 cores, 16 GB of DDR4 RAM, 64 GB of on-board eMMC Memory, and a variety of specific hardware accelerators, which allows us to install an individual operating system to flexibly deploy the offloaded applications or protocols. In this paper, we install Ubuntu 20.04 as the operating system and implement the MTDA protocol in Data-Center-Infrastructure-On-A-Chip-Architecture (DOCA) [36] which is a highly programmable SDK development platform tool for NVIDIA-Mellanox's DPU. Both the core component and application development of MTDA are implemented using the C programming language with a total of 11480 lines of code (LOC). Specifically, we develop four applications for experimental evaluation, including real-time analytics, flow monitor, IPv4 router, and firewall, with 2292 LOC, 2431 LOC, 1348 LOC, and 1219 LOC respectively.

Architecture independence and portability discussion: Although MTDA is implemented on BlueField-2 DPU in this paper, it can also be suitable for other DPUs because specific hardware accelerators are the essential components. In addition, the core component of MTDA is platform agnostic. While we implement it in DOCA currently, we can easily extend MTDA to other SoC-based DPUs, such as MIPS architecture, or port it to FPGA-based DPUs. Moreover, we might also apply MTDA to other architectures in multi-tenant scenarios in addition to DPUs, which will be explored in our future work.

V. Evaluation

In this section, we run a set of experiments to evaluate the performance of MTDA on our BlueField-2 DPU platform by using a variety of workloads we have mentioned in Section II-C, which demonstrate that MTDA has several advantages in throughput, latency, fairness, and stability compared with other offloading methods, including Native DPDK, iPipe [3], FairNIC [11], and LogNIC [12]. We also implement them on the BlueField-2 DPU for comparison.

A. Experimental Setup

1) Experimental platform: Similar to FairNIC [11], we also emulate a simple cloud environment with two Intel servers, which are equipped with an NVIDIA BlueField-2 25-Gbps DPU and a regular Intel E810 25-Gbps NIC respectively. Both of them support an Intel 8171M 52-core processor running at 2.6 GHz, 128 GB RAM, and 512 GB NVMe SSD. We install Ubuntu 20.04 in the servers and instantiate tenants in VMs by KVM and SR-IOV. The server with regular NIC is used to generate workloads of different formats by DPDK Pktgen [37], and connected to the server with BlueField-2 DPU via an SFP28 cable.

2) Workloads: We select four typical data center applications, that present both compute-intensive and memory-intensive behaviors to fully evaluate the heterogeneous platform.

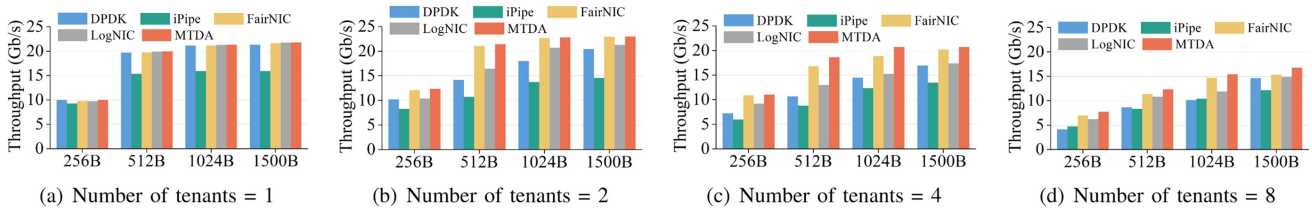
Real-time Analytics: Real-time analytics (RTA) [10] is a data processing approach that involves real-time analysis and extraction of valuable insights and patterns from extensive datasets. Significant data ingestion and processing make it compute-intensive. Additionally, as it involves large volumes of data access and management, RTA is also memory-intensive.

Flow Monitor: Flow Monitor [25] performs the capture and analysis of network flows, with the primary goal of identifying abnormal traffic patterns and network issues. It offers real-time insights into network performance and security. Notably, this process is highly memory-intensive, especially when dealing with substantial volumes of network data.

IPv4 Router: IPv4 router [26] is responsible for routing IPv4 packets through the internet. It functions by receiving packets and forwarding them from the source address to the intended destination. Complex operations like packet forwarding and address translation make it compute-intensive.

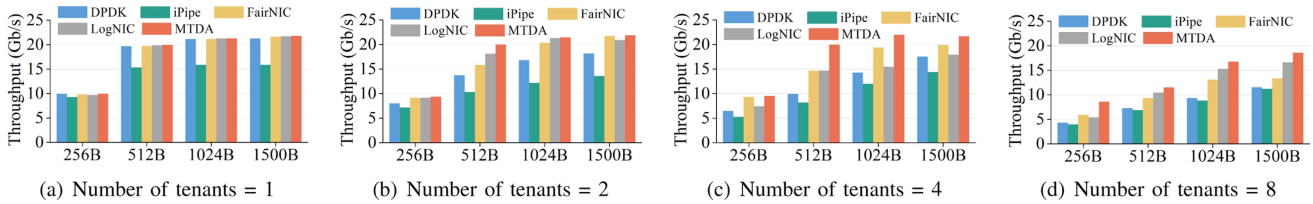
Firewall: Firewall [27] involves the inspection and filtering of network traffic to enforce security policies. This operation is computationally intensive because it involves analyzing each incoming packet for potential security threats and comparing it against a set of predefined rules to determine whether it should be allowed or blocked.

Workload Configuration: To comprehensively assess MTDA's performance, we expand the number of tenants from 1 to 8 and design two sets of workloads (balanced and unbalanced workloads). In the cloud computing environments, it is common for VMs holding aggressive applications (e.g., video, Hadoop) with relatively high request arriving speeds, to run together with VMs holding non-aggressive applications like web and mail [38]. To simulate this common situation,



(a) Number of tenants = 1 (b) Number of tenants = 2 (c) Number of tenants = 4 (d) Number of tenants = 8

Fig. 4. Throughput of balanced workload.



(a) Number of tenants = 1 (b) Number of tenants = 2 (c) Number of tenants = 4 (d) Number of tenants = 8

Fig. 5. Throughput of unbalanced workload.

TABLE IV
CONFIGURATION OF BALANCED AND UNBALANCED WORKLOADS

Test case	Tasks of each VM
NOT-1	rta (1)
NOT-2	rta (4) + fm (1)
NOT-4	rta (4) + fm (2) + router (2) + fw (1)
NOT-8	4rta (4) + 2fm (2) + router (2) + fw (1)

* Note that NOT is short for the number of tenants.
* Numbers in the brackets represent traffic generation rates for tenants under unbalanced workloads.

we assign different request generation rates to tenants under unbalanced workloads. The configuration details are outlined in Table IV. We deploy specific applications on each tenant and allocate varying traffic generation rates for them in the case of unbalanced workloads.

B. Throughput

We first evaluate the overall system throughput under balanced and unbalanced workloads. Experimental results in different packet sizes are shown in Figs. 4 and 5.

Since resource allocation strategy is not triggered in scenarios involving a single tenant, MTDA achieves similar throughput compared with other methods, as shown in Figs. 4(a) and 5(a). Moreover, with the increase in packet size, the overall system throughput generally exhibits an increasing trend. iPipe requires additional resource overhead to manage migration tasks between hosts and DPUs, which inevitably impacts its performance. While FairNIC demonstrates reasonable performance when confronted with balanced workloads, it exhibits unsatisfactory performance in scenarios involving unbalanced workloads due to its adoption of static resource partitioning strategies. LogNIC abstracts offloaded programs as directed acyclic graphs and makes resource allocation decisions based on factors such as computation transfer overhead and potential queueing delay. However, it overlooks the actual resource requirements of each tenant. Overall, when dealing with multiple tenants, MTDA presents a superior performance of throughput than the other four methods, especially for unbalanced workloads. Take Fig. 5(c) as an example, MTDA improves the throughput by up to 101.2%, 143.2%, 36.1%, and 41.7% respectively compared to DPDK, iPipe, FairNIC, and LogNIC. This significant enhancement in throughput is attributed to MTDA's unique characteristics. Through the implementation of independent virtual channels, MTDA effectively mitigates resource competition among tenants and provides optimized system performance.

C. Latency

Figs. 6 and 7 provide a comprehensive evaluation of latency in both balanced and unbalanced workloads. They show a similar trend as the experimental results of throughput. As shown in Fig. 7, for the four tenants operating under unbalanced workloads, as the packet size increases from 256B to 1500B, the latency gradually decreases for all five methods. This trend is primarily due to the decrease in the number of packets processed per second as the packet size grows. MTDA consistently maintains the lowest latency across various packet sizes and reduces the latency by up to 50.3%, 58.9%, 26.6%, and 29.4% respectively compared to DPDK, iPipe, FairNIC, and LogNIC. The reason is that MTDA guarantees the appropriate allocation of hardware resources based on the individual requirements of each tenant, thereby reducing queueing delay and request processing time. The performance experimental results underscore MTDA's capability to achieve nearly line-rate processing within multi-tenant environments.

p99 tail latency: For generic cloud applications such as social networking or search, the end-to-end latency is primarily determined by the slowest operations, which are often referred to as the request tail latency [39], [40]. In this subsection, we measure the tail latency at the 99th percentile for four tenants under the packet size of 1024B, results are displayed in Fig. 8.

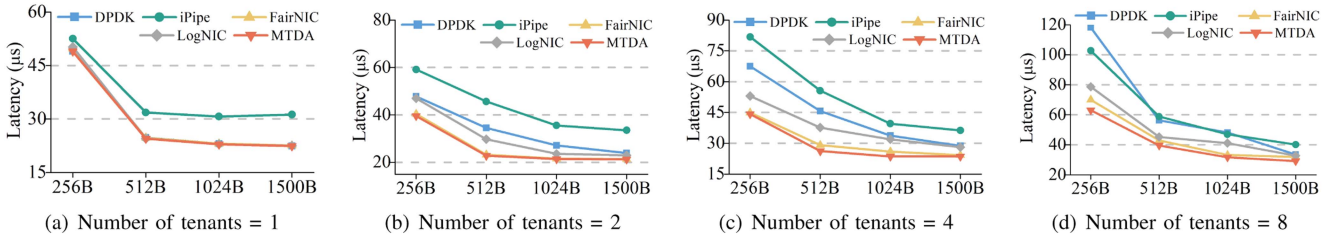


Fig. 6. Latency of balanced workload.

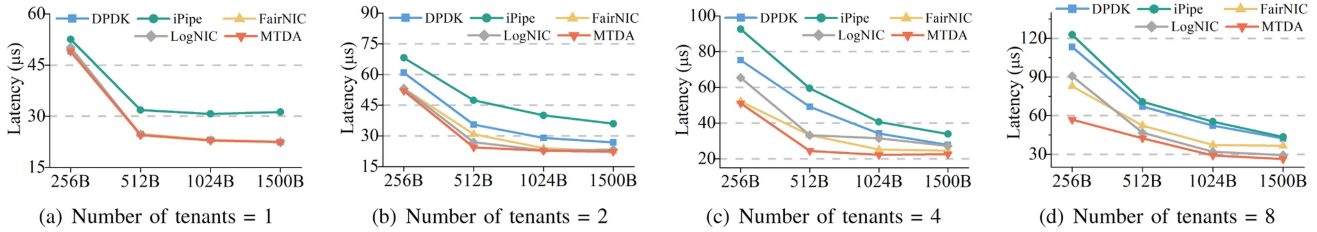


Fig. 7. Latency of unbalanced workload.

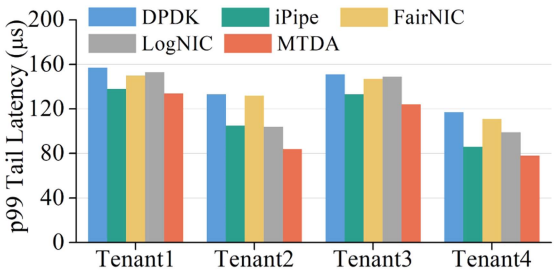


Fig. 8. p99 tail latency of each tenant.

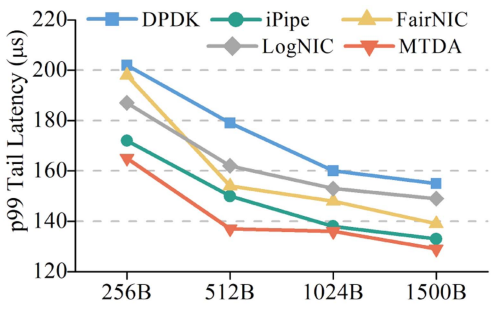


Fig. 9. Overall system p99 tail latency.

iPipe effectively reduces the tail latency by directing requests into a DRR runnable queue when latency exceeds the pre-defined threshold. However, compared with DPDK, iPipe, FairNIC, and LogNIC, MTDA significantly lowers the tail latency by up to 36.8%, 20.0%, 36.4%, and 21.2% respectively. Through resource isolation and fair allocation, MTDA alleviates significant queue build-up caused by resource contention, leading to reduced end-to-end response times and enhanced user experience.

To further validate MTDA's latency-reduction capabilities, we assess the overall system p99 tail latency for four tenants across various packet sizes. As illustrated in Fig. 9, we achieve up to 23.5%, 14.5%, 16.7%, and 15.4% 99th percentile latency savings compared to DPDK, iPipe, FairNIC, and LogNIC, respectively. The results not only emphasize the effectiveness of MTDA but also point to its potential in addressing real-world cloud computing challenges where latency and response times are critical metrics.

D. Fairness

In this subsection, we evaluate the fairness index of DPDK, iPipe, FairNIC, LogNIC, and MTDA according to (1) and (3), respectively.

First of all, we measure the results of the fairness index $Fair_1$, which depicts the difference between the expected weight and the actual weight of $Tenant_i$, with a lower value of $Fair_1$ indicating superior fairness. Take the throughput of DPDK and MTDA in Fig. 4(c) as an example, for the balanced workload, the expected weight W^{ep} should be equal to $[1/4, 1/4, 1/4, 1/4]$, and the actual throughput of DPDK and MTDA is (4.349, 3.284, 3.273, 4.439) and (5.288, 5.342, 5.343, 5.045) respectively. So we can calculate the measured weights of DPDK and MTDA, which are $W^{real}_{dpdk} = [4.349/15.345, 3.284/15.345, 3.273/15.345, 4.439/15.345] = [0.283, 0.214, 0.213, 0.289]$ and $W^{real}_{mtda} = [5.288/21.018, 5.342/21.018, 5.343/21.018, 5.045/21.018] = [0.252, 0.254, 0.254, 0.240]$ respectively, and obtain the fairness index $Fair_1^{dpdk} = 0.145$ and $Fair_1^{MTDA} = 0.02$.

Fig. 10 illustrates the results of $Fair_1$ of five methods for the balanced workload in the packet size of 1024B. It can be observed that fairness gradually decreases as the number of tenants increases. This trend may be attributed to the intensified resource competition among tenants as their numbers grow. Since each tenant will receive fewer resources, leading to a

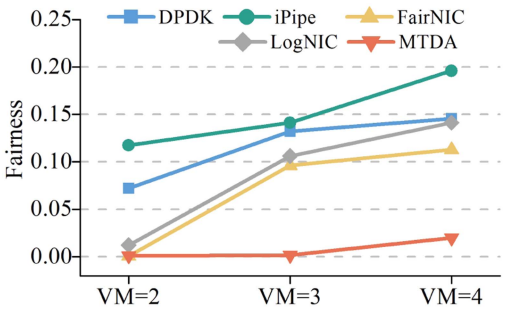


Fig. 10. Fairness index results of $Fair_1$, which depict the difference between the expected weight and the actual weight of $Tenant_i$, with a lower value of $Fair_1$ indicating superior fairness.

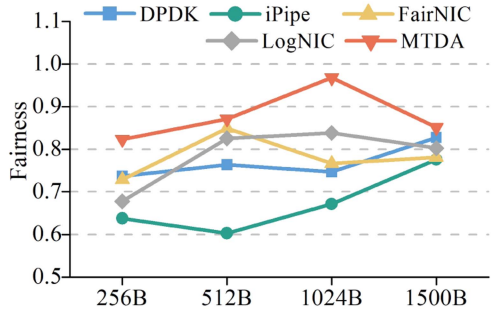


Fig. 11. Fairness index results of $Fair_2$, which denote the ratio of the minimum and maximum slowdowns, where a higher value indicates better fairness.

less fair resource allocation. Compared with DPDK, iPipe, and LogNIC, the fairness index of MTDA is improved by up to 98.8%, 99.0%, and 98.4% respectively for multiple tenants. Although MTDA obtains a similar fairness result to FairNIC for two tenants, it promotes the fairness index by 98.3%, and 82.3% for three and four tenants respectively. This improvement is due to the static resource allocation strategy in FairNIC, which fails to meet the diverse resource requirements of multi-tenants. In contrast, through the credit-based model and traffic-aware scheduling, MTDA dynamically allocates computing resources based on the actual needs of tenants, thereby reducing the weight deviation between ideal and reality.

To demonstrate the benefits of MTDA in reducing inter-tenant interference, we further measure the results of the fairness index $Fair_2$, which denotes the ratio of the minimum and maximum slowdowns, where a higher value indicates better fairness. Results of four tenants under various packet sizes are depicted in Fig. 11. We improve the fairness index by up to 29.6%, 44.4%, 26.3%, and 21.4% respectively. MTDA divides offloading requests from multiple tenants into independent virtual channels and effectively alleviates the performance slowdown caused by resource contention and mutual interference.

E. Stability

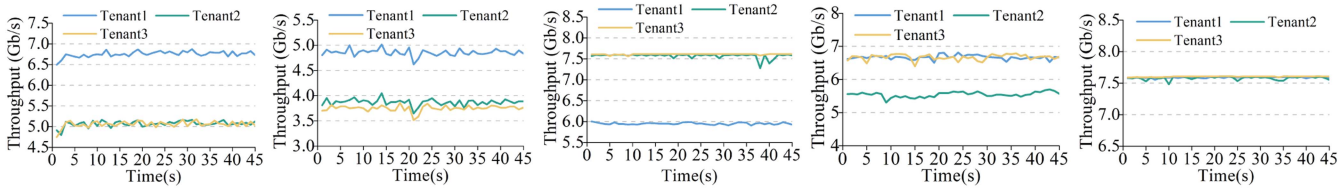
To demonstrate that MTDA has stable performance, we evaluate the stability of DPU resource allocation by measuring the throughput of three tenants in 45 seconds for balanced workloads. The packet size is set to 1024B. Fig. 12 presents the experimental results. As shown in Fig. 12(a) and (b), DPDK and iPipe reveal significant throughput fluctuations over time. This instability can be problematic for maintaining consistent performance levels. For FairNIC, while its throughput remains stable due to the strict resource isolation, Tenant 1's performance (indicated by the blue line in the figure) experiences a notable drop in performance due to its static resource allocation. LogNIC achieves performance isolation by partitioning computing resources into virtual instances. However, tenants fail to attain optimal performance due to unfair resource allocation. MTDA, on the other hand, not only ensures stable performance but also accomplishes nearly line-rate packet processing within the multi-tenant environment, as shown in Fig. 12(e).

Stability under burst traffic: Unexpected burst traffic is a common phenomenon within the data center environment, presenting a potential risk of packet dropping and serious performance degradation [41]. To assess the efficiency of MTDA under extreme workload conditions, we measure the p99 tail latency fluctuations over 30 seconds under four tenants. As shown in Fig. 13, we initially employ a lightweight workload and intensify it at the 8th second to emulate a burst traffic situation. DPDK, FairNIC, and LogNIC experience noticeable latency increments since they all employ static resource allocation policies, which lack robust adaptive mechanisms capable of dynamically addressing unexpected situations. iPipe pushes requests to the DRR runnable queue when the tail latency exceeds the pre-defined threshold, effectively alleviating the surge in tail latency. However, it introduces additional request migration overhead. In contrast, through traffic-aware scheduling and credit reallocation, MTDA demonstrates an effective response to burst traffic and unexpected scenarios, providing stable tail latency levels and enhanced overall performance.

F. Overhead of MTDA

To achieve stable and fair DPU resource allocation, MTDA introduces an independent virtual channel for individual tenants and implements a credit-based resource allocation model. The evaluation experiments demonstrate that MTDA effectively alleviates resource competition, and provides stable and enhanced performance. However, credit allocation and traffic-aware scheduling require continuous monitoring of the whole request processing and collection of relevant information mentioned in Section III-C. MTDA might also introduce some additional overhead and potentially impact the overall system performance.

To measure the overhead introduced by MTDA, we manually adjust the resource allocation strategies among tenants to achieve the possible optimal performance and compare it with MTDA. Fig. 14(a) show the throughput of MTDA and the manually-tuned version in the packet size of 512B varying the number of tenants from 2 to 4. Fig. 14(b) displays the average latency. As illustrated in the figure, when compared to the manually-tuned results, MTDA reduces the throughput by less than 4.9% and increases the latency by no more than 5.2%. We also evaluate the additional CPU cores and memory utilization



(a) DPDK (b) iPipe (c) FairNIC (d) LogNIC (e) MTDA

Fig. 12. Stability of DPU resource allocation in 3 tenants.

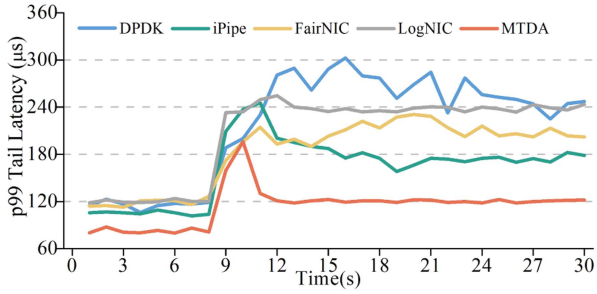


Fig. 13. Stability under burst traffic.

induced by MTDA, as shown in Fig. 14(c). MTDA exhibits a modest increase in CPU and memory usage, with an average rise of 3.3%, and 1.4%, respectively. The experimental findings indicate that the additional overhead caused by MTDA's fair resource allocation and traffic-aware scheduling is acceptable, in terms of both performance impact and implementation costs.

VI. Related Work

DPU acceleration: Researchers have done a lot of work to explore the potential of SmartNIC across various domains, including network [4], [5], [6], [42], [43], [44], storage [7], [8], [9], [45], [46], [47] and computation [3], [10], [48], [49] acceleration.

For reducing network overhead, AccelTCP [4] employs SmartNIC to accelerate the TCP protocol, offloading complex TCP operations like connection setup and teardown to free up valuable CPU cycles. Lynx [5] introduces an accelerator-centric network server architecture that offloads server data and control planes to the SmartNIC. UNO [6] studies how to make network function placement decisions between the NIC and the host. FlexNIC [42] provides a flexible DMA interface for network I/O, enabling operating systems and applications to offload stateless packet processing tasks. Clara [43] generates automated offloading insights for network functions, extracting features from network functions and predicting their performance characteristics on SmartNIC using machine learning techniques. FlexTOE [44] offloads all TCP data-path processing to SmartNIC and leverages the fine-grained parallelization of TCP data-path and segment reordering for exceptional performance.

To relieve cloud providers from heavy storage tax burden, LeapIO [9] presents a novel cloud storage stack that leverages ARM-based co-processors to offload complex storage services. LineFS [7] decomposes distributed file system (DFS) operations into execution stages that can be offloaded to a parallel data-path execution pipeline and offloads CPU-intensive DFS tasks like replication, compression, and data publication. Xenic [8] establishes a SmartNIC-optimized transaction processing system, and employs a co-designed data store distributed between the NIC and x86 host to enhance communication flexibility and efficiency. AINiCo [45] proposes an intelligent transaction processing system, which schedules transaction requests to different CPU cores to minimize inter-transaction contention.

In addition, researchers have also proposed several methods to make full use of the substantial computing resources on SmartNIC and move computation tasks onto it. Floem [10] provides a flexible programming system that simplifies the development of network applications. It offers a unified framework for implementing applications distributed between the CPU and NIC. iPipe [3] explores the computational capabilities of multicore SoC SmartNICs and proposes an actor-based framework for distributed application offloading to achieve host CPU and latency saving. E3 [48] offloads microservice-based applications to SmartNIC-accelerated server systems for energy efficiency.

To further optimize CPU resource allocation and alleviate the heavy data center tax burden, MTDA offloads generic data center tasks for multiple tenants and fairly allocates DPU internal resources according to their specific demands.

Task scheduling and resource allocation: To make full use of the multicore processors on DPU, several studies have concentrated on task scheduling and core allocation. IX [50] presents an operating system that separates the control plane from the data plane. The core of the IX control plane is a dynamic controller that adjusts the number of cores allocated to applications by monitoring CPU utilization. ZygOS [51] implements a work-conserving scheduler, introducing a single-producer, multiple-consumer shuffle queue for each core. This design allows idle cores to aggressively steal pending events. However, work stealing is not free and has associated costs. Shinjuku [52] implements preemptive scheduling at the microsecond scale, aided by hardware support for virtualization. It leverages preemption to select c-FCFS for low dispersion workloads and PS for all other cases based on observed service times. iPipe [3] proposes a hybrid scheduler that combines FCFS and DRR. Heavy-weight actors are executed on DRR cores, while lightweight actors run on FCFS cores. These studies primarily aim at the allocation of multicore processors and scheduling requests among multiple cores within DPUs. In contrast, MTDA focuses on multi-tenant deployment and scheduling resources among tenants including both cores and accelerators.

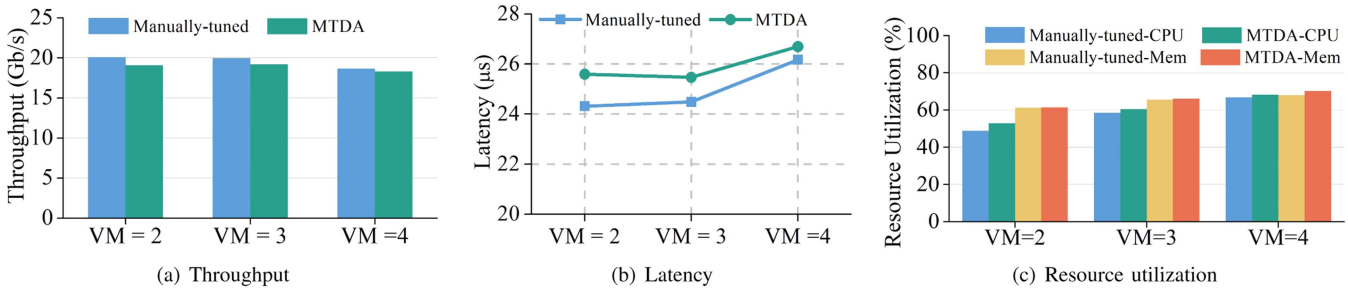


Fig. 14. Overhead of MTDA.

Multi-tenancy isolation: With the expansion of data centers, cloud providers have started to deploy services on a shared DPU, leading to increased competition for available hardware resources. Researchers have proposed several solutions to alleviate the competition and provide cross-tenant performance isolation. λ-NIC [13] aims to run serverless workloads (lambda) on SmartNICs. To achieve isolated memory access, it maps lambdas across different memory hierarchies within NICs based on their memory-access patterns. FairNIC [11] focuses on strict performance isolation, it isolates typical packet processing, core cycles, shared memory, and fixed-function coprocessor access. However, this static partitioning approach fails to adapt to the diverse performance requirements of multi-tenants, especially when their workloads are unbalanced. PANIC [33] enables priority scheduling on FPGA-based SmartNIC, utilizing a new hardware-based priority queue called PFIO, and allocates bandwidth according to flow priorities. Different from the above-mentioned approaches, MTDA aims at alleviating resource competition and achieving fair resource allocation based on the specific resource requirements of each individual tenant.

VII. Conclusion and Future Work

In this paper, we have proposed an efficient and fair DPU offloading method for multi-tenants called MTDA. It divides the unified management of all offloading requests into virtual multi-channels to avoid internal competition and interference of requests among tenants. By abstracting the DPU resources to credit values according to a credit-based resource allocation model, MTDA achieves the fair allocation of DPU resources for each tenant based on its actual needs. After the resource allocation, a traffic-aware scheduling algorithm is designed to dynamically submit each tenant's requests to hardware accelerators for execution. Evaluations using four typical data center applications show that MTDA increases the throughput by up to 101.2%, 143.2%, 36.1%, and 41.7%, lowers the latency by up to 50.3%, 58.9%, 26.6%, and 29.4%, improves the fairness by up to 98.8%, 99.0%, 98.3%, and 98.4%, and provides more stable performance for multi-tenants, compared with DPDK, iPipe, FairNIC, and LogNIC.

Future work: In the current design, we primarily focus on the fair allocation of computing resources, including multi-core processors and hardware accelerators. However, it is essential to acknowledge the significance of other critical resources within data centers. For instance, research Falloc [32] and SoftBW [23] highlight the importance of fair network bandwidth allocation. In future work, we aim to extend MTDA to ensure the fair allocation of other key resources, such as on-board memory and bandwidth. Additionally, our research effort is primarily conducted on a typical SoC-based DPU platform Nvidia BlueField-2. However, as a software-level design, we can easily extend MTDA to other SoC-based options such as the Cavium LiquidIO. Moreover, given the platform-agnostic nature of MTDA's core architecture, we also have the potential to extend MTDA to other types of DPUs, such as FPGA-based DPUs, or even other architectures within multi-tenant scenarios. We intend to pursue these extensions and explorations in our future work.

Acknowledgment

The authors are grateful to the anonymous reviewers for their helpful feedback.

References

[1] Z. Wang, H. Huang, J. Zhang, F. Wu, and G. Alonso, "FpgaNIC: An FPGA-based versatile 100Gb SmartNIC for GPUs," in *Proc. USENIX Annu. Tech. Conf.*, 2022, pp. 967–986.
[2] M. Cooney, "Speed race: Just as 400Gb ethernet gear rolls out, an 800GbE spec is revealed," 2020. [Online]. Available: https://download.intel.com/newsroom/2022/corporate/vision/Intel-IPU-Roadmap-Fact-Sheet.pdf
[3] M. Liu, T. Cui, H. Schuh, A. Krishnamurthy, S. Peter, and K. Gupta, "Offloading distributed applications onto smartNICs using iPipe," in *Proc. ACM Special Int. Group Data Commun.*, 2019, pp. 318–333.
[4] Y. Moon, S. Lee, M. A. Jamshed, and K. Park, "AccelTCP: Accelerating network applications with stateful TCP offloading," in *Proc. 17th USENIX Symp. Netw. Syst. Des. Implementation*, 2020, pp. 77–92.
[5] M. Tork, L. Maudlej, and M. Silberstein, "Lynx: A smartnic-driven accelerator-centric architecture for network servers," in *Proc. 25th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2020, pp. 117–131.
[6] Y. Le et al., "UNO: Uniflying host and smart NIC offload for flexible packet processing," in *Proc. Symp. Cloud Comput.*, 2017, pp. 506–519.
[7] J. Kim et al., "LineFS: Efficient smartnic offload of a distributed file system with pipeline parallelism," in *Proc. ACM SIGOPS 28th Symp. Operating Syst. Princ.*, 2021, pp. 756–771.
[8] H. N. Schuh, W. Liang, M. Liu, J. Nelson, and A. Krishnamurthy, "Xenic: SmartNIC-accelerated distributed transactions," in *Proc. ACM SIGOPS 28th Symp. Operating Syst. Princ.*, 2021, pp. 740–755.
[9] H. Li et al., "Leapio: Efficient and portable virtual NVMe storage on ARM SoCs," in *Proc. 25th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2020, pp. 591–605.
[10] P. M. Phothilimthana, M. Liu, A. Kaufmann, S. Peter, R. Bodik, and T. Anderson, "Floem: A programming system for NIC-accelerated network applications," in *Proc. 13th USENIX Symp. Operating Syst. Des. Implementation*, 2018, pp. 663–679.

[11] S. Grant, A. Yelam, M. Bland, and A. C. Snoeren, "SmartNIC performance isolation with FairNIC: Programmable networking for the cloud," in *Proc. Annu. Conf. ACM Special Int. Group Data Commun. Appl. Technol. Architectures Protoc. Comput. Commun.*, 2020, pp. 681–693.
[12] Z. Guo et al., "LogNIC: A high-level performance model for SmartNICs," in *Proc. 56th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2023, pp. 916–929.
[13] S. Choi, M. Shahbaz, B. Prabhakar, and M. Rosenblum, "λ-NIC: Interactive serverless compute on programmable SmartNICs," in *Proc. IEEE 40th Int. Conf. Distrib. Comput. Syst.*, 2020, pp. 67–77.
[14] J. Liu, C. Maltzahn, C. Ulmer, and M. L. Curry, "Performance characteristics of the BlueField-2 SmartNIC," 2021, *arXiv:2105.06619*.
[15] F. Xu, F. Liu, H. Jin, and A. V. Vasilakos, "Managing performance overhead of virtual machines in cloud computing: A survey, state of the art, and future directions," in *Proc. IEEE*, vol. 102, no. 1, pp. 11–31, Jan. 2014.
[16] C. Zeng, F. Liu, S. Chen, W. Jiang, and M. Li, "Demystifying the performance interference of co-located virtual network functions," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 765–773.
[17] D. Firestone et al., "Azure accelerated networking: SmartNICs in the public cloud," in *Proc. 15th USENIX Symp. Netw. Syst. Des. Implementation*, 2018, pp. 51–66.
[18] L. Thostrup, D. Failing, T. Ziegler, and C. Binnig, "A DBMS-centric evaluation of bluefield DPUS on fast networks," in *Proc. 13th Int. Workshop Accelerating Analytics Data Manage. Syst. Using Modern Processor Storage Architectures*, 2022, pp. 1–10.
[19] NVIDIA, "NVIDIA bluefield-2 DPU," 2022. [Online]. Available: https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/documents/datasheet-nvidia-bluefield-2-dpu.pdf
[20] Mellanox, "Mellanox innova-2flex," 2020. [Online]. Available: http://www.mellanox.com/related-docs/prod_adapter_cards/PB_Innova-2_Flex.pdf
[21] Intel, "Intel ASIC-based IPU mount evans," 2022. [Online]. Available: https://download.intel.com/newsroom/2022/corporate/vision/Intel-IPU-Roadmap-Fact-Sheet.pdf
[22] F. Xu, F. Liu, and H. Jin, "Heterogeneity and interference-aware virtual machine provisioning for predictable performance in the cloud," *IEEE Trans. Comput.*, vol. 65, no. 8, pp. 2470–2483, Aug. 2016.
[23] J. Guo, F. Liu, T. Wang, and J. C. S. Lui, "Pricing intra-datacenter networks with over-committed bandwidth guarantee," in *Proc. USENIX Annu. Tech. Conf.*, 2017, pp. 69–81.
[24] F. Liu, J. Guo, X. Huang, and J. C. S. Lui, "eBA: Efficient bandwidth guarantee under traffic variability in datacenters," *IEEE/ACM Trans. Netw.*, vol. 25, no. 1, pp. 506–519, Feb. 2017.
[25] N. Kr Sharma, M. Liu, K. Atreya, and A. Krishnamurthy, "Approximating fair queueing on reconfigurable switches," in *Proc. 15th USENIX Symp. Netw. Syst. Des. Implementation*, 2018, pp. 1–16.
[26] J. Kim, K. Jang, K. Lee, S. Ma, J. Shim, and S. Moon, "NBA (network balancing act) a high-performance packet processing framework for heterogeneous processors," in *Proc. 10th Eur. Conf. Comput. Syst.*, 2015, pp. 1–14.
[27] B. Li et al., "ClickNP: Highly flexible and high performance network processing with reconfigurable hardware," in *Proc. ACM SIGCOMM Conf.*, 2016, pp. 1–14.
[28] A. Gulati, A. Merchant, M. Uysal, P. Padala, and P. Varman, "Efficient and adaptive proportional share I/O scheduling," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 37, no. 2, pp. 79–80, 2009.
[29] H. Tan, L. Huang, Z. He, Y. Lu, and X. He, "DMVL: An I/O bandwidth dynamic allocation method for virtual networks," *J. Netw. Comput. Appl.*, vol. 39, pp. 104–116, 2014.
[30] R. Liu, Z. Tan, L. Long, Y. Wu, Y. Tan, and D. Liu, "Improving fairness for SSD devices through DRAM over-provisioning cache management," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 10, pp. 2444–2454, Oct. 2022.
[31] A. Tavakkol et al., "FLIN: Enabling fairness and enhancing performance in modern NVMe solid state drives," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit.*, 2018, pp. 397–410.
[32] J. Guo, F. Liu, J. C. S. Lui, and H. Jin, "Fair network bandwidth allocation in IaaS datacenters via a cooperative game approach," *IEEE/ACM Trans. Netw.*, vol. 24, no. 2, pp. 873–886, Apr. 2016.
[33] J. Lin, K. Patel, B. E. Stephens, A. Sivaraman, and A. Akella, "PANIC: A high-performance programmable NIC for multi-tenant networks," in *Proc. 14th USENIX Symp. Operating Syst. Des. Implementation*, 2020, pp. 243–259.
[34] P. Barham et al., "Xen and the art of virtualization," *ACM SIGOPS Operating Syst. Rev.*, vol. 37, no. 5, pp. 164–177, 2003.
[35] K. Tian, Y. Dong, and D. Cowperthwaite, "A full GPU virtualization solution with mediated pass-through," in *Proc. USENIX Annu. Tech. Conf.*, 2014, pp. 121–132.
[36] NVIDIA, "DOCA," 2023. [Online]. Available: https://developer.nvidia.com/networking/doca
[37] K. Wiles, "The pktgen application–pktgen 3.2.4 documentation," 2020. [Online]. Available: https://pktgen-dpdk.readthedocs.io/en/latest/
[38] H. Fan et al., "NCQ-aware I/O scheduling for conventional solid state drives," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2019, pp. 523–532.
[39] J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, pp. 74–80, 2013.
[40] Z. Guo, Y. Shan, X. Luo, Y. Huang, and Y. Zhang, "Clio: A hardware-software co-designed disaggregated memory system," in *Proc. 27th ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2022, pp. 417–433.
[41] Y. Niu et al., "PostMan: Rapidly mitigating bursty traffic via on-demand offloading of packet processing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 2, pp. 374–387, Feb. 2022.
[42] A. Kaufmann, S. Peter, T. E. Anderson, and A. Krishnamurthy, "FlexNIC: Rethinking network DMA," in *Proc. 15th USENIX Conf. Hot Topics Operating Syst.*, 2015, Art. no. 7.
[43] Y. Qiu et al., "Automated smartnic offloading insights for network functions," in *Proc. ACM SIGOPS 28th Symp. Operating Syst. Princ.*, 2021, pp. 772–787.
[44] R. Shashidhara, T. Stamler, A. Kaufmann, and S. Peter, "FlexTOE: Flexible TCP offload with fine-grained parallelism," in *Proc. 19th USENIX Symp. Netw. Syst. Des. Implementation*, 2022, pp. 87–102.
[45] J. Li, Y. Lu, Q. Wang, J. Lin, Z. Yang, and J. Shu, "AlNiCo:SmartNIC-accelerated contention-aware request scheduling for transaction processing," in *Proc. USENIX Annu. Tech. Conf.*, 2022, pp. 951–966.
[46] H. Ji et al., "STYX: Exploiting SmartNIC capability to reduce datacenter memory tax," in *Proc. USENIX Annu. Tech. Conf.*, 2023, pp. 619–633.
[47] Z. Guo, H. Zhang, C. Zhao, Y. Bai, M. Swift, and M. Liu, "LEED: A low-power, fast persistent key-value store on SmartNIC JBOFs," in *Proc. ACM SIGCOMM Conf.*, 2023, pp. 1012–1027.
[48] M. Liu, S. Peter, A. Krishnamurthy, and P. M. Phothilimthana, "E3: Energy-efficient microservices on SmartNIC-accelerated servers," in *Proc. USENIX Annu. Tech. Conf.*, 2019, pp. 363–378.
[49] D. Du, Q. Liu, X. Jiang, Y. Xia, B. Zang, and H. Chen, "Serverless computing on heterogeneous computers," in *Proc. 27th ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2022, pp. 797–813.
[50] A. Belay et al., "The IX operating system: Combining low latency, high throughput, and efficiency in a protected dataplane," *ACM Trans. Comput. Syst.*, vol. 34, no. 4, pp. 1–39, 2016.
[51] G. Prekas, M. Kogias, and E. Bugnion, "ZygOS: Achieving low tail latency for microsecond-scale networked tasks," in *Proc. 26th Symp. Operating Syst. Princ.*, 2017, pp. 325–341.
[52] K. Kaffes, T. Chong, J. T. Humphries, A. Belay, D. Mazières, and C. Kozyrakis, "Shinjuku: Preemptive scheduling for μ second-scale tail latency," in *Proc. 16th USENIX Symp. Netw. Syst. Des. Implementation*, 2019, pp. 345–360.

Zhaoyang Huang received the BS degree in computer science and technology from the China West Normal University, in 2020. She is currently working toward the PhD degree with the College of Computer Science and Electronic Engineering, Hunan University. Her current research interests include cloud computing and data centers.

Yanjie Tan received the BS and MS degrees from the Huazhong University of Science and Technology, China, in 2011 and 2015, respectively, and the PhD degree from Hunan University, China, in 2021. He is a postdoctor with the College of Computer Science and Electronic Engineering, Hunan University. His current research interests include real-time system and image and video processing.

Yifu Zhu (Graduate Student Member, IEEE) received the BS degree from the College of Electronic Science and Engineering, Jilin University, in 2019, and the MS degree from Hunan University, in 2023. He is currently working toward the PhD degree with the College of Computer Science and Electronic Engineering, Hunan University. His current research interests include FPGA and real-time systems.

Huailiang Tan received the BS degree from Central South University, China, in 1992, the MS degree from Hunan University, China, in 1995, and the PhD degree from Central South University, China, in 2001. He has more than eight years of industrial R&D experience in the field of information technology. He was a visiting scholar with Virginia Commonwealth University from 2010 to 2011. He is currently a full professor of computer science and technology with Hunan University, China. His research interests include high performance I/O, image and video processing, and embedded systems.

Keqin Li (Fellow, IEEE) received the BS degree in computer science from Tsinghua University, in 1985, and the PhD degree in computer science from the University of Houston, in 1990. He is a SUNY distinguished professor with the State University of New York and a National distinguished professor with Hunan University (China). He has authored or co-authored more than 1000 journal articles, book chapters, and refereed conference papers. He received several best paper awards from international conferences including PDPTA-1996, NAECON-1997, IPDPS-2000, ISPA-2016, NPC-2019, ISPA-2019, and CPSCom-2022. He holds nearly 75 patents announced or authorized by the Chinese National Intellectual Property Administration. He is among the world's top five most influential scientists in parallel and distributed computing in terms of single-year and career-long impacts based on a composite indicator of the Scopus citation database. He was a 2017 recipient of the Albert Nelson Marquis Lifetime Achievement Award for being listed in Marquis Who's Who in Science and Engineering, Who's Who in America, Who's Who in the World, and Who's Who in American Education for more than twenty consecutive years. He received the Distinguished Alumnus Award from the Computer Science Department, University of Houston, in 2018. He received the IEEE TCCLD Research Impact Award from the IEEE CS Technical Committee on Cloud Computing, in 2022 and the IEEE TCSVC Research Innovation Award from the IEEE CS Technical Community on Services Computing, in 2023. He won the IEEE Region 1 Technological Innovation Award (Academic), in 2023. He is a member of the SUNY Distinguished Academy. He is an AAAS fellow, an AAIA fellow, and an ACIS founding fellow. He is an academician member and fellow of the International Artificial Intelligence Industry Alliance. He is a member of Academia Europaea (Academician of the Academy of Europe).