DockRDMA: Hybrid RDMA Virtualization for Containerized Clouds

Xiaoyu Li[1], Ran Shu[2], Zhongjie Chen[1], Xiaohui Luo[1], Yiran Zhang[1], Bo Wang[1], Qingkai Meng[1], Fengyuan Ren[1]

[1]*Tsinghua University*, [2]*Microsoft Research*
Beijing, China

***Abstract*—Containers have become the *de facto* choice for major cloud services. Meanwhile, with demands for extremely high performance, data centers have widely adopted RDMA for their online services. RDMA virtualization is the critical technology that enables RDMA for containers. Hybrid RDMA virtualization leverages the software flexibility in the control path, and keeps the native performance in the data path. Thus, it is the best choice for RDMA virtualization. State-of-the-art hybrid RDMA virtualization cannot address container-specific problems. This paper proposes DockRDMA, the first hybrid RDMA virtualization solution for containerized clouds. DockRDMA develops several mechanisms, including embedding physical addresses in virtual ones to provide efficient address translation, hybrid network policy enforcement at scale, a general virtual RDMA NIC initialization method to be compatible with all container platforms, and namespace checking to protect the RDMA NIC instances. Evaluation results show that DockRDMA provides bare-metal RDMA performance in the data path, and almost native communication setup time in the control path. Compared with the state-of-the-art hybrid virtualization technology, DockRDMA reduces Hadoop job completion time by 6%. It offers seamless integration with existing container platforms, protects critical information of RDMA NIC instances, and exhibits excellent scalability to meet diverse network policies required by different containers.**

I. Introduction

Containers have become popular in modern data centers thanks to their lightweight virtualization. Nowadays, they have become the *de facto* choice of major cloud services, such as key-value stores [1], [2]. Cloud vendors (e.g., Amazon EC2 [3], Microsoft Azure [4], Google Cloud [5], etc.) have offered container services. Meanwhile, Remote Direct Memory Access (RDMA) achieves extremely high throughput, ultra-low latency, and high CPU efficiency compared to the traditional software transport, TCP. Modern data centers have widely adopted RDMA to boost the performance of cloud computing applications, including machine learning [6], distributed storage [7], [8], databases [9], [10], etc.

The key technology to support RDMA for containers is virtualization. RDMA virtualization technologies (despite whether it is specialized for containers or not) can be classified into software-based virtualization, hardware-based virtualization, and hybrid virtualization. The software-based virtualization [11], [12] realizes all the virtualization logic in the host software. Although benefiting from software flexibility, it incurs non-trivial overheads in the data path, which significantly counteracts the RDMA's performance benefits [13], [14]. The hardware-based virtualization implements multiple logical RDMA NIC (RNIC) instances in the hardware so that VMs or containers can use a virtual RNIC instance exclusively. Despite keeping bare-metal performance, the hardware-based virtualization is restricted by hardware capability, and is inflexible to support complex network policies in cloud environments [12], [14]. Hybrid virtualization extends the philosophy of separating the control path and data path in RDMA, only virtualizes the control path, and leverages the fully offloaded data path capabilities of the hardware [13], [14]. As hybrid virtualization combines software virtualization flexibility and bare-metal RDMA performance, it becomes the most attractive solution to virtualize RDMA.

While hybrid RDMA virtualization was well-studied in virtual machines [13], [14], none of the prior research investigates hybrid RDMA virtualization in containerized clouds to the best of our knowledge. Adopting hybrid RDMA virtualization in containers has technical issues to address. *1) Address Translation Efficiency*. The control path performance is critical in container scenarios [15], [16], [17]. However, the virtualization layer needs to translate the virtual network addresses of both the local and remote side to the physical ones, which incurs overheads in the control and data path. It is challenging to achieve high efficiency of address translation. *2) Network Policies at Scale*. Containers may be deployed at an extremely high density [18]. It is challenging to keep the data path fully offloaded while supporting a large number of containers, especially when the containers have different network policy requirements. *3) Generality*. Containers directly rely on the host drivers to virtualize the NIC instances. The virtualization layer needs to interact with container platforms. So far, there have been various container platforms, each with various publicly released versions. It is unrealistic to extend each version to support RDMA containerization. Thus, offering a general solution is important. *4) Security*. Containers provide weaker isolation than VMs. Simply realizing hybrid RDMA virtualization may cause security issues. First, a container may see the information of physical RNICs and other containers' virtual RNICs as they are managed by host OS. Second, the hybrid virtualization itself can expose extra vulnerabilities.

This paper proposes DockRDMA, a hybrid RDMA virtualization solution for containerized clouds. DockRDMA develops several mechanisms to tackle the above issues. *1)* For high efficiency of address translation, DockRDMA embeds the physical addresses into the virtual ones to allow any host to translate a remote container's address efficiently on the local side. *2)* DockRDMA leverages the offloaded Open vSwitch (OVS) [19] to enforce policies on the policy instances. To support network policies at a large scale, DockRDMA leverages the massive number of IPs supported by RNIC. Each IP works as an individual policy instance. The IPs of the policy instances are offloaded to OVS. DockRDMA assigns the IPs to vRNICs to control the vRDMAs' traffic. *3)* To seamlessly support different container platforms, DockRDMA initializes a virtual RNIC instance inside the callback invoked upon initialization of each virtual Ethernet NIC instance. This callback only resides in the kernel and is independent of any container platforms. *4)* Regarding security, to protect the information of RNIC instances, DockRDMA leverages namespace to let a container only aware of its own vRNIC. Besides, DockRDMA encrypts the virtual addresses. The encryption keys are assigned to each tenant. The key space is very large, making malicious containers difficult to attack a service.

In summary, DockRDMA's main contributions include:

- Proposing the first hybrid RDMA virtualization solution for containers to the best of our knowledge, and tackling the critical issues with several mechanisms, including a new method of address virtualization and translation, hybrid policy enforcement, automatic virtual RNIC initialization, and a mechanism to address container-specific security issue.
- Implementing DockRDMA on Mellanox OFED driver and library. Evaluation results demonstrate that DockRDMA realizes hybrid virtualization, and achieves near-native communication setup time in the control path. It works well with existing container platforms and supports network policies with high deployment density. Compared to the state-of-the-art hybrid virtualization technology, DockRDMA reduces Hadoop job completion time by 6%.

II. Backgound and Motivation

A. RDMA

As data center network speed grows faster than the computation power of a server, the traditional network stack degrades application performance. RDMA achieves high throughput, ultra-low latency, and CPU efficiency through kernel-bypassing, zero-copy interfaces, and remote memory abstractions. Modern data centers have widely adopted RDMA to accelerate their services (including distributed storage systems [7], [8], transactional systems [20], [10], [21], graph processing systems [22], [23], [24], etc.).

RDMA offloads data transport into the hardware, which is the key reason why RDMA achieves such a high performance. RNICs abstract communications as Queue Pairs (QPs), similar to the concept of socket in TCP. However, different from TCP, the metadata of QP is managed by the RNIC. When sending messages, applications call RDMA APIs to post work requests directly to the RNIC. RNICs packetize messages with the information stored in the QP metadata. To establish the data path, applications need to initialize the QP metadata and set up the QPs before communication, which involves the RDMA driver located in the software. Therefore, RDMA uses an architecture where the control and data path are separate.

Each QP falls into one of the two types, i.e., connection-oriented QPs and datagram QPs. For connection-oriented QPs, applications need to set up the QP with the source and destination address plus the destination QPN (QP Number). The RNIC then establishes connections with the other side. For datagram QPs, applications only use the source address to initialize and set up the QP. They need to specify the destination address and QPN each time they communicate over a datagram QP. In both cases, both sides need to exchange their network addresses and QPNs before communication, and this exchange is typically out-of-band (e.g., through Ethernet networks) [25].

B. RDMA Virtualization

Virtualization is one of the key technologies to realize resource sharing and reduce operational costs in cloud. In the past few years, RDMA virtualization has attracted much attention [13], [14], [12], [11], [26]. Existing RDMA virtualization techniques can be classified into three types:

Software-based virtualization realizes all the virtualization logic in the software (Figure 1(a)). The virtualization layer is inserted in both the control and data path. VMware's paravirtual RDMA [11] introduces a frontend driver in the guests and a backend driver in the host to allow the I/O requests to cross the guest-host boundary. FreeFlow [12] introduces a virtualization layer in the host's userspace, which provides virtualized RDMA networks for containers and supports different kinds of control/data path policies. Its main idea is similar to Open vSwitch [27]. Although providing flexible virtualization features, software-based virtualization incurs unignorable overheads in the data path [13], [14].

Hardware-based virtualization adds virtualization capabilities to RNICs (Figure 1(b)). It relies on SR-IOV [28] to define the abstraction of multiple logical devices called virtual functions (VFs) over a single physical device. As such, VMs can use logical RNIC instances exclusively. In addition to I/O virtualization, some network virtualization features are also supported by RNIC hardware. Mellanox extends MACVLAN support for RDMA [26]. It leverages the RNIC's capabilities to expose containers' addresses to the physical network. The hardware-based virtualization provides bare-metal performance but its capabilities are restricted by hardware support, especially when containers/VMs have network policy requirements [14].

Hybrid virtualization extends the philosophy of separating the control and data path in RDMA (Figure 1(c)). It uses software to virtualize the control path, but keeps the data transport hardware offloaded. It combines the flexibility of the software and the bare-metal performance achieved by data

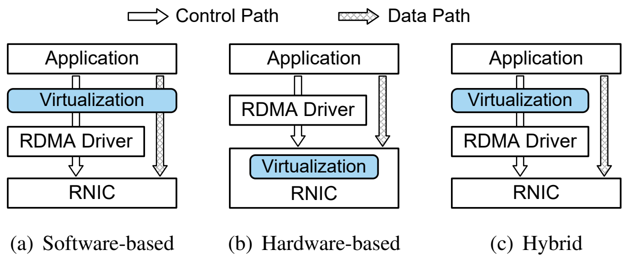


Figure 1. RDMA Virtualization Techniques

Table I
UNIQUENESS OF HYBRID RDMA VIRTUALIZATION FOR CONTAINERS

Containers vs. VMs	Challenges of This Work
Control path performance is critical to containers.	Minimization of the address translation overhead
Container's deployment density is much higher.	Network policies for large scale
Containers share the underlying OS. Thus, they rely on the host drivers to virtualize NIC instances.	A general solution for all container platforms
	Extra container-specific security issues

transport offloading. Hybrid virtualization was first proposed by HyV [13], which provides RDMA virtualization for VMs and only focuses on I/O virtualization. MasQ [14] extends the hybrid virtualization. It uses the control path to realize overlay networks and network security, and leverages hardware capabilities to enforce data path policies (e.g., QoS). Thus, it can achieve almost all the features of FreeFlow while keeping the bare-metal performance.

C. Hybrid RDMA Virtualization for Containerized Clouds

Thanks to lightweight virtualization, containers have become the *de facto* choices for major cloud services. The recent trends of serverless computing and cloud-native have further driven the need for containers [29], [9], [30], [31]. With increasing demands for both extremely high performance and resource management agility, enterprises have now been focusing on RDMA containerization [9], [29], [32], [33]. As analyzed in the previous section, Hybrid RDMA virtualization is the most attractive solution to virtualize RDMA. This work is motivated to investigate how to realize hybrid RDMA virtualization for containers.

However, containers are quite different from VMs (Table I). There are still issues when porting a state-of-the-art hybrid virtualization solution from VMs to containers. We need to tackle four unique challenges, stated as follows:

Minimizing the address translation overheads. The trends of serverless computing and cloud-native have driven the need for elasticity of service provisioning, where the control path performance is critical [15], [16], [17]. However, hybrid RDMA virtualization incurs overheads of network address translation in both the control and data path. In the RDMA connection setup, RNICs are in charge of connecting to the other side. As virtualized in software, any virtual address is unknown to RNICs. Thus, the virtualization layer needs to translate the peer's virtual address to the physical address. An existing solution translates the peer's virtual addresses by asking the cloud manager that manages the global virtual RDMA addresses [14], which incurs quite significant overheads. A possible solution is to store the mapping from virtual addresses to physical ones for all other hosts, which becomes a memory-intensive task. Besides, the address translation may also incur significant overheads in the data path. The reason is that when communicating over datagram QPs, applications need to specify the peer address each time they send messages.

Supporting network policies for large scale. Containers may be deployed at an extremely high density [18], and typically have different network policy requirements. To realize hybrid virtualization, the RNIC becomes the vantage point to enforce network policies. As the virtualization is done in software, the RNIC is only aware of the physical information. It is challenging to enforce different network policies for containers merely with the hardware capabilities, especially for the high deployment density.

Offering a general solution for all container platforms. Different from VMs, where hypervisors have offered the abstraction of para-virtual drivers [34], containers directly rely on the host drivers to virtualize the NIC instances. As a result, hybrid virtualization for containers requires adding into the host RDMA driver a virtualization layer that needs to interact with container platforms. There are different container platforms, each with various publicly released versions. It is unrealistic to extend each version to support RDMA containerization. Thus, it is critical to provide a solution without modifying any version of container platforms. Nevertheless, it is difficult to find a possible point to initialize a virtual RNIC and expose it to a container without the container platforms' awareness. Offering a general solution is challenging.

Addressing container-specific security issues. Containers share the underlying OS. Thus, extending the existing driver to support hybrid RDMA virtualization may expose extra security vulnerabilities. On the one hand, the RDMA driver stores the information of RNIC instances in a filesystem that mainly stores the system configuration. This filesystem is accessible by all the containers. As a result, without any extra isolation, it will be trivial to sniff the information of physical RNICs and other container' virtual RNICs. On the other hand, since the extended RDMA driver handles the network address translation, an attacker container may forge a virtual network address to connect to a victim RDMA-based service and directly access its memory. It is significant to provide a hybrid RDMA containerization solution with security guarantees.

III. DOCKRDMA DESIGN

A. Overview

To realize hybrid RDMA virtualization in containers, we develop DockRDMA to tackle the above fundamental issues through the following key design ideas:

- *Embedding remote physical address into virtual ones to achieve efficient local translation*: DockRDMA embeds physical addresses and QPNs into virtual addresses, and exposes the virtual ones to containers. Any host can translate a remote container's virtual address or QPN without extra communication or local storage overhead.
- *Hybrid policy enforcement at scale*: Modern RNICs can support a much larger number of IP addresses than the number of VFs. RNICs use the IP addresses to identify policy instances. DockRDMA maps different virtual RNIC instances to the IP addresses, and leverages the hardware offloaded OVS to enforce network policies for the logical policy instances.
- *Setting up virtualized RDMA using host OS mechanism to realize generality*: The initialization of a container triggers the network stack to create a virtual Ethernet interface. This step is independent of any specific container platforms. DockRDMA leverages the callback functions provided by the network stack to trigger the setup of vRDMA (virtual RDMA interface, which represents a virtual RNIC instance). The container platforms do not require any modification.
- *Design for container-specific security*: DockRDMA records the information of the namespace in the metadata of each vRDMA, and performs namespace checking when a new vRDMA is created. The kernel only mounts the vRDMA and its information files to the container with the same namespace as the vRDMA. DockRDMA uses lightweight encryption to avoid leaking the physical network information, and exposes the encrypted virtual addresses to containers. As such, the critical RDMA information is well protected.

Figure 2 illustrates our design overview. DockRDMA contains three parts – DockRDMA driver, DockRDMA daemon, and DockRDMA library. DockRDMA driver extends the existing RDMA driver, with initialization and isolation of vRDMAs added. Different from the physical RDMA interface, each vRDMA has a namespace, and can translate the virtual addresses and QPNs on the control path. Besides, each vRDMA is attached to a policy instance when being initialized. The vRDMA only virtualizes the control path, while the data path directly goes to the RNIC and bypasses the vRDMA. DockRDMA library is slightly modified from the original RDMA library. It adds virtualization of QPNs in the data path to support communication over UD QPs. DockRDMA library keeps the APIs unchanged, thus guaranteeing the transparency to applications. DockRDMA daemon interacts with the cloud manager and writes the input from the cloud manager into a configuration file. This configuration file will be read by the DockRDMA driver when a container is being initialized.

When loaded by the OS, the DockRDMA driver registers a callback function to the network stack (❶). This callback is originally designed to add customized network features in the configuration of Ethernet interfaces flexibly. When the cloud manager wants to create a container supporting RDMA,

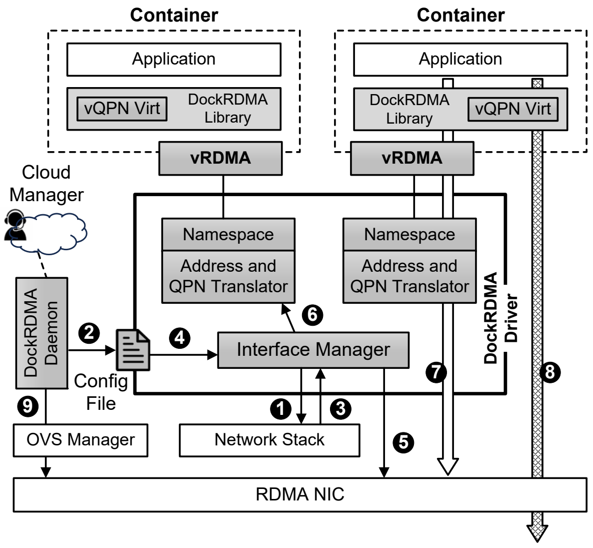
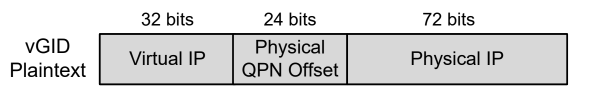


Figure 2. DockRDMA Overview. (❶ Register callback. ❷ Write the configuration of the vRDMA to be initialized. ❸ Invoke the callback of the DockRDMA driver. ❹ Read the configuration. ❺ Write the IP address of the policy instance. ❻ Set the namespace and virtualize the address. ❼ Set up QPs. ❽ Post WQEs. ❾ Configure flow tables.)

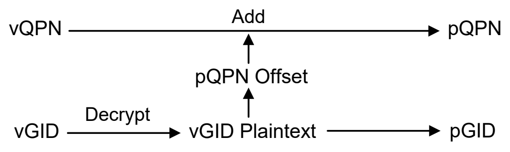
it calls the DockRDMA daemon. The DockRDMA daemon writes the vRDMA's configuration (including the IP address of the policy instance, the capacity of the vRDMA's QPs, etc.) in the configuration file of the new container (❷). During the initialization of the container, the network stack invokes the callback when creating a virtual Ethernet interface (❸). The DockRDMA driver reads the configuration file (❹). If the IP address of the policy instance has not been added to the RNIC, the DockRDMA driver writes the IP address into the RNIC (❺). Then, the DockRDMA driver initializes the vRDMA (❻). The vRDMA initialization includes setting the namespace of vRDMA, virtualizing the address, initializing a QP pool for the vRDMA, and virtualizing the QPN space. Before communicating with others, applications in containers need to set up QPs (❼). For connection-oriented communication like Reliable Connection (RC), the vRDMA on the driver side translates the address and QPN of both the source and destination into the physical ones. For datagram communication like Unreliable Datagram (UD), the vRDMA only translates the source address and QPN. When the container transmits data, the DockRDMA library posts WQEs to the RNIC (❽). DockRDMA leverages the offloaded OVS to enforce network policies. The DockRDMA daemon can enforce policies for an IP of a policy instance by configuring the OVS manager (❾).

B. Virtualization and Translation of Address and QPN

The vRDMAs need to provide containers with the view that they are using their own RNICs, and these RNICs are independent of each other. Thus, each vRDMA should have its own virtual address. As mentioned in §II-C, DockRDMA needs to translate the addresses with minimal overheads. Embedding physical addresses and QPNs into virtual ones is the solution



(a) GID Virtualization



(b) Translation of vGID and vQPN

Figure 3. Virtualization and Translation of GID and QPN

adopted by DockRDMA. Thus, DockRDMA achieves remote side address translation locally with low overhead.

In addition to the virtualization of addresses, QPN virtualization is also needed. In DockRDMA, RNIC only has the physical addresses and QPNs. If QPNs are not virtualized, a container may connect to an arbitrary QPN on an RNIC of a peer server using the virtual address plus an arbitrary QPN. Although using a filter in the host driver can solve this issue, the filter needs to maintain a large table to store all available addresses and QPNs, causing too much overhead. Therefore, DockRDMA also needs to realize the virtualization of QPN and translate the virtual QPNs with minimal overheads.

Different from Ethernet, RDMA uses the address format from the IB network called Global Identifier (GID) and Local Identifier (LID). The GID has 128 bits and the LID has 16 bits. For QPN, the field length is 24 bits in messages [35]. In RoCEv2, the GID field is the IP address and the LID field is 0. To embed the physical GID and QPN in virtual GID (vGID), we need to use the available bits of vGID carefully. The virtual LID (vLID) does not need to be embedded into the vGID as applications designed for RoCEv2 may omit LID fields. The address embedding method should not add any limitations on the number of QPs available to each vRDMA interface, and thus all 24 bits of vQPN are used. To get the physical QPN, the physical QPN offset of a vRDMA is embedded in vGID, and the offset is also 24-bit long (Figure 3(a)). To calculate the pQPN, one can just add the vQPN and the physical QPN offset (Figure 3(b)). For virtual IP, we assume 32 bits are enough for a tenant. Thus, 72 $(128 - 32 - 24)$ bits are available for the physical IP address, which is enough for a provider to manage a cloud even if many addresses (up to tens of thousands) are needed for a single host [36].

DockRDMA needs to translate from virtual GIDs and QPNs to physical ones (and vise versa) at multiple steps in RDMA communication. During QP setup, DockRDMA assigns a virtual QP to the application. When posting a request to QP, the application passes a local vQPN to DockRDMA library. DockRDMA library translates the local vQPN to local pQPN. DockRDMA also needs to translate remote vGIDs and vQPNs to physical ones. The translation happens at different places for different types of communications. For connection-oriented QPs, the translation happens at connection setup. The DockRDMA driver can touch the virtual addresses and vQPNs during communication setup. DockRDMA parses the physical address and QPN, and uses the physical address and QPN to set up the QP. For datagram QPs, DockRDMA translates remote addresses and QPNs in DockRDMA library after the local vQPN translation. The remote address and QPN translation for both types of QPs is done efficiently using the information embedded in virtual ones and stored locally. No extra communication is required.

C. Enforcing Network Policies at Scale

Operators desire to enforce different kinds of policies, including: *1) ACL*: operators discard traffic that illegally accesses the services, and *2) Bandwidth isolation*: operators isolate the bandwidth resource that containers use. Modern commodity RNICs have supported in-line packet processing capabilities, such as OVS hardware offloading. Like the software-based OVS, the hardware-offloaded OVS allows operators to enforce network policies by configuring 5-tuple flow table entries. The hardware OVS can save 256K different 5-tuple flow entries [19]. Modern RNICs support multiple VFs, and each VF can support multiple IP addresses. A VF can configure 256 IP addresses at most [37], and modern RNIC can support 2,000 VFs [38]. Thus, the total supported IPs can be up to hundreds of thousands. Operators can leverage multiple IP addresses to distinguish packets of different policy configurations. Therefore, using the IP addresses offloaded into the VFs to enforce network policies is the solution DockRDMA adopts.

DockRDMA maps each vRDMA to an IP address that is used to identify the logical policy instance (“policy IP”). Thanks to the hybrid virtualization choice of DockRDMA, the policy IP is the physical IP seen by RNIC for communication. User uses virtual GIDs for the container overlay networking. DockRDMA controls the mapping between vGIDs of vRDMAs to the corresponding “policy IPs” to map the vRDMA’s traffic to the corresponding network policies. Users typically require QoS policy enforcement for containers of the same services [39] or the same tenants [40] as a whole. Thus, multiple vRDMAs can be mapped to a “policy IPs” to let apply policies to a group of containers.

The cloud manager maintains the “policy IPs” of all hosts. When creating a new policy, the cloud manager communicates with the DockRDMA Daemons and assigns a “policy IP” to this host. After getting the IP, the DockRDMA Daemon interacts with the DockRDMA driver to write the IP into RNIC. The DockRDMA daemon then offloads flow tables corresponding to the “policy IPs”. When initializing a vRDMA for a group, the DockRDMA driver embeds the corresponding “policy IPs” into the vGID. The vGIDs will be translated into the “policy IPs” during communication.

D. Generality of Initialization

During the creation of a container, the container platforms need to interact with the DockRDMA driver to initialize a vRDMA. However, there are various container platforms, and each of them has many released versions. As a result, offering a general solution without modifying the codes of the existing container platforms is critical.

To achieve the generality, DockRDMA needs to find a point that only resides in the kernel, and is independent of any container platforms. During container creation, any container platform interacts with the host network stack to create a virtual Ethernet interface for the container. To enable kernel developers to add customized network features flexibly, the network stack has provided many callbacks, including those for Ethernet interfaces [41]. This gives us an opportunity to initialize a vRDMA automatically when the network stack creates a virtual Ethernet interface, which is transparent to the container platforms.

DockRDMA needs to tackle three issues in vRDMA initialization – two are about using the callback functions of Ethernet interfaces, and one is the functionality of vRDMA.

1) Passing vRDMA Configuration to Callbacks: The input of the callback functions is unable to be customized. As a result, the vRDMA configuration cannot be passed directly to the callback. DockRDMA solves this issue by using configuration files. When operators want to start a container equipped with RDMA, they first call the DockRDMA daemon, taking the vRDMA's configuration as the input. DockRDMA daemon writes the vRDMA's configuration to the configuration file created by the DockRDMA driver for each container. When setting up a vRDMA, the DockRDMA driver reads the configuration file and initializes the vRDMA.

2) Filtering the Callbacks: The network stack invokes the callback when any event of any Ethernet interface occurs. Ethernet interfaces include the virtual Ethernet interface managed purely in the software, and the interface corresponding to the NIC instances. Events include the creation and deletion of the Ethernet interface, state changes (the interface is brought up or down), interface renaming, etc. DockRDMA needs to check whether the callback is invoked when the network stack has just finished configuring a container's virtual Ethernet interface, or when the network stack is deleting a container's virtual Ethernet interface.

To determine whether the current Ethernet interface is a container's virtual Ethernet interface, DockRDMA first checks the name of the driver that creates the Ethernet interface, which is recorded in the metadata of the Ethernet interface. If the Ethernet interface is created by the virtual Ethernet driver, the current interface is a virtual Ethernet interface. Only checking the name of the driver is unable to filter out the virtual Ethernet interface created manually (e.g., through the "`ip link add`" command). Typically, when a container platform creates a container, the virtual Ethernet interface resides in the namespace other than the host, and it is connected to a bridge. DockRDMA leverages the kernel functions to check its namespace and to identify whether it has been connected to a bridge. With this further checking, DockRDMA can eliminate the virtual Ethernet interfaces created manually, and only handle the ones created by container platforms.

In terms of events, DockRDMA needs to check whether the current event indicates that the network stack has just finished the configuration of a virtual Ethernet interface, or that the network stack is deleting a virtual Ethernet interface. As the callback takes the current event as one of the inputs, DockRDMA can check whether the current event equals the target event. For the event when the network stack is deleting a virtual Ethernet interface, the target event is deletion. For the event when the network stack has just finished configuring a virtual Ethernet interface, container platforms typically call the network stack to create a virtual Ethernet interface, configure it (e.g., setting namespace, connecting it to a bridge, etc.), and then bring it up. Thus, DockRDMA chooses bringing the interface up as the target event. Each configuration file records a flag to indicate whether the vRDMA has been initialized, which is read and written by the DockRDMA driver. When the virtual Ethernet interface is brought up, and the vRDMA has not been created, the DockRDMA driver initializes the vRDMA for the newly created container, and sets the flag. After that, when the virtual Ethernet interface is brought down and up again, the DockRDMA driver will not create a new vRDMA one more time.

E. Security

Many research works have discussed security issues in both containers and RDMA. In terms of containers, some works aimed at prevention from attacks on TCP/IP container networks [42] and system-based attacks [43]. Regarding RDMA, ReDMArk [25] analyzed the vulnerabilities of RDMA, demonstrated the possibility of attacks on RDMA, and provided advice to mitigate the attacks. These works can be integrated into DockRDMA.

Nevertheless, to realize hybrid RDMA virtualization for containers, DockRDMA introduces an extra virtualization layer to the driver, which exposes additional security vulnerabilities. Firstly, the information of RNICs is stored in a shared filesystem (i.e., `sysfs`) that is visible to all containers. This filesystem is used to store the system configuration. It is mounted into a container when the container is being initialized so that containers can directly obtain the system configuration. Without any isolation mechanisms, it is trivial to sniff the information of physical RNICs and other containers' virtual RNICs (e.g., their network addresses). Secondly, the RDMA driver does not know whether the container requesting address translation is benign or malicious. The introduced virtualization layer still translates the virtual address forged by a malicious container. Therefore, the malicious containers may connect to the victim service and launch attacks.

For the first vulnerability, there has been a mature mechanism to isolate the virtual Ethernet interfaces – network namespace. The metadata of each Ethernet interface contains the information of the network namespace. When initializing an Ethernet interface, the kernel checks whether the network namespace equals that of the container [44]. If so, the Ethernet interface is visible to the container. Otherwise, the information on the Ethernet interface is hidden. The kernel realizes the namespace checking function as a callback. Any NIC driver can customize its own namespace checking function. Thereby,

Table II
EVALUATION SETUP

Parameter		Setting
Server	CPU	Two AMD Ryzen 9 3950X 16-Core Processors
	Memory	64 GB
	OS	Ubuntu 16.04
	Kernel	Linux 4.15.0
Container	Image	Ubuntu:16.04
	Platform	• Docker-ce 17.03.0 (default); • LXC 5.19
RNIC	Hardware	• Mellanox CX-6 100 Gbps RoCE (Firmware: 22.30.1004) • MTU: 1,024
	Driver	Mellanox OFED-5.0-1.0.0.0
Open vSwitch		2.14.1

to provide isolation, the DockRDMA driver enables namespace checking for physical and virtual RDMA interfaces. For physical RDMA interfaces, the namespace checking function in the DockRDMA driver directly returns the host namespace. For virtual RDMA interfaces, when the DockRDMA driver initializes a vRDMA in the callback, it reads the namespace from the metadata of the virtual Ethernet interface, and sets the namespace to the metadata of vRDMA. The namespace checking function directly returns the namespace recorded in the vRDMA metadata.

In terms of the second vulnerability, DockRDMA encrypts the virtual network addresses (vGID). The encryption is done when DockRDMA creates a vRDMA and initializes its vGID. When DockRDMA translates the vGID, it decrypts the vGID and parses the physical GID. The length of plaintext and ciphertext needs to be the same as the length of GID. AES [45] is one of the appropriate choices for encryption and decryption. The encryption keys are assigned to each tenant. Only containers of the same tenants can establish RDMA communication with each other. DockRDMA generates a key when a new tenant registers to the clouds, and will assign the identical key to the containers of the same tenant.

IV. EVALUATION

A. Implementation and Evaluation Setup

We implement DockRDMA on Mellanox OFED 5.0-1.0.0.0. $\sim$4,500 lines of C codes are added to the RDMA driver. $\sim$400 lines of C codes are implemented in the RDMA library for virtual address and vQPN translation in the data path. We also implement a simple DockRDMA daemon with $\sim$600 lines of BASH and Python scripts in total.

Table II lists our evaluation setup. Our testbed consists of two servers. Each server is equipped with a Mellanox CX-6 100 Gbps RoCE NIC, an AMD Ryzen 9 3950X 16-core processor, and 64 GB memory. We use the default MTU (1,024) of the RNIC. The servers are connected through an Ethernet switch. They run Linux kernel 4.15.0 on Ubuntu 16.04. Containers are deployed through Docker-ce 17.03.0 and LXC 5.19, the former is the default.

The candidates compared with DockRDMA are native-RDMA, FreeFlow [12], and MasQ [14]. Native-RDMA runs RDMA in the host, without any virtualization layer. FreeFlow is a state-of-the-art RDMA virtualization scheme for containers. Its implementation is based on `libmlx4` library, which only supports CX-3 RNICs. Thus, we port FreeFlow on `libmlx5` library to evaluate it in our testbed. MasQ is a hybrid RDMA virtualization originally designed for virtual machines. The differences between MasQ and DockRDMA are two-fold. First, in MasQ, the remote-side address translation is done by an additional address translation node. As MasQ has not been open-sourced, we modify DockRDMA's code to implement MasQ. The translation is required during connection setup for connection-oriented QPs and each data transfer for datagram QPs. Thus, MasQ does not support communication through UD QPs. To enable UD QPs, we enhance MasQ with an address translation cache of the remote vGID inside the RDMA library. The RDMA library fetches the translation remotely for the first time of translation, and all the subsequent translation of the same vGID is done locally. Second, MasQ leverages VF's rate limiting to enforce QoS policies. The number of policies supported by MasQ is restricted by the number of VFs the RNIC supports.

Our evaluation consists of five parts. *1) Data path performance*. We will verify that DockRDMA realizes hybrid RDMA virtualization, which achieves bare-metal performance in the data path. *2) Communication setup time*. An evaluation will be conducted to measure the communication setup time of DockRDMA, FreeFlow, and MasQ. *3) Generality*. In addition to Docker, containers supporting RDMA will be deployed using other container platforms. We take LXC as an example and explain why DockRDMA can achieve generality. *4) Network policies*. We evaluate the preciseness of enforcing different polices on 300 tenants, and compare the results with MasQ's. *5) Real-world applications*. We choose two applications that are common in cloud, i.e., Hadoop and key-value store. In these applications, both the control path and data path performance are critical.

B. Data Path Performance

First, we evaluate the single-flow goodput and latency using `perftest` [46]. The goodput is 92 Gbps when the network bandwidth is saturated in CX-6 100 Gbps RNICs. RNICs can only send messages with the size up to MTU through a datagram QP. Therefore, the results of UD send only include the message sizes that equal or are below the MTU. Figure 4 shows the results of both RC two-sided verbs, RC one-sided verbs, and UD two-sided verbs. In terms of goodput (Figure 4(a) to 4(c)), DockRDMA achieves the same goodput as native-RDMA for reliable connections. MasQ also adopts hybrid RDMA virtualization, and can also achieve bare-metal performance under reliable communications. Therefore, the evaluation of MasQ's RC send and RC write is omitted. For datagram communications, DockRDMA realizes almost the same goodput as native-RDMA. However, MasQ has performance declines. The major overhead comes from the

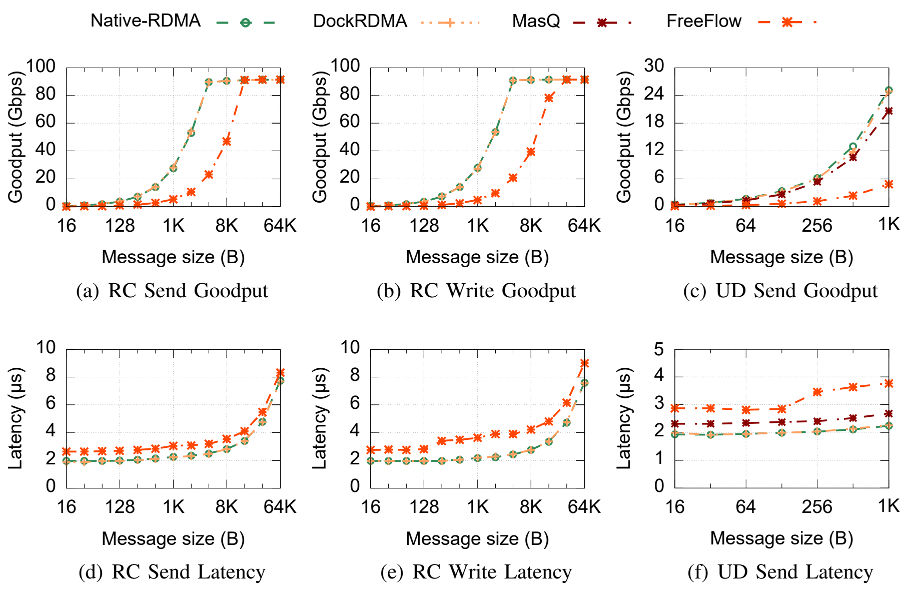


Figure 4. Single-flow Performance Evaluation

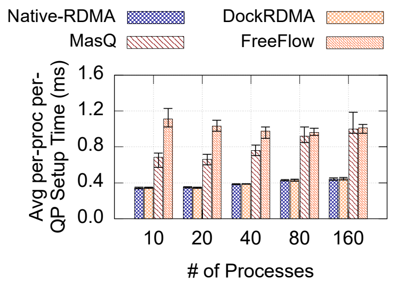


Figure 5. QP Setup Time

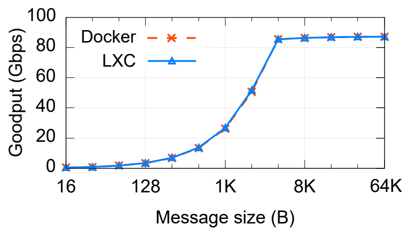


Figure 6. Single-flow Performance against Varying Container Platforms

remote fetching for the first translation. For all the three cases, FreeFlow incurs substantial overheads in the data path, especially when the message size is below 16 KB. When it comes to latency (Figure 4(d) to 4(f)), DockRDMA does not incur extra latency in all the test cases. MasQ incurs extra latency in UD send. FreeFlow has additional latency for all the three cases.

The reasons why DockRDMA exhibits the same performance as native-RDMA are two-fold. For connection-oriented QPs, they have been set up with the peers, so no GID/QPN translation is incurred in the data path. In this case, DockRDMA achieves the bare-metal performance. For datagram QP, although we have overheads in translation, these overheads are small (only several CPU cycles are introduced in around a hundred CPU cycles required for a data path operation), thus negligible. In summary, DockRDMA realizes hybrid virtualization in container scenarios.

C. Communication Setup Time

In the evaluation of communication setup time, we write an application that sets up 100 QPs and records the starting and ending timestamps. A BASH script is also written to start multiple such applications simultaneously, calculate the setup time as the interval between the earliest starting and the latest ending timestamp, and show the average per-process per-QP setup time. The result reflects the time spent by container orchestrators to simultaneously deploy multiple containers running RDMA-based applications on the same host. Figure 5 shows that DockRDMA achieves almost the same QP setup time as native-RDMA, which means the overhead of address translation is negligible. FreeFlow and MasQ incur much more latency when setting up a QP. For FreeFlow, the extra latency comes from the additional memory allocation and memory mapping performed in the control path to keep the zero-copying characteristics of the data path. FreeFlow implements the address translation as a static table. Thus, its address translation overhead is ignored in this evaluation. For MasQ, the extra latency is due to the translation of remote addresses, including communication with the address translation node, the table lookups by the remote node, and the cache walk to find the remote physical addresses locally. As the number of processes increases, the overheads of address translation gradually become dominant in MasQ. The result indicates that when container orchestrators simultaneously provision multiple RDMA-based services on the same host, DockRDMA achieves the same service launching time as native-RDMA, while MasQ takes a much longer time to finish launching services.

D. Generality

Next, we prove that DockRDMA supports different container platforms without modifying the codes of the container platforms. DockRDMA uses ∼200 lines of code to support vRDMA initialization under various container platforms. Thus, without DockRDMA, users need to port those lines of code to each platform, separately. We did not explore how to add the code to each container platform. Instead, we evaluate DockRDMA with the generality support against different container platforms. As Figure 6 shows, DockRDMA achieves the same performance under different container platforms without any code modifications, indicating that we have achieved generality. The reason is that the process of initializing a virtual Ethernet interface of a container by different container platforms (not limited to Docker and LXC) is similar – creating a virtual Ethernet interface, configuring it, and setting it up. DockRDMA leverages the callback of Ethernet interfaces to automatically initialize a vRDMA for the container, which is independent of any specific container platform. Therefore, DockRDMA is compatible with different container platforms.

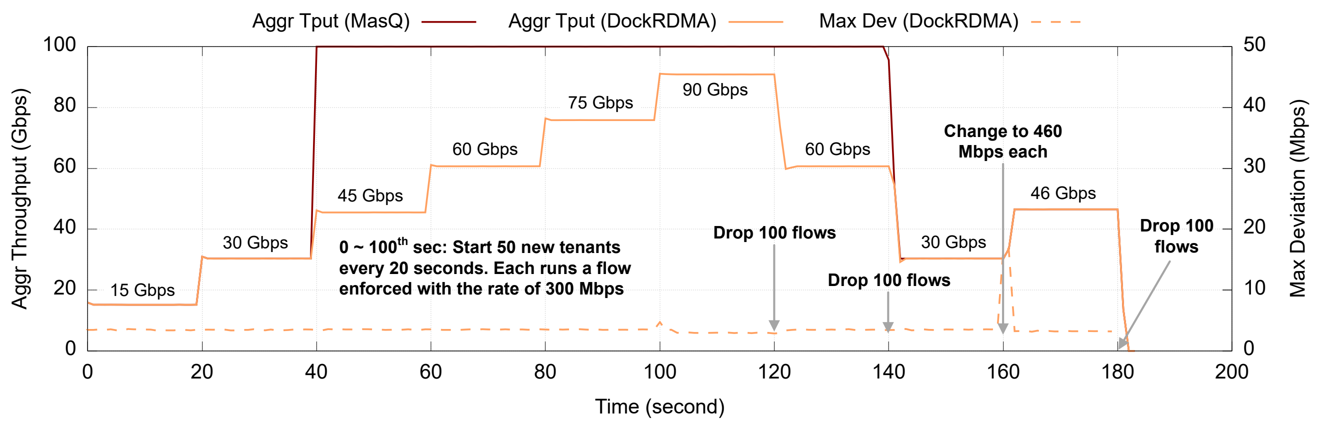


Figure 7. Network Policy Evaluation

E. Network Policies at Scale

This subsection compares DockRDMA with MasQ and validates that DockRDMA supports network policies flexibly. We add 300 tenants in total, each starting a flow by running `perftest`, which runs infinitely and prints the goodput periodically. For MasQ, if the number of tenants exceeds the number of supported policies, we will not enforce any policies for the subsequently added tenants. As `perftest` only shows the bytes of messages per second, we normalize the results and calculate the throughput of RNIC's sending packets. At the beginning, we initialize the first 50 tenants, and enforce a 300-Mbps rate on each tenant. From 0th to 100th seconds, we initialize 50 new tenants every 20 seconds, and each tenant is enforced with a 300-Mbps rate. At 120th and 140th seconds, 100 flows are dropped. Then, at 160th second, we change the rate of each tenant to 460 Mbps. Then, after 20 seconds, we drop all the flows. To evaluate the preciseness of the rate-limiting, we calculate the deviation of the measured throughput of each flow from its ideal rate at every time point, and show the maximum deviation.

Figure 7 indicates that DockRDMA can enforce QoS policies precisely for the tenants. The maximum deviation is negligible. In contrast, MasQ is unable to enforce policies when the number of tenants increases. In our testbed, the RNIC can support up to 127 VFs. Under this configuration, MasQ can only support 127 policy instances. When the number of tenants exceeds that restriction, MasQ cannot enforce rate limits for the new tenants. These tenants saturate the RNIC's bandwidth, resulting in the aggregate throughput at line rate. In this case, the maximum deviation is meaningless, and therefore omitted.

F. Real-world Applications

Finally, we deploy DockRDMA in real-world applications. This evaluation takes two common applications (i.e., Hadoop and key-value store) in cloud as examples. The performance of native-RDMA, DockRDMA, and MasQ are compared.

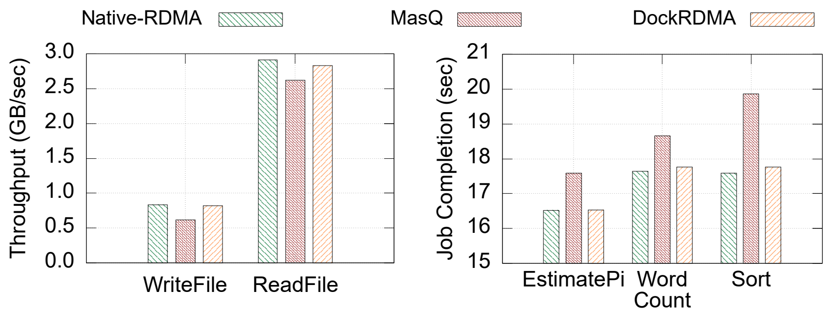
Hadoop File System [47], [48]: Hadoop File System (HDFS) has been applied in many cloud-native applications [31]. Upon each time of job submission, the master distributes computing tasks to the slaves. Each task establishes new connections. In this case, the connection setup time affects the performance of the job execution.

In this evaluation, 16 containers are started on each of the two servers, each using a logical CPU core exclusively. All the containers deploy RDMA-based Apache Hadoop (2.x) published in HiBD project [49]. We first evaluate the throughput of writing and reading a 128-MB file. The evaluation result is shown in Figure 8(a). The read/write throughput of HDFS running on native-RDMA and DockRDMA are almost identical, while the read/write throughput on MasQ degrades. The major overhead comes from the extra time taken to set up new RDMA connections.

Next, we evaluate the completion time of three short-lived jobs, i.e., estimation of π, word count, and sorting. As Figure 8(b) shows, native-RDMA and DockRDMA almost achieve the same JCT (Job Completion Time) in all the three jobs. In contrast, MasQ takes an extra $1 \sim 2$ seconds to complete these jobs. Some jobs may have strict SLOs [50]. Additional seconds of completion time may lead to the violation of SLOs, causing revenue loss.

Key-value Store [51], [52]: Key-value Store (KVS) is another building block in cloud applications. There are many designs of RDMA-based KVSes. In this paper, we choose HERD implemented in `rdma_bench` [53]. In HERD, multiple clients establish long RDMA connections with a single server before data transmission, so the control path overhead is not the dominant factor. However, the clients write the key-value pairs to the server through UC write, and the server responds to clients through UD send. Thus, the address translation overhead may still affect the data path performance.

In the evaluation, we start a container on a host, which is a KVS server running 12 workers. Besides, multiple containers are deployed on the other host and run as KVS clients. Each container starts exactly one client thread to write the key-value pairs, with a key size of 16 bytes and a value size of 32 bytes. Figure 9 illustrates the throughput varying with the number of clients. When the CPU is not saturated, native-RDMA, DockRDMA, and MasQ achieve the same throughput. When the CPU is saturated, MasQ's throughput is 5% lower than



(a) Read/Write Throughput (b) Job Completion Time

Figure 8. Hadoop

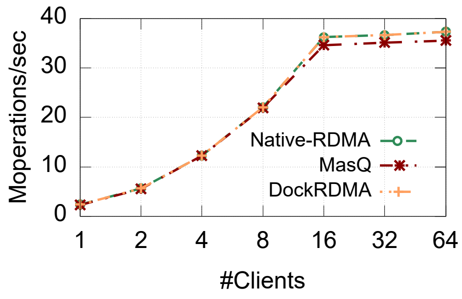


Figure 9. Key-value Store

the other two. MasQ achieves such a performance thanks to the optimization we adopt (§IV-A), without which, MasQ's throughput would be very low. Besides, MasQ achieves the performance close to native-RDMA under a small scale, where the overhead of remote address translation is not the dominant factor. As the scale becomes larger, the RDMA library needs to cache excessively increasing address translation for all other hosts. In this case, the remote address translation of MasQ can be still expensive even with optimization.

V. Related Work

There are some related works other than RDMA virtualization. We class them into four categories.

Container networking: Major container platforms provide various overlay network approaches, such as Flannel [54], Weave [55], Calico [56], and Docker Overlay [57]. For security enforcement, Cilium [58] realizes a security extension to redirect container traffic to its containerized security service. DockRDMA can work with the existing approaches. Recent research mainly focuses on performance optimization and security issues of container networks. Slim [59] designs a low-overhead overlay network by manipulating connection-level metadata to reduce the number of times packets traverse the network stack. Nam et al. [42] reveal the inherent limitations of the existing container networking, and propose a new extension for container network isolation and protection in the host network stack. DockRDMA is orthogonal to them as their effort extends and optimizes container networking which is on the control path of DockRDMA.

Container security: Many works have focused on container security [60], [43], [61], [62], [42]. For example, Dua et al. [60] analyzed various container implementations and found that they still had vulnerabilities in file system, network, and memory isolation. Some works demonstrate the security issues of Docker, including Docker escape attack [43], vulnerabilities of Docker images [61], [62], etc. BASTION [42] focused on the security issues of container networks. DockRDMA leverages the namespace that modern container platforms commonly use to protect the physical and virtual RNICs.

Offloading with SmartNIC/DPU: SmartNICs or DPUs have emerged in data centers, aiming to bridge the gap between increasing network bandwidth and stagnating CPU power [63], [64], [65], [66]. Both academia and industry have proposed to offload packet processing (e.g., encapsulation/decapsulation, stateful ACLs/NAT and metering, QoS policy) or even more complex data center server tasks (e.g., distributed applications, storage) to SmartNIC/DPU for acceleration. The advanced SmartNIC/DPU can benefit DockRDMA by providing even more offload capabilities.

RDMA security: Plenty of research and documents reveal the security issues of RDMA [67], [68], [69], [25]. RFC 5042 [67] analyzed the security issues of RDMA protocols and provided a guideline for designing protocols based on RDMA. Kurth et al. [68] and Tsai et al. [69] demonstrated side-channel attacks exploited by RNICs and capabilities of CPUs like Intel DDIO technology. ReDMArk [25] testified the applicability of possible threat models in RDMA to specific implementations of RDMA, and offered solutions to mitigate the attacks. DockRDMA mainly focuses on the extra security vulnerabilities exposed by the newly introduced virtualization layer.

VI. Conclusion

Supporting RDMA in containerized clouds is important for cloud platforms. This paper continues the design philosophy of hybrid RDMA virtualization, but solves the unique challenges when adopting it in containers. To the best of our knowledge, DockRDMA is the first hybrid RDMA virtualization solution specifically designed for containerized cloud platforms. Our evaluation proves the benefits of DockRDMA, including bare-metal performance in the data path, negligible overheads in communication setup, seamless coordination with existing container platforms, and scalability to meet hundreds of different network policies.

Acknowledgments

The authors gratefully acknowledge the anonymous reviewers and our shepherd, Guyue (Grace) Liu, for their constructive comments and suggestions. This work is supported in part by the National Key Research and Development Program of China (No. 2022YFB2901404), and by National Natural Science Foundation of China (NSFC) under Grant No. 62132007 and No. 62221003.

References

[1] Redis Ltd, "Redis," https://redis.io/, 2024.

[2] B. Fitzpatrick, "Distributed Caching with Memcached," *Linux Journal*, vol. 124, no. 5, 2004.

[3] AWS, "Amazon Elastic Container Service," https://aws.amazon.com/en/ecs/, 2021.

[4] M. Azure, "Container Instances," https://azure.microsoft.com/en-us/products/container-instances/, 2021.

[5] G. Cloud, "Containers at Google," https://cloud.google.com/containers, 2021.

[6] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: A System for Large-Scale Machine Learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, Nov. 2016, pp. 265–283. [Online]. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi

[7] Y. Gao, Q. Li, L. Tang, Y. Xi, P. Zhang, W. Peng, B. Li, Y. Wu, S. Liu, L. Yan, F. Feng, Y. Zhuang, F. Liu, P. Liu, X. Liu, Z. Wu, J. Wu, Z. Cao, C. Tian, J. Wu, J. Zhu, H. Wang, D. Cai, and J. Wu, "When Cloud Storage Meets RDMA," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, Apr. 2021, pp. 519–533. [Online]. Available: https://www.usenix.org/conference/nsdi21/presentation/gao

[8] B. Zhu, Y. Chen, Q. Wang, Y. Lu, and J. Shu, "Octopus+: An RDMA-Enabled Distributed Persistent Memory File System," *ACM Trans. Storage*, vol. 17, no. 3, Aug. 2021. [Online]. Available: https://doi.org/10.1145/3448418

[9] W. Cao, Y. Liu, Z. Cheng, N. Zheng, W. Li, W. Wu, L. Ouyang, P. Wang, Y. Wang, R. Kuan, Z. Liu, F. Zhu, and T. Zhang, "POLARDB Meets Computational Storage: Efficiently Support Analytical Workloads in Cloud-Native Relational Database," in *18th USENIX Conference on File and Storage Technologies (FAST 20)*. Santa Clara, CA: USENIX Association, Feb. 2020, pp. 29–41. [Online]. Available: https://www.usenix.org/conference/fast20/presentation/cao-wei

[10] E. Zamanian, C. Binnig, T. Harris, and T. Kraska, "The End of a Myth: Distributed Transactions Can Scale," *Proc. VLDB Endow.*, vol. 10, no. 6, p. 685–696, Feb. 2017. [Online]. Available: https://doi.org/10.14778/3055330.3055335

[11] A. Ranadive and B. Davda, "Toward a Paravirtual vRDMA Device for VMware ESXi Guests," *VMware Technical Journal, Winter 2012*, vol. 1, no. 2, Dec. 2012.

[12] D. Kim, T. Yu, H. H. Liu, Y. Zhu, J. Padhye, S. Raindel, C. Guo, V. Sekar, and S. Seshan, "FreeFlow: Software-based Virtual RDMA Networking for Containerized Clouds," in *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. Boston, MA: USENIX Association, Feb. 2019, pp. 113–126. [Online]. Available: https://www.usenix.org/conference/nsdi19/presentation/kim

[13] J. Pfefferle, P. Stuedi, A. Trivedi, B. Metzler, I. Koltsidas, and T. R. Gross, "A Hybrid I/O Virtualization Framework for RDMA-Capable Network Interfaces," in *Proceedings of the 11th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, ser. VEE '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 17–30. [Online]. Available: https://doi.org/10.1145/2731186.2731200

[14] Z. He, D. Wang, B. Fu, K. Tan, B. Hua, Z.-L. Zhang, and K. Zheng, "MasQ: RDMA for Virtual Private Cloud," in *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1–14. [Online]. Available: https://doi.org/10.1145/3387514.3405849

[15] X. Wei, F. Lu, R. Chen, and H. Chen, "KRCORE: A Microsecond-scale RDMA Control Plane for Elastic Computing," in *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. Carlsbad, CA: USENIX Association, Jul. 2022, pp. 121–136. [Online]. Available: https://www.usenix.org/conference/atc22/presentation/wei

[16] H. Zhang, A. Cardoza, P. B. Chen, S. Angel, and V. Liu, "Fault-tolerant and Transactional Stateful Serverless Workflows," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, Nov. 2020, pp. 1187–1204. [Online]. Available: https://www.usenix.org/conference/osdi20/presentation/zhang-haoran

[17] Z. Jia and E. Witchel, "Boki: Stateful Serverless Computing with Shared Logs," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, ser. SOSP '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 691–707. [Online]. Available: https://doi.org/10.1145/3477132.3483541

[18] F. Manco, C. Lupu, F. Schmidt, J. Mendes, S. Kuenzer, S. Sati, K. Yasukata, C. Raiciu, and F. Huici, "My VM is Lighter (and Safer) than Your Container," in *Proceedings of the 26th Symposium on Operating Systems Principles*, ser. SOSP '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 218–233. [Online]. Available: https://doi.org/10.1145/3132747.3132763

[19] NVIDIA, "OVS Offload Using ASAP2 Direct," https://docs.nvidia.com/networking/display/mlnxofedv582030lts/ovs+offload+using+asap%C2%B2+direct, 2023.

[20] Y. Chen, X. Wei, J. Shi, R. Chen, and H. Chen, "Fast and General Distributed Transactions Using RDMA and HTM," in *Proceedings of the Eleventh European Conference on Computer Systems*, ser. EuroSys '16. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/2901318.2901349

[21] X. Wei, Z. Dong, R. Chen, and H. Chen, "Deconstructing RDMA-Enabled Distributed Transactions: Hybrid is Better," in *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'18. USA: USENIX Association, 2018, p. 233–251.

[22] M. Wu, F. Yang, J. Xue, W. Xiao, Y. Miao, L. Wei, H. Lin, Y. Dai, and L. Zhou, "GraM: Scaling Graph Computation to the Trillions," in *Proceedings of the Sixth ACM Symposium on Cloud Computing*, ser. SoCC '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 408–421. [Online]. Available: https://doi.org/10.1145/2806777.2806849

[23] H. Chen, C. Li, C. Zheng, C. Huang, J. Fang, J. Cheng, and J. Zhang, "G-Tran: A High Performance Distributed Graph Database with a Decentralized Architecture," *Proc. VLDB Endow.*, vol. 15, no. 11, p. 2545–2558, sep 2022. [Online]. Available: https://doi.org/10.14778/3551793.3551813

[24] J. Shi, Y. Yao, R. Chen, H. Chen, and F. Li, "Fast and Concurrent RDF Queries with RDMA-Based Distributed Graph Exploration," ser. OSDI'16. USA: USENIX Association, 2016, p. 317–332.

[25] B. Rothenberger, K. Taranov, A. Perrig, and T. Hoefler, "ReDMArk: Bypassing RDMA Security Mechanisms," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 4277–4292. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/rothenberger

[26] L. Liss, "Containing RDMA and High Performance Computing," https://events.static.linuxfound.org/sites/events/files/slides/containing_rdma_final.pdf, 2015.

[27] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The Design and Implementation of Open vSwitch," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. Oakland, CA: USENIX Association, May 2015, pp. 117–130. [Online]. Available: https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/pfaff

[28] PCI-SIG, "Sigle Root I/O Virtualization," https://pcisig.com/specifications/iov/single_root/, 2021.

[29] Z. Wang, T. Ma, L. Kong, Z. Wen, J. Li, Z. Song, Y. Lu, G. Chen, and W. Cao, "Zero Overhead Monitoring for Cloud-native Infrastructure using RDMA," in *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. Carlsbad, CA: USENIX Association, Jul. 2022, pp. 639–654. [Online]. Available: https://www.usenix.org/conference/atc22/presentation/wang-zhe

[30] Z. Shen, Z. Sun, G.-E. Sela, E. Bagdasaryan, C. Delimitrou, R. Van Renesse, and H. Weatherspoon, "X-Containers: Breaking Down Barriers to Improve Performance and Isolation of Cloud-Native Containers," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 121–135. [Online]. Available: https://doi.org/10.1145/3297858.3304016

[31] Q. Li, L. Chen, X. Wang, S. Huang, Q. Xiang, Y. Dong, W. Yao, M. Huang, P. Yang, S. Liu, Z. Zhu, H. Wang, H. Qiu, D. Liu, S. Liu, Y. Zhou, Y. Wu, Z. Wu, S. Gao, C. Han, Z. Luo, Y. Shao,

G. Tian, Z. Wu, Z. Cao, J. Wu, J. Shu, J. Wu, and J. Wu, "Fisc: A Large-scale Cloud-native-oriented File System," in *21st USENIX Conference on File and Storage Technologies (FAST 23)*. Santa Clara, CA: USENIX Association, Feb. 2023, pp. 231–246. [Online]. Available: https://www.usenix.org/conference/fast23/presentation/li-qiang-fisc

[32] P. P. Dror Goldenberg, "Mellanox Container Journey," https://qnib.org/data/hpcw19/7_END_2_MellanoxJourney.pdf, 2019.

[33] X. Li, R. Shu, Y. Xiong, and F. Ren, "Software-based Live Migration for Containerized RDMA," in *Proceedings of the 8th Asia-Pacific Workshop on Networking*, ser. APNet '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 52–58. [Online]. Available: https://doi.org/10.1145/3663408.3663416

[34] R. Russell, "virtio: Towards a De-Facto Standard for Virtual I/O Devices," *SIGOPS Oper. Syst. Rev.*, vol. 42, no. 5, p. 95–103, jul 2008. [Online]. Available: https://doi.org/10.1145/1400097.1400108

[35] "Infiniband Architecture Specification Volume 1," https://www.afs.enea.it/asantoro/V1r1_2_1.Release_12062007.pdf, 2007.

[36] L. Salamatian, T. Arnold, I. Cunha, J. Zhu, Y. Zhang, E. Katz-Bassett, and M. Calder, "Who Squats IPv4 Addresses?" *SIGCOMM Comput. Commun. Rev.*, vol. 53, no. 1, p. 48–72, April 2023. [Online]. Available: https://doi.org/10.1145/3594255.3594260

[37] M. Technology, "RoCEv2 GID Disappeared?" https://forums.developer.nvidia.com/t/rocev2-gid-disappeared/206970, 2018.

[38] NVIDIA, "NVIDIA ConnectX-7 400G Ethernet," https://solutions.asbis.com/api/uploads/files/40/connectx-7-datasheet.pdf.

[39] B. Raghavan, K. Vishwanath, S. Ramabhadran, K. Yocum, and A. C. Snoeren, "Cloud Control with Distributed Rate Limiting," in *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 337–348. [Online]. Available: https://doi.org/10.1145/1282380.1282419

[40] L. Chen, X. Jiang, X. Hu, T. Xu, Y. Yang, X. Li, B. Lu, C. Wei, and W. Chen, "CMDRL: A Markovian Distributed Rate Limiting Algorithm in Cloud Networks," in *Proceedings of the 8th Asia-Pacific Workshop on Networking*, ser. APNet '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 59–66. [Online]. Available: https://doi.org/10.1145/3663408.3663417

[41] torvalds/linux, https://github.com/torvalds/linux/blob/v4.15/net/core/dev.c#L1570.

[42] J. Nam, S. Lee, H. Seo, P. Porras, V. Yegneswaran, and S. Shin, "BASTION: A Security Enforcement Network Stack for Container Networks," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, Jul. 2020, pp. 81–95. [Online]. Available: https://www.usenix.org/conference/atc20/presentation/nam

[43] Z. Jian and L. Chen, "A Defense Method against Docker Escape Attack," in *Proceedings of the 2017 International Conference on Cryptography, Security and Privacy*, ser. ICCSP '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 142–146. [Online]. Available: https://doi.org/10.1145/3058060.3058085

[44] torvalds/linux, https://github.com/torvalds/linux/blob/v4.15/net/core/net-sysfs.c#L1512.

[45] tutorialspoint, "Advanced Encryption Standard," https://www.tutorialspoint.com/cryptography/advanced_encryption_standard.htm.

[46] "GitHub - linux-rdma/perftest: Infiniband Verbs Performance Tests," https://github.com/linux-rdma/perftest.

[47] A. Hadoop, https://hadoop.apache.org/, 2021.

[48] N. S. Islam, D. Shankar, X. Lu, M. Wasi-Ur-Rahman, and D. K. Panda, "Accelerating I/O Performance of Big Data Analytics on HPC Clusters through RDMA-Based Key-Value Store," in *2015 44th International Conference on Parallel Processing*, 2015, pp. 280–289.

[49] N.-B. C. Laboratory, "High-Performance Big Data (HiBD)," http://hibd.cse.ohio-state.edu/, 2021.

[50] Z. Bai, Z. Zhang, Y. Zhu, and X. Jin, "PipeSwitch: Fast Pipelined Context Switching for Deep Learning Applications," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, Nov. 2020, pp. 499–514. [Online]. Available: https://www.usenix.org/conference/osdi20/presentation/bai

[51] "RDMA-Bench," https://github.com/efficient/rdma_bench, 2019.

[52] A. Kalia, M. Kaminsky, and D. G. Andersen, "Using RDMA Efficiently for Key-Value Services," in *Proceedings of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 295–306. [Online]. Available: https://doi.org/10.1145/2619239.2626299

[53] ——, "Design Guidelines for High Performance RDMA Systems," in *2016 USENIX Annual Technical Conference (USENIX ATC 16)*. Denver, CO: USENIX Association, Jun. 2016, pp. 437–450. [Online]. Available: https://www.usenix.org/conference/atc16/technical-sessions/presentation/kalia

[54] F. I/O, "Flannel," https://github.com/flannel-io/flannel, 2023.

[55] weaveworks, "Weave GitOps: Continuous Deployment and Operations with GitOps," https://www.weave.works/, 2014.

[56] Calico, "What is Project Calico?" https://www.tigera.io/project-calico/, 2023.

[57] Docker, "Use Overlay Networks," https://docs.docker.com/network/overlay/, 2013.

[58] cilium, "eBPF-based Networking, Observability, Security," https://cilium.io/.

[59] D. Zhuo, K. Zhang, Y. Zhu, H. H. Liu, M. Rockett, A. Krishnamurthy, and T. Anderson, "Slim: OS Kernel Support for a Low-Overhead Container Overlay Network," in *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. Boston, MA: USENIX Association, Feb. 2019, pp. 331–344. [Online]. Available: https://www.usenix.org/conference/nsdi19/presentation/zhuo

[60] R. Dua, A. R. Raja, and D. Kakadia, "Virtualization vs Containerization to Support PaaS," in *2014 IEEE International Conference on Cloud Engineering*, 2014, pp. 610–614.

[61] R. Shu, X. Gu, and W. Enck, "A Study of Security Vulnerabilities on Docker Hub," in *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, ser. CODASPY '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 269–280. [Online]. Available: https://doi.org/10.1145/3029806.3029832

[62] B. Tak, C. Isci, S. Duri, N. Bila, S. Nadgowda, and J. Doran, "Understanding Security Implications of Using Containers in the Cloud," in *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. Santa Clara, CA: USENIX Association, Jul. 2017, pp. 313–319. [Online]. Available: https://www.usenix.org/conference/atc17/technical-sessions/presentation/tak

[63] D. Firestone, A. Putnam, S. Mundkur, D. Chiou, A. Dabagh, M. Andrewartha, H. Angepat, V. Bhanu, A. Caulfield, E. Chung, H. K. Chandrappa, S. Chaturmohta, M. Humphrey, J. Lavier, N. Lam, F. Liu, K. Ovtcharov, J. Padhye, G. Popuri, S. Raindel, T. Sapre, M. Shaw, G. Silva, M. Sivakumar, N. Srivastava, A. Verma, Q. Zuhair, D. Bansal, D. Burger, K. Vaid, D. A. Maltz, and A. Greenberg, "Azure Accelerated Networking: SmartNICs in the Public Cloud," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. Renton, WA: USENIX Association, Apr. 2018, pp. 51–66. [Online]. Available: https://www.usenix.org/conference/nsdi18/presentation/firestone

[64] M. Liu, T. Cui, H. Schuh, A. Krishnamurthy, S. Peter, and K. Gupta, "Offloading Distributed Applications onto SmartNICs using iPIPE," in *Proceedings of the ACM Special Interest Group on Data Communication*, 2019, pp. 318–333.

[65] J. Lin, K. Patel, B. E. Stephens, A. Sivaraman, and A. Akella, "PANIC: A High-performance Programmable NIC for Multi-tenant Networks," in *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation*, 2020, pp. 243–259.

[66] S. Novakovic, Y. Shan, A. Kolli, M. Cui, Y. Zhang, H. Eran, B. Pismenny, L. Liss, M. Wei, D. Tsafrir, and M. Aguilera, "Storm: A Fast Transactional Dataplane for Remote Data Structures," in *Proceedings of the 12th ACM International Conference on Systems and Storage*, ser. SYSTOR '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 97–108. [Online]. Available: https://doi.org/10.1145/3319647.3325827

[67] J. Pinkerton and E. Deleganes, "Direct Data Placement Protocol (DDP) / Remote Direct Memory Access Protocol (RDMAP) Security," https://www.rfc-editor.org/rfc/pdfrfc/rfc5042.txt.pdf, 2007.

[68] M. Kurth, B. Gras, D. Andriesse, C. Giuffrida, H. Bos, and K. Razavi, "NetCAT: Practical Cache Attacks from the Network," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 20–38.

[69] S.-Y. Tsai, M. Payer, and Y. Zhang, "Pythia: Remote Oracles for the Masses," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 693–710. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/tsai