RosebudVirt: A High-Performance and Partially Reconfigurable FPGA Virtualization Framework for Multitenant Networks

Yiwei Chang and Zhichuan Guo

***Abstract*— Field-programmable gate arrays (FPGAs) are key accelerators in cloud data centers due to their parallelism and programmability. However, challenges such as low hardware utilization and high virtualization overhead persist. This brief presents RosebudVirt, a high-performance and partially reconfigurable FPGA virtualization framework tailored for multitenant networks. It enhances the original Rosebud by introducing single-root I/O virtualization (SR-IOV) support, partitioning the PCIe-attached FPGA device into multiple physical functions (PFs) and virtual functions (VFs) accessible to the linux kernel via PF and VF drivers. This facilitates direct mapping among tenants, VFs, and reconfigurable packet-processing units (RPUs) within the FPGA. RosebudVirt achieves near-native throughput with <1% area overhead and increases hardware utilization by up to 7.6 times by additional VF drivers and network interfaces. What is more, RosebudVirt is compatible with Kubernetes and Docker.**

***Index Terms*— Cloud data centers, field-programmable gate array (FPGA) virtualization, multitenant networks, partial reconfiguration (PR), single-root I/O virtualization (SR-IOV).**

I. Introduction

The significance of field programmable gate array (FPGA) in cloud data centers is well-established, as demonstrated by numerous studies [1], [2], [3]. Microsoft's Catapult [4], which deployed Intel Stratix V FPGAs across 1632 servers, achieved a remarkable 95% ranking throughput improvement for the Bing web search engine. Amazon also integrated Xilinx VU9P FPGAs in AWS [5], offering a novel solution for enterprises and academic institutions with high-performance computing demands. Despite their widespread acceptance, a series of novel challenges remain [6], [7], including efficiently managing hardware resources, maximizing their utilization, and virtualizing them for cloud computing. These considerations underscore the fundamental question of how to attain high-performance FPGA virtualization, thereby ensuring optimal resource allocation and utilization in the cloud.

FPGA virtualization comprises three main categories: resource, node, and multinode level [2]. Virtualization at resource level primarily focuses on architecture and I/O resources. Overlay-based methods such as FPGAVirt [8], which constructs a compatibility layer for different vendors, often introduce area overhead, thus reducing the amount of hardware resources available for users. In contrast, partial reconfiguration (PR) technique [9] divides FPGA fabric into static regions with fixed modules and dynamic regions with user-defined logic, effectively eliminating the area overhead associated with overlay and enables runtime updates for the dynamic regions. Single-root I/O virtualization (SR-IOV), an extension to the PCI express (PCIe) specification, allows a PCIe device to be virtualized into multiple full-featured physical functions (PFs) and lightweight virtual functions (VFs), offering tenants with near bare-metal performance [10], who share these resources in virtualized environments through containers or others. Virtualization at node level refers to the management of virtualized resources within a single FPGA and the creation of isolated and secure environments for multiple tenants, each with its own dedicated resources, such as Nimblock [11]. Virtualization at multinode level involves the coordination and management of resources across multiple FPGAs. This can be achieved by leveraging cloud-native tools such as Kubernetes and OpenStack, which provide powerful capabilities for resource scheduling.

The ultimate objectives of FPGA virtualization are more comprehensively outlined in [1], as follows.

1) Abstraction hides hardware complexities from application developers and operating systems (OS), providing a uniform interface to virtual FPGA resources.
2) Multitenancy enables multiple applications and tenants to share FPGA resources.
3) Resource management ensures load balancing and fault-tolerant resource allocation.
4) Isolation guarantees strict performance and data separation among tenants for security and system resilience.

Rosebud [12] is an acceleration framework for network functions that partitions FPGA fabrics through PR. The reconfigurable packet-processing units (RPUs) featuring 64-bit RISC-V cores and hardware accelerators in the dynamic regions enable Rosebud to handle ~200 Gbps of traffic with latency ranging from 0.7 to 7 μs. However, Rosebud only virtualizes its architecture at the resource level, neglecting I/O resources. Thus, it can only provide a single *network interface* from the kernel for multiple tenants and scheduled by software, which will cause serious performance loss when implementing virtualization. In this brief, we introduce RosebudVirt, a high-performance and partially reconfigurable virtualization framework that enhances Rosebud's capabilities for virtualization at resource, node, and multinode levels, achieving the above objectives. The main contributions are as follows.

1) *Resource Level:* RosebudVirt enhances Rosebud's capabilities with high-performance SR-IOV by virtualizing the PCIe-attached FPGA device into 1 PF along with 16, 8, or 4 VFs for multitenancy, achieving near-native throughput with an area overhead below 1%.
2) *Node Level:* RosebudVirt offers linux kernel PF, VF drivers to abstract virtualized FPGA, uses a vFPGA dispatcher to isolate multitenant traffic, and establishes a direct mapping between tenants, VFs, and RPUs, increasing hardware utilization by up to 7.6 times.
3) *Multinode Level:* RosebudVirt is compatible with Kubernetes and Docker, enabling resource management and orchestration across multiple FPGAs in a cluster.

Manuscript received 5 May 2024; revised 1 July 2024; accepted 25 July 2024. Date of publication 7 August 2024; date of current version 31 December 2024. This work was supported by the Strategic Priority Research Program of Chinese Academy of Sciences: Research on Information Collaborative Service Data Sharing Technology under Grant XDA031050100. *(Corresponding author: Zhichuan Guo.)*

The authors are with the National Network New Media Engineering Research Center, Institute of Acoustics, Chinese Academy of Sciences, Haidian, Beijing 100190, China, and also with the School of Electronic, Electrical and Communication Engineering, University of Chinese Academy of Sciences, Shijingshan, Beijing 100049, China (e-mail: changyw@dsp.ac.cn; guozc@dsp.ac.cn).

Digital Object Identifier 10.1109/TVLSI.2024.3436017

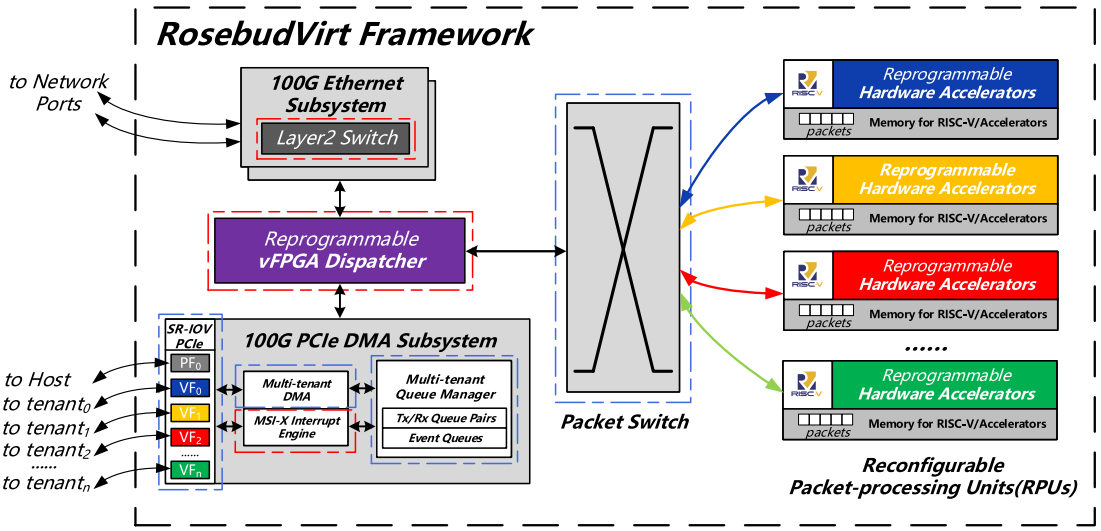

Fig. 1. Hardware overview of the RosebudVirt framework.

The subsequent sections are organized as follows: Section II provides discussion on related works, Section III explains the overall architecture, Section IV presents implementation and evaluation results, and Section V draws the conclusion.

II. Related Works and Discussion

Nowadays, to virtualize FPGA, there are numerous solutions offered by both industrial and academic institutions. At the resource level, FPGAVirt [8] streamlines I/O access for multiple VMs using virtio and constructs an FPGA overlay. However, paravirtualization overhead limits performance compared with SR-IOV, and the overlay consumes extra hardware resources. Coyote [13] provides configurable FPGA abstractions to OS through PR in cloud environments but lacks support for SR-IOV. At the node level, SVFF [14] provides a VF management framework, whose contribution mainly focuses on software without a detailed hardware implementation. Nimblock [11] uses PR and its novel scheduling algorithm to enable fine-grained FPGA sharing but with unavoidable software overhead. SuperNIC [15] offers a multitenancy SmartNIC by mapping network tasks into physical chains through PR but without efficiency I/O virtualization. At the multinode level, IBM cloudFPGA [16] relies on Openstack and VM-based solutions for FPGA virtualization, which are notorious for their high CPU usage and prolonged response times. FFIVE [17] implements FPGA virtualization with lightweight Docker managed by Kubernetes. However, it also lacks PR capability, hindering its adaptability to dynamic network environments. Despite using PR, the AWS EC2 F1 [5] can only be occupied by a single tenant at any given time, which goes against the objective of multitenancy in FPGA virtualization.

III. Overall Architecture of Proposed Design

In this section, we will elaborate on the hardware and software implementations of RosebudVirt.

A. Hardware Implementation

The hardware implementation of RosebudVirt, shown in Fig. 1, consists of fixed static regions and reprogrammable dynamic regions, using only two basic Xilinx hard IP cores: PCIe and CMAC. Components within the blue dashed boxes are enhanced Rosebud modules, while those in the red dashed boxes are new additions built from scratch, all specifically designed to enhance Rosebud's capabilities for virtualization.

1) SR-IOV PCIe: RosebudVirt enables SR-IOV capability in PCIe IP to partition FPGA into 1 PF along with 16, 8, or 4 VFs [18] and partitions its fabric into 16, 8, or 4 slots, each hosting an RPU mapped to a VF, which is then provided to tenants to meet varying demands. The 4 RPUs case accommodates applications with complex functionality, while the 16 RPUs case optimizes hardware utilization and cost-effectiveness. The PF is dedicated to control-plane tasks such as serving as SDN agent to interact with controllers. SR-IOV PCIe distinguishes multitenant traffic using function details in transaction layer packets (TLPs) descriptor fields.

2) Multitenant DMA: The DMA manages host–FPGA interaction by handling descriptor and data transfer tasks. Descriptors, which contain essential metadata related to specific data, have been enhanced for virtualization to include function details encapsulated within DMA read and write requests. These details are eventually integrated into TLPs, which are then routed to different tenants.

3) Multitenant QM: The queue manager (QM) polls on parameterizable Tx/Rx queue pairs (QPs) and event queues (EQs), using producer and consumer pointers. Packets queue up in the Tx/Rx QPs, while entries in the EQs indicate which QPs' packets' transmission has been completed. In RosebudVirt, PF and VF drivers assign separate QPs and EQs to tenants, allowing multiple tenants to share a public DMA and QM while ensuring isolation of traffic among multiple tenants.

4) MSI-X Interrupt Engine: MSI-X provides greater flexibility with 2048 discrete interrupts in virtualized environments compared with MSI and legacy interrupts, and when enabling SR-IOV capability in PCIe IP, only MSI-X interrupts are supported. Thus, we add an MSI-X interrupt engine to RosebudVirt, embedding an MSI-X table shown in Fig. 2, which is lacking in Rosebud. During transmission, the table is referenced to trigger interrupts for different tenants based on the *interrupt index* initialized by drivers from EQs, ensuring isolation of interrupts among multiple tenants.

5) vFPGA Dispatcher: The vFPGA dispatcher, replacing Rosebud's load balancer, efficiently distributes multitenant packets in a pipelined manner without additional bubbles. A slot keeper within it maintains the count and index of free slots for all the RPUs as descriptors, indicating the RPUs' load and pointing to the specified RPU memory. When a packet is received from PCIe or CMAC, the vFPGA dispatcher retrieves a descriptor from the slot keeper based on the packet's function details. If successful, the packet is sent to

TABLE I
COMPARISON OF RESOURCE UTILIZATION BETWEEN ROSEBUDVIRT AND ROSEBUD

Framework	SR-IOV	PFs	VFs	Interrupts	Tx/Rx QPs	EQs	Queue Size	LUTs	Registers	BRAM	URAM
RosebudVirt-16 RPUs	✓	1	16	2MSI-X*16	16*16/1024	4*16/256	1024	257.4k(22.35%)	330.0k(14.32%)	552(25.56%)	626(65.21%)
Rosebud-16 RPUs	✗	1	0	32MSI	256/1024	64/256	1024	256.7k(22.28%)	324.9k(14.10%)	550(25.46%)	626(65.21%)
RosebudVirt-8 RPUs	✓	1	8	4MSI-X*8	32*8/1024	8*8/256	1024	163.6k(14.21%)	224.1k(9.73%)	348(16.11%)	338(35.21%)
Rosebud-8 RPUs	✗	1	0	32MSI	256/1024	64/256	1024	163.3k(14.19%)	219.4k(9.53%)	346(16.02%)	338(35.21%)
RosebudVirt-4 RPUs	✓	1	4	8MSI-X*4	64*4/1024	16*4/256	1024	137.1k(11.88%)	228.3k(9.89)	362(16.76%)	244(25.42%)
Rosebud-4 RPUs	✗	1	0	32MSI	256/1024	64/256	1024	136.1k(11.81%)	223.6k(9.69%)	358(16.57%)	244(25.42%)

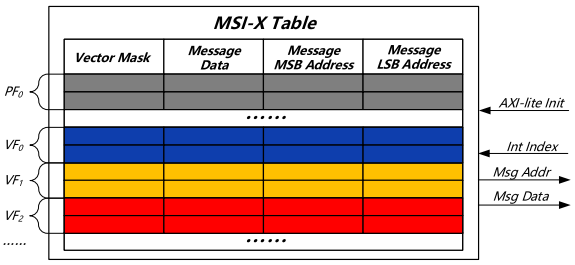

Fig. 2. MSI-X table structure.

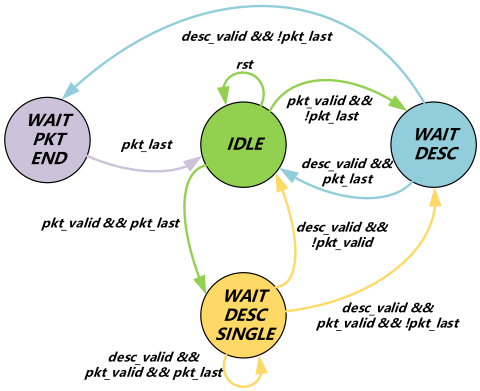

Fig. 3. FSM of the vFPGA dispatcher.

the corresponding RPU memory via the packet switch; otherwise, the pipeline is blocked until the RPU is free, at which point packet loss may occur. When the packet leaves, the slot keeper reclaims the descriptor and updates the count of free slots. Its finite state machine (FSM) is shown in Fig. 3. Compared with Rosebud's load balancer, the vFPGA dispatcher adds only one clock cycle of delay with no throughput loss, providing high performance and isolation in virtualized environments.

6) Packet Switch/Ethernet Subsystem/RPUs: The packet switch connects the vFPGA dispatcher to the RPUs, and we have enhanced it to support 4 RPUs. The Ethernet subsystem enables network access and embeds a Layer2 switch to allocate function details for incoming packets based on their destination MAC addresses. RPUs offer hardware and software co-design capabilities for packet processing.

7) Work Flow: The diagram in Fig. 4 illustrates the receiving and sending of multitenant packets within RosebudVirt. For packet receiving, incoming packets from the vFPGA dispatcher are directed to the corresponding Rx QPs based on their function details. The producer pointers are then incremented, triggering DMA to fetch descriptors and write packets into the tenant's memory. Upon completion, the producer pointers of the tenant's EQs are incremented to signal QPs completion, and an interrupt notifies the tenant to

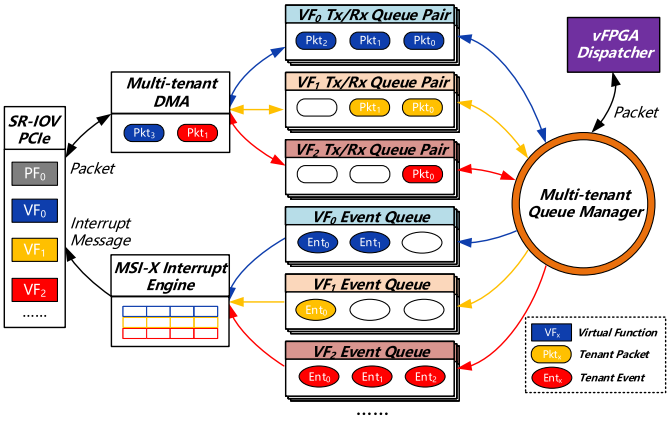

Fig. 4. Receiving and sending of multitenant packets.

retrieve packets from memory and update the consumer pointers of QPs and EQs accordingly. For packet sending, the tenant increments the producer pointers of its own Tx QPs to signal packets for sending. The DMA pulls down descriptors and fetches packets from the tenant's memory. After transmission, the producer pointers of the tenant's EQs are incremented to mark QPs completion, and an interrupt prompts the tenant to free memory and update the consumer pointers.

B. Software Implementation

1) Driver: The PF, VF drivers create *network interfaces* and *character devices* aligned with RPUs, providing uniform interfaces for the OS. They manage Tx/Rx QPs, EQs, interrupt allocation, handler registration, device memory mapping, and MAC address assignment.

2) Kubernetes and Docker: We use an open-source kubernetes SR-IOV network device plugin [19] to integrate these VFs, which map RPUs and *network interfaces*, into cluster resource pool and orchestrate Docker containers to request resources according to tenants' demands, thereby enhancing RosebudVirt's capability in virtualization at multinode level.

IV. IMPLEMENTATION AND EVALUATION RESULTS

We have successfully implemented RosebudVirt on the Xilinx AU200 platform deployed within a Dell R740xd server featuring dual Xeon[1] Silver 4216 CPUs and connected it to an IXIA tester using two 100G optical fibers to verify SR-IOV and conduct performance testing.

A. Implementation

To ensure impartial comparisons across all the cases, we allocate Tx/Rx QPs, EQs, and MSI-X or MSI interrupts proportionately from

[1]Registered trademark.

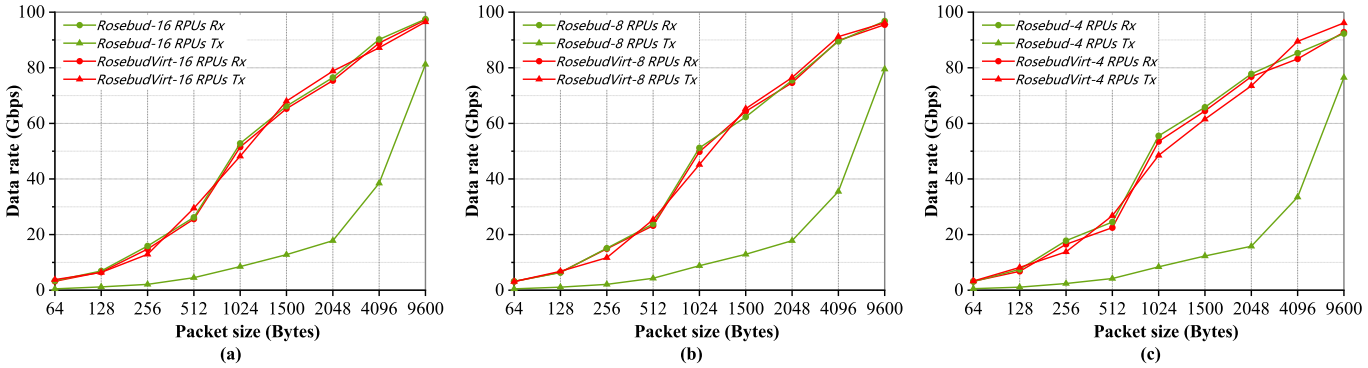

Fig. 5. Comparison of the throughput between RosebudVirt and Rosebud. (a) 16 RPUs. (b) 8 RPUs. (c) 4 RPUs.

```
[root@k8s-work ~]# lspci | grep 1234
42:00.0 Ethernet controller: Device 1234:1001
42:00.4 Ethernet controller: Device 1234:0000
42:00.5 Ethernet controller: Device 1234:0000
42:00.6 Ethernet controller: Device 1234:0000
42:00.7 Ethernet controller: Device 1234:0000
[root@k8s-work ~]# lspci -s 42:00.0 -v
42:00.0 Ethernet controller: Device 1234:1001
        Subsystem: Device 1234:1001
        Flags: bus master, fast devsel, latency 0, NUMA node 1, IOMMU group 53
        Memory at d0000000 (64-bit, prefetchable) [size=16M]
        Capabilities: [40] Power Management version 3
        Capabilities: [60] MSI-X: Enable+ Count=8 Masked-
        Capabilities: [70] Express Endpoint, IntMsgNum 0
        Capabilities: [100] Advanced Error Reporting
        Capabilities: [140] Single Root I/O Virtualization (SR-IOV)
        Capabilities: [180] Alternative Routing-ID Interpretation (ARI)
        Capabilities: [1c0] Secondary PCI Express
        Capabilities: [350] Vendor Specific Information: ID=0001 Rev=1 Len=02c <?>
        Kernel driver in use: mqnic
```

Listing. 1: Logs from linux for 4 RPUs.

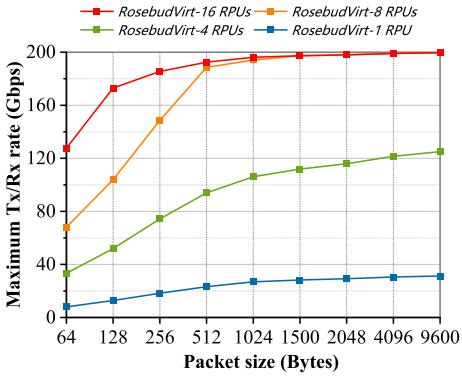

Fig. 6. RPU performance of RosebudVirt.

a total pool of 1024 Tx/RX QPs, 256 EQs, 32+ MSI-X, or 32 MSI interrupts. Specifically, RosebudVirt's allocations are structured as follows: 16/4/2 for 16 RPUs, 32/8/4 for 8 RPUs, and 64/16/8 for 4 RPUs. For comparisons, Rosebud's allocations consist of 256 Tx/Rx QPs, 64 EQs, and 32 MSI interrupts across 16, 8, and 4 RPUs, respectively, from the same total pool. The resource utilization for RosebudVirt and Rosebud is shown in Table I, and the resources allocated for each RPU in LUTs, Registers, BRAM, and URAM among cases of 16, 8, and 4 RPUs are approximately 26k/52k/48/32, 55k/111k/120/64, and 140k/280k/270/144. The results reveal that RosebudVirt achieves efficient virtualization with an area overhead below 1%, highlighting its resource utilization efficiency and scalability for multitenancy. What is more, RosebudVirt can be ported to other platforms easily.

B. Evaluation

1) SR-IOV Verification: After enabling SR-IOV and IOMMU in the BIOS, and configuring the bitstreams, we loaded the PF and VF drivers and used the *lspci* command to print relevant information. The logs from Linux for 4 RPUs are depicted in Listing 1, where the host identifies one PF and 4 VFs that have been integrated into cluster resource pool. Containers orchestrated by Kubernetes request resources and bind VFs, RPUs, and *network interfaces* for transmitting packets, accessing registers, dynamically updating hardware logic or RISC-V binaries, and monitoring hardware loads.

2) SR-IOV Performance Testing: We compared the Rx/Tx performance between RosebudVirt and Rosebud, aiming to demonstrate the efficiency of RosebudVirt's virtualization. RPUs were configured to perform host-CMAC forwarding. For Rx testing, we used IXIA to conduct stress tests with a specified number of packets and compared the kernel's *network interface* statistics with IXIA to determine packet loss. Packets were aggregated from multiple containers within RosebudVirt. The results depicted in Fig. 5 reveal that 16, 8, and 4 RPUs achieve throughputs ranging from 3.1 to 97.5 Gbps. The performance bottleneck occurs in the linux kernel driver and PCIe DMA subsystem inherited from Rosebud, rather than in the RPUs, because their bandwidth far exceeds this bottleneck. PCIe factors such as max payload size (MPS) and max read request size (MRRS) on our server also matter. RosebudVirt's total throughput closely matches Rosebud's in different RPU cases, indicating minimal virtualization overhead. For Tx testing, we used PF_RING [20] to send packets from the kernel's *network interfaces* and measured extreme Rx rates on IXIA. Packets were sent from multiple containers within RosebudVirt. In Fig. 5, RosebudVirt demonstrates superior throughput in total compared with Rosebud due to latter's inadequate hardware utilization caused by the inefficient data copying between *user-space* and *kernel-space*. By loading additional VF drivers and creating more *network interfaces*, RosebudVirt achieves up to 7.6 times higher Tx Rate due to higher hardware utilization. Thus, we consider it a 7.6 times increase in hardware utilization, while the bottleneck in the linux kernel driver and PCIe DMA subsystem remains.

3) RPU Performance Testing: We sent 200 Gbps multitenant traffic with different destination MAC addresses from IXIA to RosebudVirt to test the maximum performance of RPUs, which were dynamically updated to perform CMAC-CMAC forwarding. Theoretically, a single RPU can provide 32 Gbps bandwidth due to its 128-bit bus at 250 MHz. Thus, 4, 8, and 16 RPUs provide 128, 256, and 512 Gbps, respectively. The results in Fig. 6 show that performance improves with more RPUs. A single RPU reaches up to 31.3 Gbps, while 4 RPUs achieve 125.0 Gbps. Both 8 and 16 RPUs achieve full 200 Gbps line rate except for small packets, as their bandwidth exceeds 200 Gbps. The results also demonstrate the high performance and isolation of the vFPGA dispatcher with no

throughput loss compared with Rosebud and confirm that RPUs are not the bottleneck in SR-IOV performance testing.

V. Conclusion

RosebudVirt provides a high-performance and partially reconfigurable FPGA virtualization framework for multitenant networks, which enhances the FPGA virtualization capabilities at resource, node, and multinode level lacking in Rosebud. It achieves near-native throughput using SR-IOV, increasing hardware utilization by up to 7.6 times with an area overhead below 1% compared with the original design. What is more, it is also compatible with Kubernetes and Docker. RosebudVirt not only promotes high-performance computing in cloud data centers but also facilitates innovative network experimentation with its advanced FPGA virtualization capabilities.

References

[1] M. H. Quraishi, E. B. Tavakoli, and F. Ren, "A survey of system architectures and techniques for FPGA virtualization," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 9, pp. 2216–2230, Sep. 2021.

[2] A. Vaishnav, K. D. Pham, and D. Koch, "A survey on FPGA virtualization," in *Proc. 28th Int. Conf. Field Program. Log. Appl. (FPL)*, Aug. 2018, pp. 131–1317.

[3] M. Sha, Z. Guo, and M. Song, "A review of FPGA's application in high-speed network processing," *Netw. New Media*, vol. 10, pp. 1–11, Nov. 2021.

[4] A. Putnam et al., "A reconfigurable fabric for accelerating large-scale datacenter services," *IEEE Micro*, vol. 35, no. 3, pp. 10–22, May 2015.

[5] Amazon. (2024). *Amazon EC2 F1 Instances*. [Online]. Available: https://aws.amazon.com/ec2/instance-types/f1

[6] C. Bobda et al., "The future of FPGA acceleration in datacenters and the cloud," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 15, no. 3, pp. 1–42, Feb. 2022, doi: 10.1145/3506713.

[7] S. Yazdanshenas and V. Betz, "The costs of confidentiality in virtualized FPGAs," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 10, pp. 2272–2283, Oct. 2019.

[8] J. Mbongue, F. Hategekimana, D. T. Kwadjo, D. Andrews, and C. Bobda, "FPGAVirt: A novel virtualization framework for FPGAs in the cloud," in *Proc. IEEE 11th Int. Conf. Cloud Comput. (CLOUD)*, Jul. 2018, pp. 862–865.

[9] Xilinx. (2024). *Dynamic Function Exchange*. [Online]. Available: https://docs.amd.com/r/en-US/ug909-vivado-partial-reconfiguration

[10] J. Li, S. Xue, W. Zhang, R. Ma, Z. Qi, and H. Guan, "When I/O interrupt becomes system bottleneck: Efficiency and scalability enhancement for SR-IOV network virtualization," *IEEE Trans. Cloud Comput.*, vol. 7, no. 4, pp. 1183–1196, Oct. 2019.

[11] M. Mandava, P. Reckamp, and D. Chen, "Nimblock: Scheduling for fine-grained FPGA sharing through virtualization," in *Proc. 50th Annu. Int. Symp. Comput. Archit.* New York, NY, USA: Association for Computing Machinery, Jun. 2023, pp. 1–3, doi: 10.1145/3579371.3589095.

[12] M. Khazraee, A. Forencich, G. C. Papen, A. C. Snoeren, and A. Schulman, "Rosebud: Making FPGA-accelerated middlebox development more pleasant," in *Proc. 28th ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.*, vol. 3, Mar. 2023, pp. 586–605.

[13] D. Korolija, T. Roscoe, and G. Alonso, "Do OS abstractions make sense on FPGAs?" in *Proc. 14th USENIX Symp. Operating Syst. Design Implement. (OSDI 20)*. Berkeley, CA, USA: USENIX Association, Nov. 2020, pp. 991–1010. [Online]. Available: https://www.usenix.org/conference/osdi20/presentation/roscoe

[14] S. Cirici, M. Paolino, and D. Raho, "SVFF: An automated framework for SR-IOV virtual function management in FPGA accelerated virtualized environments," in *Proc. Int. Conf. Comput., Inf. Telecommun. Syst. (CITS)*, Jul. 2023, pp. 1–6.

[15] W. Lin, Y. Shan, R. Kosta, A. Krishnamurthy, and Y. Zhang, "SuperNIC: An FPGA-based, cloud-oriented SmartNIC," in *Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays*. New York, NY, USA: Association for Computing Machinery, Apr. 2024, pp. 130–141, doi: 10.1145/3626202.3637564.

[16] B. Ringlein, F. Abel, A. Ditter, B. Weiss, C. Hagleitner, and D. Fey, "System architecture for network-attached FPGAs in the cloud using partial reconfiguration," in *Proc. 29th Int. Conf. Field Program. Log. Appl. (FPL)*, Sep. 2019, pp. 293–300.

[17] M. Ewais, J. C. Vega, A. Leon-Garcia, and P. Chow, "A framework integrating FPGAs in VNF networks," in *Proc. 12th Int. Conf. Netw. Future (NoF)*, Oct. 2021, pp. 1–9.

[18] Xilinx. (2024). *Ultrascale+ Devices Integrated Block for PCI Express V1.3*. [Online]. Available: https://docs.xilinx.com/r/en-US/pg213-pcie4-ultrascale-plus

[19] k8snetworkplumbingwg. (2024). *SR-IOV Network Device Plugin for Kubernetes*. [Online]. Available: https://github.com/k8snetworkplumbingwg/sriov-network-device-plugin

[20] ntop. *PF_Ring, a High Speed Packet Capture Library*. Accessed: Feb. 1, 2024. [Online]. Available: https://www.ntop.org/guides/pf_ring