Evaluating the Use of RDMA in High Performance Containers

Honors Computer Science Bachelor's Thesis Proposal Document

Author: Emerson Ford

Advisor: Ryan Stutsman

Honors Faculty Advisor: Tom Henderson

Director of Undergraduate Studies: James de St. Germain

June 27, 2021

Abstract

Containers are an increasingly popular packaging framework for complex applications, offering benefits such as lightweight isolation, portability, and ease of deployment. These benefits have the potential to solve a myriad of issues that have long been present in the high performance computing world, presenting a compelling narrative for their use in these environments. Unfortunately, standard container runtimes rely on relatively slow, virtualized network stacks to achieve network isolation and portability, which are incompatible with the kernel bypass networking technology RDMA. This has limited the widespread adoption of containers in high performance computing environments where RDMA-reliant applications are quite common.

Fortunately, recent projects such as Microsoft's FreeFlow and Mellanox's RDMA hardware have created solutions for enabling RDMA inside of containers while maintaining varying degrees of the benefits of standard containers. However, despite the strong claims made by these solutions, many of their characteristics such as performance, isolation sacrifices, and scalability are either not well documented or simply not known; these are characteristics that are critical to assess for several high performance computing use cases. This thesis attempts to remedy this issue by identifying, measuring, and comparing these characteristics for the various solutions available today for enabling RDMA in containers; ultimately providing high performance computing end-users a more holistic perspective on which solutions may be most viable for their environment and use case.

1 Introduction

Containers offer a promising solution to a host of issues that have long plagued the high performance computing (HPC) world from dependency management to portable and reproducible environments to even lightweight sharing of hardware resources; all this is achieved through the use of statically built images and standardized container runtimes [60]. For users, cluster administrators, and application developers, this would provide more freedom and ease in application development and deployment as it eliminates the burden of working around the hundreds of possible environments on which the application may be deployed. As an added benefit, this also opens the door for easy migration and utilization of new HPC platforms such as the cloud, Kubernetes, and SLATE that offer new compute and scheduling capabilities and treat containers as first-class objects [CITE NEEDED].

However, despite these numerous benefits, the adoption of containers in the HPC world has been notably slow [CITE NEEDED]. Among other issues like the until-recent lack of rootless container runtimes, native containers do not support Remote Direct Memory Access (RDMA), a networking technology that provides extremely low latency and high throughput with minimal CPU overhead by offloading memory accesses from the CPU to NIC hardware (hence the term "kernel bypass networking") [54]. This lack of support heavily restricts the use of native containers in HPC environments as trends like disaggregated storage, multi-machine workloads, and increased use of parallel libraries such as MPI have necessitated the use of high performance networking like RDMA to maintain performance. To make the lack of RDMA support even worse, standard con-

tainer network stacks by themselves have been shown to be both CPU intensive and significantly reduce container networking performance [2], further worsening the networking downsides of containers in HPC environments.

While it is possible to run containers without container networking by exposing the host's NIC directly to running containers (a mode called "host networking"), this comes with several notable disadvantages: (1) controlling fair sharing of the RDMA NIC between multiple containers cannot be done programmatically, (2) running containers are not relocatable/portable across multiple hosts, (3) common container orchestration tools like Kubernetes cannot be used, and (4) network isolation and network routing policies for containers is disabled. Thus, while this mode may be a viable for some environments, it is not adequate for the vast majority of container environments — such as those that utilize Kubernetes heavily — and additional solutions for enabling RDMA in containers are still needed.

Unfortunately, enabling support for RDMA in containers (and fast networking as a whole) is a nontrivial task. Container runtimes rely on "virtualized" network stacks for creating flexible network overlays, enforcing isolation, and providing portability [CITE NEEDED]. These network stacks have, until recently, only been expressible through software layers provided by the host kernel, adding significant software overhead to the processing of all network packets originating from and destined to containers. When provided by software alone, these network stacks are fundamentally incompatible with RDMA which requires bypassing the host kernel entirely for its performance guarantees.

Despite this, multiple groups have recently developed solutions for enabling RDMA in containers, each with certain sacrifices in container and/or RDMA guarantees to over-

come this incompatibility. The challenge is then to determine which solution (and subsequent sacrifices in guarantees) works best for a given environment. This is the core problem this thesis seeks to address. It is currently unnecessarily difficult to pick a solution due to a lack of exhaustive performance data and thorough explanations on exactly which guarantees are sacrificed for these various solutions. This information is critical in HPC environments as applications may heavily rely on RDMA's low latency and/or high throughput guarantees; similarly, container security and isolation guarantees may be a requirement for those HPC environments that are multi-tenant or are running untrusted applications.

Therefore, this thesis will analyze multiple properties of the several solutions available today for enabling RDMA in containers, such as SoftRoCE [39], Microsoft's FreeFlow [25], MasQ [20], and Mellanox's Shared HCA/SRIOV [52]; these properties being grouped in two main categories:

Container Network Properties:

- *Network Isolation*: each container has its own interface, port space, etc (i.e. network namespace); further, a container's IP should not inherently depend on the underlying NIC's IP
- *Controllability*: enforcing admission control, routing policies, and traffic shaping on a container with an RDMA NIC
- *Resource Utilization*: container networks should not be CPU or memory intensive

RDMA Properties:

- *Throughput*: how closely can a solution match host RDMA throughput
- *Latency*: what latency overhead is incurred per message for a given solution

and analyzed with varying levels of:

- number of containers present across the cluster and on a single host
- message size of network payloads
- messages sent per second

which should adequately test the scalability of these properties. Finally, comments will also be made on:

- *Proprietary*: can a solution be used across various RDMA NIC vendors
- *Maturity*: what support exists for a solution, how well tested is a solution
- *Ease in Deployment*: are application changes necessary for a solution, how difficult is it to deploy the solution
- *Execution Privileges*: can the solution be used without elevated capabilities such as `CAP_SYS_ADMIN`
- *Network Pressure*: what additional pressure do these solutions put on the network

to assist in determining if a given environment can even support a given solution.

Additionally, these solutions can be generalized to three implementations: pure software, paravirtualization-like, and pure hardware; with each solution in an implementation likely having similar properties. Thus, this thesis should also provide general insight as to which implementations may be best suited for a given HPC environment.

2 Background

2.1 Networking Planes

For both clarity on the terms to be used as well as to facilitate better understanding of RDMA and container networking fundamentals, the concept of the control plane and data plane of networking will be explained.

Control Plane handles connection state (e.g. initiating, maintaining, and terminating connections), routing policies, admission control, etc. Systems such as firewalls, routing tables, `iptables` / `nftables`, interface management, and IP assignment (e.g. DHCP) are all considered as part of the control plane.

Data Plane (also known as the forwarding plane) handles the actual transmission of network data. Systems such as memory copies into network buffers, packet transmission, and traffic shaping are considered as part of the data plane.

2.2 RDMA Networking

Remote Direct Memory Access (RDMA) is a network protocol designed for extremely fast and high throughput access to memory regions on remote hosts with minimal CPU overhead, usually run on top of the Infiniband network fabric or Ethernet (via RoCE). RDMA can provide a substantial performance boost for inter-machine memory sharing and synchronization, as well as savings on CPU cycles that would otherwise be used for network operations, both . As a result, support for RDMA can now be found in the vast majority of parallel applications and libraries, such as TensorFlow and OpenMPI, and has

become ubiquitously used in the high performance computing world [25] [ADDITIONAL CITE].

At its core, RDMA is a form of kernel-bypass networking, meaning it shifts all data plane operations from the kernel to the network card (also called the HCA in RDMA terminology). This eliminating bottlenecks that arise from context switches and potential kernel memory copies.

Control plane operations are also moved into interactions with the sysfs / `udev` systems on Linux.

current top-of-the-line RDMA enabled NICs are capable of achieving sub-1µs latency with >100Gb/s flows [1] [ADDITIONAL CITE]. Due these impressive latency and throughput characteristics, it has become widely adopted across the high performance computing world [25] [ADDITIONAL CITE].

2.3 Containers and Container Networking

2.4 Related Work

2.4.1 RDMA Namespaces and Cgroups

With namespaces and cgroups being the standard systems for container isolation in the Linux kernel, there have been developments to add RDMA implementations of these to the Linux kernel itself [8] [44] [41].

References

Containers

[7] Joe Breen et al. "Building the SLATE Platform". In: *Proceedings of the Practice and Experience on Advanced Research Computing*. Association for Computing Machinery, 2018. DOI: 10.1145/3219104.3219144. URL: https://doi.org/10.1145/3219104.3219144.

[28] John Kreisa. *Introducing the Docker Index: Insight from the World's Most Popular Container Registry*. Docker. Feb. 4, 2020. URL: https://www.docker.com/blog/introducing-the-docker-index/.

[46] *SLATE Homepage*. SLATE CI. URL: https://slateci.io.

[59] A. J. Younge et al. "A Tale of Two Systems: Using Containers to Deploy HPC Applications on Supercomputers and Clouds". In: *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. 2017, pp. 74–81. DOI: 10.1109/CloudCom.2017.40.

[60] Andrew J. Younge. "Containers in HPC". In: *Workshop on NSF and DOE High Performance Computing Tools*. Sandia National Laboratories. July 11, 2019. URL: https://oaciss.uoregon.edu/NSFDOE19/talks/nsfdoe_workshop_ajy.pdf.

Container Networking

[2] Ubaid Abbasi et al. "A performance comparison of container networking alternatives". In: *IEEE Network* 33.4 (2019), pp. 178–185.

[4] Gabriele Ara et al. “Comparative Evaluation of Kernel Bypass Mechanisms for High-performance Inter-container Communications.” In: *CLOSER*. 2020, pp. 44–55.

[5] Gabriele Ara et al. “On the use of kernel bypass mechanisms for high-performance inter-container communications”. In: *International Conference on High Performance Computing*. Springer. 2019, pp. 1–12.

[6] Gulsum Atici and Pinar Sarisaray Boluk. “A Performance Analysis of Container Cluster Networking Alternatives”. In: *Proceedings of the 2nd International Conference on Industrial Control Network And System Engineering Research*. 2020, pp. 10–17.

[12] Tyler Duzan. “How Container Networking Affects Database Performance”. In: (Mar. 18, 2020). URL: https://www.percona.com/blog/2020/03/18/how-container-networking-affects-database-performance/.

[14] Anusha Ginka and Venkata Satya Sameer Salapu. *Optimization of Packet Throughput in Docker Containers*. 2019.

[24] Narūnas Kapočius. “Performance Studies of Kubernetes Network Solutions”. In: *2020 IEEE Open Conference of Electrical, Electronic and Information Sciences (eStream)*. IEEE. 2020, pp. 1–6.

[29] Kyungwoon Lee, Youngpil Kim, and Chuck Yoo. “The impact of container virtualization on network performance of IoT devices”. In: *Mobile Information Systems* 2018 (2018).

[30] Jiaxin Lei et al. “Tackling parallelization challenges of kernel network stack for container overlay networks”. In: *11th {USENIX} Workshop on Hot Topics in Cloud Computing (HotCloud 19)*. 2019.

[47] Kyoungjae Sun, Hyunsik Yang, and Wangbong Lee. *Considerations for Benchmarking Network Performance in Containerized Infrastructure*.

[57] Miguel G Xavier et al. “Performance evaluation of container-based virtualization for high performance computing environments”. In: *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. IEEE. 2013, pp. 233–240.

[62] Yang Zhao et al. “Performance of container networking technologies”. In: *Proceedings of the Workshop on Hot Topics in Container Networking and Networked Systems*. 2017, pp. 1–6.

RDMA in Containers

[3] Jacob Anders et al. “RDMA Enabled Kubernetes for High Performance Computing”. In: *KubeCon 2019*.

[9] Alibaba Cloud. *Using RDMA on Container Service for Kubernetes*. Feb. 20, 2019. URL: https://medium.com/@Alibaba_Cloud/using-rdma-on-container-service-for-kubernetes-c7a4484c22b5.

[20] Zhiqiang He et al. “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. 2020, pp. 1–14.

[21] Yang Hu, Mingcong Song, and Tao Li. "Towards "Full Containerization" in Containerized Network Function Virtualization". In: *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*. 2017, pp. 467–481.

[25] Daehyeok Kim et al. "FreeFlow: Software-based Virtual {RDMA} Networking for Containerized Clouds". In: *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*. 2019, pp. 113–126.

[31] Coleman Link et al. "Container Orchestration by Kubernetes for RDMA Networking". In: *2019 IEEE 27th International Conference on Network Protocols (ICNP)*. IEEE. 2019, pp. 1–2.

[32] Liran Liss. *Containing RDMA and high performance computing*. 2015.

[42] Maksym Planeta et al. "TardiS: Migrating Containers with RDMA Networks". In: *arXiv preprint arXiv:2009.06988* (2020).

[55] Animesh Trivedi, Bernard Metzler, and Patrick Stuedi. "A case for RDMA in clouds: turning supercomputer networking into commodity". In: *Proceedings of the Second Asia-Pacific Workshop on Systems*. 2011, pp. 1–5.

[56] Dongyang Wang et al. "vSocket: virtual socket interface for RDMA in public clouds". In: *Proceedings of the 15th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*. 2019, pp. 179–192.

RDMA Specific Research

[16] Chuanxiong Guo et al. "RDMA over commodity ethernet at scale". In: *Proceedings of the 2016 ACM SIGCOMM Conference*. 2016, pp. 202–215.

[35] Radhika Mittal et al. "Revisiting network support for RDMA". In: *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. 2018, pp. 313–326.

[39] Shailesh Mani Pandey and Rajath Shashidhara. *SROCE: Software RDMA over Commodity Ethernet*.

[40] Parav Pandit. "RDMA Containers Update". In: *Proceedings of the 14th OFA Workshop*. Mellanox. Apr. 2018. URL: https://www.openfabrics.org/images/2018workshop/presentations/115_PPandit_RoCEContainers.pdf.

[41] Parav Pandit and Dror Goldenberg. "RDMA Containers Update". In: *Proceedings of the 2018 ISC High Performance Container Workshop Conference*. Mellanox. 2018. URL: http://qnib.org/data/isc2018/roce-containers.pdf.

[43] Haonan Qiu et al. "Toward effective and fair RDMA resource sharing". In: *Proceedings of the 2nd Asia-Pacific Workshop on Networking*. 2018, pp. 8–14.

[58] Jaichen Xue et al. "Fast congestion control in RDMA-based datacenter networks". In: *Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos*. 2018, pp. 24–26.

[61] Yiwen Zhang et al. "Performance isolation anomalies in RDMA". In: *Proceedings of the Workshop on Kernel-Bypass Networks*. 2017, pp. 43–48.

[63] Yibo Zhu et al. “Congestion control for large-scale RDMA deployments”. In: *ACM SIGCOMM Computer Communication Review* 45.4 (2015), pp. 523–536.

High Performance Networking

[17] Nathan Hanford, Brian Tierney, and Dipak Ghosal. “Optimizing data transfer nodes using packet pacing”. In: *Proceedings of the Second Workshop on Innovating the Network for Data-Intensive Science*. 2015, pp. 1–8.

[18] Nathan Hanford et al. “Analysis of the effect of core affinity on high-throughput flows”. In: *2014 Fourth International Workshop on Network-Aware Data Management*. IEEE. 2014, pp. 9–15.

[19] Nathan Hanford et al. “Improving network performance on multicore systems: Impact of core affinities on high throughput flows”. In: *Future Generation Computer Systems* 56 (2016), pp. 277–283.

[26] Mariam Kiran et al. “Enabling intent to configure scientific networks for high performance demands”. In: *Future Generation Computer Systems* 79 (2018), pp. 205–214.

[27] Ezra Kissel et al. “Efficient wide area data transfer protocols for 100 Gbps networks and beyond”. In: *Proceedings of the Third International Workshop on Network-Aware Data Management*. 2013, pp. 1–10.

[33] Marek Majkowski. *How to achieve low latency with 10Gbps Ethernet*. Cloudflare. 2015. URL: https://blog.cloudflare.com/how-to-achieve-low-latency/.

Kernel Bypass Networking

[22] Jaehyun Hwang et al. "TCP≈RDMA: CPU-efficient Remote Storage Access with i10". In: *17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)*. 2020, pp. 127–140.

[34] Michael Marty et al. "Snap: a microkernel approach to host networking". In: *Proceedings of the 27th ACM Symposium on Operating Systems Principles*. 2019, pp. 399–413.

Documentation

[1] *200Gb/s ConnectX-6 Ethernet Single/Dual-Port Adapter IC*. NVIDIA. URL: https://www.mellanox.com/products/ethernet-adapter-ic/connectx-6-en-ic.

[8] *cgroups(7) — Linux manual page*. Mar. 2021. URL: https://man7.org/linux/man-pages/man8/rdma-system.8.html.

[10] Docker. *Docker Documentation*. URL: https://docs.docker.com.

[11] Docker. *Use host networking*. URL: https://docs.docker.com/network/host/.

[13] The Linux Foundation. *Kubernetes Documentation*. Version v1.19. URL: https://kubernetes.io/docs/home/.

[23] *Intel® Ethernet 800 Network Adapters (up to 100GbE)*. Intel. 2021. URL: https://www.intel.com/content/www/us/en/products/details/ethernet/800-network-adapters.html.

[36] NVIDIA. *NVIDIA MLNX_OFED Documentation*. Version 5.1-0.6.6.0. Aug. 5, 2020. URL: https://docs.mellanox.com/x/5pXuAQ.

[37] NVIDIA. *RDMA Aware Networks Programming User Manual*. Version v1.7. URL: https://docs.mellanox.com/pages/viewpage.action?pageId=34256560.

[44] *RDMA Controller*. May 2021. URL: https://www.kernel.org/doc/Documentation/cgroup-v1/rdma.txt.

[45] *RDMA Core Documentation*. Version v31.1. Nov. 3, 2020. URL: https://github.com/linux-rdma/rdma-core/tree/stable-v31/Documentation.

Tutorials

[38] Itay Ozery. *Accelerating Bare Metal Kubernetes Workloads, the Right Way*. Mellanox Technologies. Nov. 10, 2019. URL: https://blog.mellanox.com/2019/11/accelerating-bare-metal-kubernetes-workloads-the-right-way/.

[48] Mellanox Technologies. *Docker RDMA SRIOV Networking with ConnectX4 / ConnectX5 / ConnectX6*. July 30, 2020. URL: https://community.mellanox.com/s/article/Docker-RDMA-SRIOV-Networking-with-ConnectX4-ConnectX5-ConnectX6.

[49] Mellanox Technologies. *Docker RoCE MACVLAN Networking with ConnectX4 / ConnectX5*. June 17, 2020. URL: https://community.mellanox.com/s/article/docker-roce-macvlan-networking-with-connectx4-connectx5.

[50] Mellanox Technologies. *How-to: Deploy RDMA accelerated Docker container over InfiniBand fabric*. June 30, 2019. URL: https://docs.mellanox.com/pages/releaseview.action?pageId=15049785.

[51] Mellanox Technologies. *How-to: Deploy RoCE accelerated Docker container*. July 30, 2019. URL: https://docs.mellanox.com/pages/releaseview.action?pageId=15049758.

[52] Mellanox Technologies. *HowTo Create Docker Container Enabled with RoCE*. Mar. 1, 2020. URL: https://community.mellanox.com/s/article/howto-create-docker-container-enabled-with-roce.

[53] Mellanox Technologies. *RDG: RoCE accelerated K8s cluster deployment for ML and HPC workloads*. June 30, 2019. URL: https://docs.mellanox.com/pages/releaseview.action?pageId=15049828.

Mellanox Infosheets

[15] Dror Goldenberg and Parav Pandit. *Mellanox Container Journey*. Mellanox Technologies, June 2019. URL: http://qnib.org/data/hpcw19/7_END_2_MellanoxJourney.pdf.

[54] Mellanox Technologies. *RDMA and RoCE for Ethernet Network Efficiency Performance*. URL: https://www.mellanox.com/products/adapter-ethernet-SW/RDMA-RoCE-Ethernet-Network-Efficiency.

Appendix

A RDMA Two-Sided Ops Diagram