Exploring Secure Inter-Container Communication on Different Hosts

Varsha Natarajan
University of Colorado Boulder
Boulder, Colorado, USA
vana7343@colorado.edu

Adarsh Srinivasan
University of Colorado Boulder
Boulder, Colorado, USA
adsr8793@colorado.edu

Kireeti Mucherla
University of Colorado Boulder
Boulder, Colorado, USA
kireeti.mucherla@colorado.edu

ACM Reference Format:
Varsha Natarajan, Adarsh Srinivasan, and Kireeti Mucherla. 2023. Exploring Secure Inter-Container Communication on Different Hosts.

1 ABSTRACT

Containerization technology, epitomized by Docker and Kubernetes, has revolutionized the world of cloud computing by providing a lightweight and scalable solution for deploying applications. However, efficient inter-container communication across hosts remains a critical challenge. This paper introduces an innovative approach to tackle this problem by combining Remote Direct Memory Access (RDMA) and Redis, a high-performance, in-memory data store. Our solution leverages the low-latency, high-throughput capabilities of RDMA to enable direct memory-to-memory communication between containers, while Redis serves as a distributed, fault-tolerant data structure server to manage communication and coordination. We present a comprehensive analysis of the architecture, design considerations, and performance results, demonstrating significant inter-container communication speed and reliability improvements. This research contributes to the ongoing efforts to enhance containerized applications' orchestration and communication capabilities in multi-host environments, opening new possibilities for high-performance, microservices-based applications in cloud computing.

2 INTRODUCTION

The motivation behind this project stems from the increasing demand for high-speed, low-latency, and efficient communication between distributed Docker containers, particularly when they are running on separate hosts. In today's technology landscape, containerization has become a cornerstone of application deployment and scaling. However, traditional methods of inter-container communication, such as HTTPS/HTTP, gRPC, and messaging queues, while secure and reliable, may not always meet the requirements of real-time, low-latency communication.

The need for low-latency communication becomes particularly critical in various use cases, such as real-time data processing, distributed computing, and microservices architectures, where delays in data transfer can lead to inefficiencies and bottlenecks. This project aims to address this challenge and explore innovative solutions for improving communication efficiency between containers in different hosts.

The project also recognizes that shared memory can be a powerful resource for achieving low-latency communication. Containers running on the same host can efficiently use shared memory for synchronized communication. However, extending this concept to containers on different hosts is a challenge that requires innovative solutions. The idea of using shared memory over RDMA for inter-container communication across distributed hosts is a unique and promising approach.

The motivation behind this project is not only to experiment with cutting-edge technologies but also to address a significant challenge in containerized environments. The project's outcomes can have a wide range of applications, from optimizing data-intensive workloads to enabling real-time data processing and improving the overall efficiency of distributed systems. Ultimately, the goal is to contribute to the advancement of containerization technology and empower businesses and developers to harness the full potential of their distributed containerized applications.

3 LITERATURE SURVEY

To prepare for this project, we consulted three papers, and for implementation we followed one blog. In this section, we have provided a summary of each resource we used.

The paper titled "FreeFlow: Software-based Virtual RDMA Networking for Containerized Clouds" [3] explains about the FreeFlow, which is a software-based virtual RDMA networking framework designed to bridge the gap between containerization and RDMA technologies in cloud-based applications. Its primary goal is to enable containerized applications running in shared cloud environments to harness the high-performance benefits of RDMA, while simultaneously ensuring the critical properties of isolation, portability, and controllability that are essential for efficient container management. Unlike existing solutions, FreeFlow does not require specialized hardware or hardware-based I/O virtualization. It achieves this by employing a software virtual switch on each server, offering transparent integration with applications running inside containers. This innovative approach eliminates performance bottlenecks, allowing FreeFlow to deliver throughput and latency comparable to bare-metal RDMA while significantly enhancing the performance of data-intensive applications like Spark and TensorFlow. FreeFlow represents a promising solution for optimizing the performance, cost efficiency, and resource control in cloud-based containerized applications

The paper titled "MasQ: RDMA for Virtual Private Cloud" [5] In this paper, the authors address the challenge of enabling Remote Direct Memory Access (RDMA) in Virtual Private Clouds (VPCs) within public cloud environments, focusing on RoCEv2 networks. They propose MasQ, a novel RDMA virtualization approach that

combines software-defined rules for communication with hardware execution to achieve both high performance and scalability. MasQ introduces techniques such as vBond and RConntrack for abstracting virtual RoCE networks, providing virtual IP communication between RDMA applications, and supporting security measures like security groups and firewall as a service. The solution operates primarily in the control path, resulting in negligible performance overhead. Their prototype implementation demonstrates the feasibility of deploying RDMA in virtualized data centers, making it a significant contribution to the field of RDMA network virtualization for VPCs.

The paper "Using One-Sided RDMA Reads to Builda Fast, CPU-Efficient Key-Value Store" [2] explores the design of Pilaf, a distributed in-memory key-value store that takes advantage of Remote Direct Memory Access (RDMA) to achieve high performance with minimal CPU overhead. Pilaf leverages RDMA for read-only service requests, such as "gets," while traditional messaging handles other types of requests. This approach simplifies the design and restricts the class of memory access races that can occur, mainly concerning read operations. To address read-write races between clients and the server, Pilaf introduces self-verifying data structures, which include checksummed root data objects and pointers with checksums. Clients can bypass the server's CPU for processing read requests, achieving optimal CPU overhead. Pilaf demonstrates its high performance and low CPU utilization in experiments on Infiniband-equipped machines, outperforming traditional key-value stores like Memcached and Redis, making it a significant contribution to the utilization of RDMA in datacenter-scale systems infrastructure.

In paper "High Performance Design for Redis with Fast Event-Driven RDMA RPCs"[4], the authors address the limitations of existing key-value stores, such as Redis, which use conventional socket I/O interfaces, resulting in significant CPU copy overhead and performance constraints, especially when used with low-speed Network Interface Cards (NICs). To overcome these challenges, the paper introduces FeRR, a fast event-driven RDMA RPC framework that prioritizes low latency and high throughput. FeRR considers the network primitives of RDMA hardware and offers an efficient event-driven request notification mechanism, making it suitable for integration into existing systems. Furthermore, the authors implement a novel branch of Redis called FeRR-driven Redis to demonstrate the framework's performance enhancements. To resolve additional challenges arising from this integration, the paper addresses issues related to Redis Serialization Protocol (RESP) and the single-threaded framework of Redis. Their proposed solutions, including an optimized serialization protocol and a parallel task engine based on cuckoo hashing with optimistic locking, are shown to significantly boost Redis's performance. Through experiments, FeRR-driven Redis achieves remarkable throughput and low latency, making it a promising solution for high-performance key-value stores.

4 ARCHITECTURE

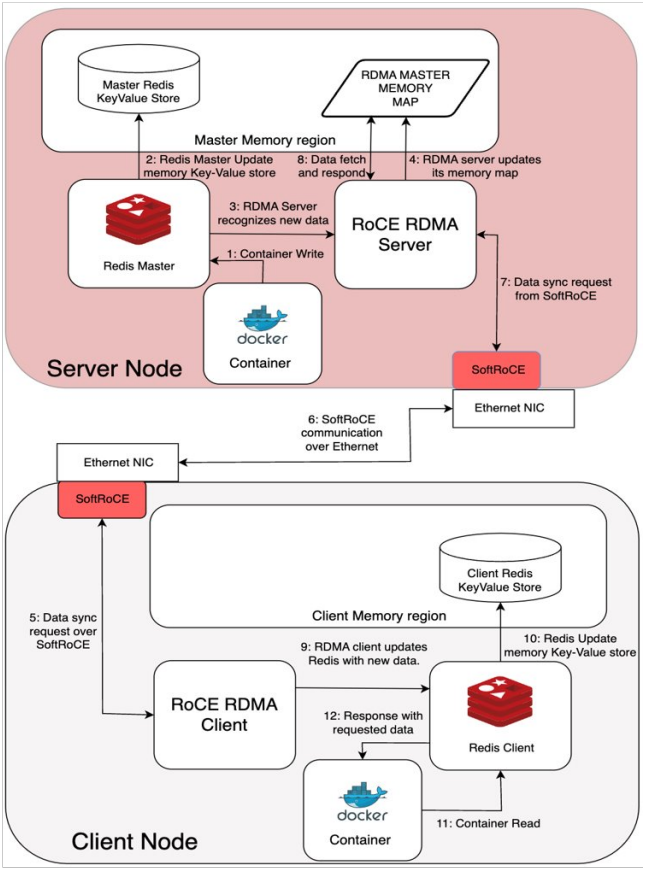

Figure 1: Proposed Architecture Model

Our architecture comprises two nodes: a master and a client node, each hosting a master and client container, respectively. The primary objective is to establish data synchronization over RDMA between these containers, ensuring that when the master container writes data, the client container can read it with minimal latency. The data synchronization process is as follows:

Step 1: Container on Server writes a message on Redis running on the server.
Step 2: The Redis instance on server updates its in-memory Key-Value store.
Step 3: RDMA-Agent on Server recognizes the change.
Step 4: RDMA-Agent syncs its Memory Map with the updated data.
Step 5: RDMA-Agent on the Client node sends a sync request.
Step 6: The sync request is forwarded over RoCE connection.
Step 7: RDMA-Agent on the Server Node receives the request.
Step 8: RDMA-Agent on the Server Node replies back with the Memory data.
Step 9: RDMA-Agent on the Client Node updates Redis.
Step 10: Redis on the Client Node updates its Key-Value Store with

the latest data.
Step 11: Container on Client Node requests for a Key from Redis.
Step 12: Container on the client node receives response for the requested Key.

5 PROPOSED WORK

The project focuses on creating a high-performance communication framework for inter-container communication across hosts in cloud computing environments. The framework will use technologies such as Remote Direct Memory Access (RDMA), Redis, Virtual machines, SoftROCE, Docker, and Socket Programming. Our project has three main tasks: the first is setting up RDMA connections between nodes using SoftROCE, the next task is to establish an RDMA-accelerated communication between Redis Master and Redis Worker, and finally, enabling real-time communication between containers across hosts. These tasks aim to mitigate the challenges of latency and improve reliability in microservices orchestration.

Task 1: Establish RDMA Connection Between Nodes In the first phase of the project, we aim to establish a reliable RDMA connection between two nodes. For this purpose, we will employ two virtual machines (VMs) running with Ubuntu 22.0 equipped with SoftiROCE, a tool that emulates a Remote Network Interface Card (RNIC) within VMs. This setup will enable us to simulate RDMA communication between the VM nodes. Our goal is to evaluate the feasibility and efficiency of RDMA for inter-node communication, with a focus on latency and data throughput.

Task 2: Establish a Connection Between Redis Master on Node 1 and Redis Worker on Node 2 Using RDMA
We plan to integrate Redis, a high-performance, in-memory data store, into the communication framework. To achieve this, we will refer to the documentation provided by the open-source project RdmaAcceleratingRedis [1], which explores the acceleration of Redis using RDMA. We will also draw insights from the paper titled "High Performance Design for Redis with Fast Event-Driven RDMA RPCs"[4] to improve the approach. The objective is to establish a seamless RDMA connection between Redis Master on Node 1 and Redis Worker on Node 2, enabling high-speed, low-latency data transfers. We will conduct experiments to verify the reliability and performance of this connection, focusing on the synchronization of the Redis memory table from the Master to the Worker node.

Subtask 2A: RDMA connection between Node 1 and Node 2
Each node will have one or more RDMA enabled NIC devices. Each RDMA device is like a PCI device. RDMA communication between two devices happens based on three queues: send queue, receive queue, and completion queue. Send queue and Receive queue will always be created as a Queue Pair. Applications enqueues a work request into the send and receive queues, send queue acts as a pointer for message buffer and receive queue acts as a pointer in which incoming message should be stored. Once the work request is completed, it gets enqueued into the completion queue. These queue pairs (QP), except for the completion queues, can be registered in a protection domain (PD). In RDMA read and write operations, only the sender is actively involved. The receiver remains passive, requiring no operation, CPU cycles, or even awareness of the data transfer. To initiate a read or write, the sender needs two pieces of information from the receiver: the remote memory address and a unique memory registration key. These details are usually acquired beforehand through traditional RDMA send/receive mechanisms. Similarly, memory regions that are registered to work with the RDMA will also be registered in a protection domain. These protection domains are a means of grouping and protecting resources from arbitrary accesses from the remote. This limits which memory regions can be accessed by which queue pairs (QP), providing a degree of protection from unauthorized access. We are using the library libibverbs, developed and maintained by Roland Dreier since 2006, that enables programs to use RDMA "verbs" for direct access to RDMA hardware from userspace.

Subtask 2B: Redis connection between Node 1 and Node 2
Traditionally, Redis Master-Slave architecture uses the asynchronous method for transferring the data from master to slave. The slave/replica node issues a SLAVEOF command to become a replica of the master node. During data synchronization, the master node dumps the data from the memory to the disk, and then starts a thread to send the data in the disk to the other replicas. This disk Read/Write operation becomes a really big network performance overhead and since the master is responsible for sending the data to the clients, the large amount of data transfer also incurs high overload to network and CPU of master. In order to reduce this overload, we plan to use the RDMA memory map for synchronization. The Redis master dumps the data to the RDMA memory map, and the Redis slave can use *RDMA read* to read the data from the master's memory map in synchronization. This eliminates the network load and enhances the overall performance.

Task 3: Communication Between Containers Across Hosts
In the final phase of the project, we will extend the benefits of RDMA-accelerated Redis communication to containerized applications spanning multiple hosts. We will set up a container on Host 1 with a link to the Redis Master, and a container on Host 2 with a link to the Redis Worker. The ultimate goal of this task is to achieve seamless, real-time communication between these containers. Specifically, when the container on host 1 writes a message to Redis Master, the container on Host 2 should promptly and securely be able to read the message. This experiment will validate the practicality of the approach in enhancing inter-container communication across hosts, showcasing the potential for high-performance, microservices-based applications in cloud computing environments.

6 SECURITY

Security stands as a paramount consideration for any application, and our system addresses this concern with a comprehensive approach. Notably, we have identified two potential avenues through which malicious entities could compromise the integrity of our system: Our architectural design addresses both these challenges by harnessing the security features provided by RDMA and Redis, thereby establishing a robust defense against potential threats.

1. Unauthorized Write Access to Redis Data Store:A malicious system may attempt to tamper with our Redis data store by initiating unauthorized write operations.
Prevention stratergy: Our security strategy in Redis hinges on the deployment of a unidirectional SSL secured connection. This proactive measure serves as a robust deterrent against unauthorized hosts attempting to write to our Redis data store. To elaborate, any application seeking to write to Redis must undergo a stringent certificate validation process with the server.
By enforcing this one-way SSL secured connection, we ensure that only recognized and authenticated hosts with valid certificates are granted the privilege to interact with our Redis data store. Consequently, this security mechanism serves as an effective barrier, mitigating our primary security concern related to unauthorized write access. The stringent validation process eliminates the first risk explained above.

2. Manipulation of Data in RDMA Memory Regions by Malicious Users: Another potential security concern involves malicious users attempting to manipulate data within RDMA memory regions.
Prevention Stratergy: RDMA, a high-performance networking technology, relies on a unique memory management strategy called memory registration. This process, orchestrated by the operating system's internal driver, defines specific properties for a designated memory region. These properties include:

- **Protection:** Safeguarding the memory from unauthorized access.
- **Byte-level range:** Defining the precise boundaries of the memory region.
- **Permissions:** Specifying Read/Write access for local and remote entities.
- **Pinning:** Ensuring the memory remains resident in RAM, preventing swapping and guaranteeing low latency.

Once registered, this memory becomes readily accessible for RDMA operations. However, for additional security, these regions are further fortified by protection domains. Imagine these as virtual walls around each memory region, controlling access and preventing unauthorized entry. Thus protection domains act as a robust barrier, safeguarding the confidentiality and integrity of data within the RDMA framework. By implementing this security measure, we successfully mitigate our second security concern.

7 EVALUATION

In this section, we will delve into the performance differences between various data transfer protocols. We will compare and analyze key aspects such as latency, bandwidth, and data size for TCP, UDP, and RDMA. This analysis will serve to highlight the significant speed advantage offered by RDMA over traditional TCP/UDP protocols.
The second part of this section delves deeper into the performance benefits of RDMA in comparison to other protocols. We present empirical results showcasing the contrasting speeds achieved by Redis when utilizing RDMA versus traditional RDMA and other protocols like HTTP and RPC. This analysis provides concrete evidence of RDMA's superior performance for key operations.

In our initial phase of evaluation, we plotted various graphs comparing different aspects of network to understand the performances of TCP, UDP and Remote Direct Memory Access (RDMA).

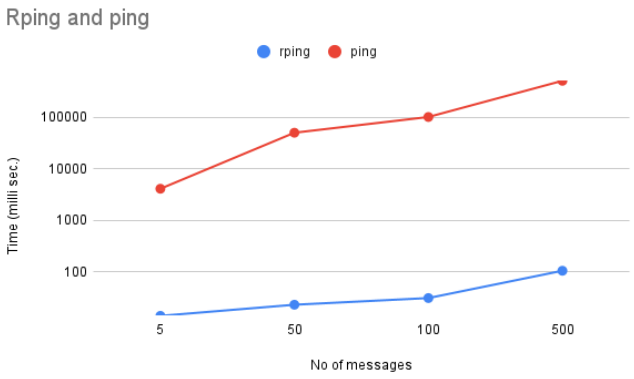

Figure 2: Graph plot between ping and Rping RTT's

In further analysis, we observed a notable distinction in the behavior of regular ping packets, which adhere to the traditional routing methods, compared to the RDMA-based communications. As the count of ping packets increased, the Round-Trip Time (RTT) exhibited a linear growth for regular ping packets. In contrast, the RTT for RDMA remained remarkably constant, showcasing the inherent efficiency of Remote Direct Memory Access in maintaining consistent low-latency performance even under varying loads.

To delve deeper into these experiments, we directed our attention toward comparing latencies and bandwidths.
We performed bandwidth test using qperf and id_read_bw tools to evaluate TCP, UDP, and RDMA-ROCE performances.

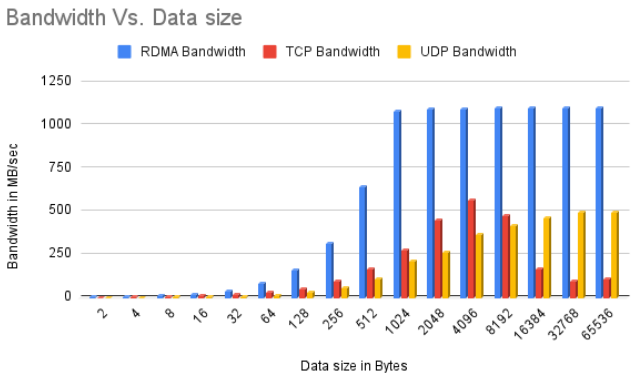

Figure 3: Graph plot between bandwidth and Data size

If we observe the graph Fig. 3, we can say that RDMA out performed

both TCP and UDP. As data size increases, there is an improvement in the bandwidth of TCP and UDP but is no where comparable to RDMA.
In the next step, we performed the latency test among TCP, UDP and RDMA to understand the RDMA's latency performance.

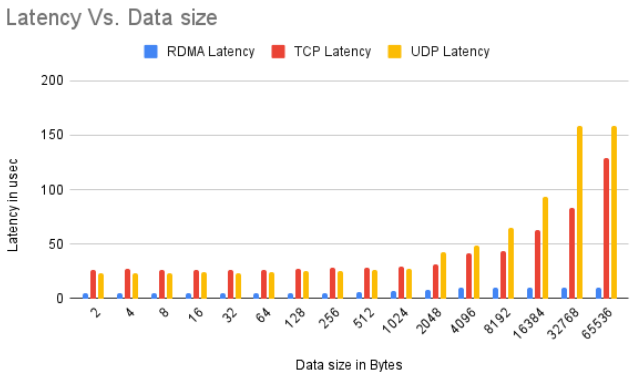

Figure 4: Graph plot between Latency and Data size

It can be clearly observed from Fig. 4 that RDMA's latency is much much lower compared to TCP and UDP. As, the data size increases, there is a sharp increase of latency in both TCP and UDP but, the latency of RDMA is almost constant.
Having established RDMA's superiority over TCP and UDP, next section is dedicated to compare RDMA-REDIS, REDIS, HTTP, and RPC. To comprehensively assess their performance, extensive tests are conducted with varying data sizes and sending a different number of packets in each iteration. This meticulous analysis will reveal the nuances of each protocol's performance under various conditions, providing valuable insights into their strengths and weaknesses.

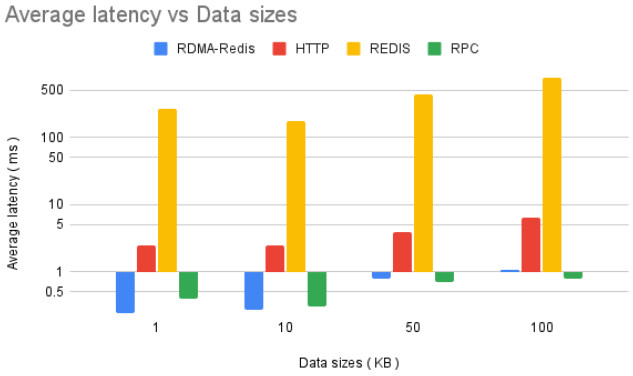

Figure 5: Average latency comparing RDMA-Redis, Redis, HTTP and RPC, with data sizes (1KB, 10KB, 50KB, 100KB)

	RDMA-REDIS	HTTP	REDIS	RPC
1	sum: 0.237 count: 1 avg: 0.237	sum: 25 count: 10 avg: 2.5	sum: 2642 count: 10 avg: 264.2	sum: 4 count: 10 avg: 0.4
10	sum: 0.274 count: 1 avg: 0.274	sum: 25 count: 10 avg: 2.5	sum: 1763 count: 10 avg: 176.3	sum: 3 count: 10 avg: 0.3
50	sum: 0.79 count: 1 avg: 0.79	sum: 39 count: 10 avg: 3.9	sum: 4345 count: 10 avg: 434.5	sum: 7 count: 10 avg: 0.7
100	sum: 1.053 count: 1 avg: 1.053	sum: 63 count: 10 avg: 6.3	sum: 7643 count: 10 avg: 764.3	sum: 8 count: 10 avg: 0.8

Figure 6: Average latency comparing RDMA-Redis, Redis, HTTP and RPC, with data sizes (1KB, 10KB, 50KB, 100KB)

Figure 5,6 shows the experiment results and it vividly demonstrates the superior performance of RDMA-REDIS compared to traditional Redis. As data size increases, this advantage becomes even more pronounced. Furthermore, RDMA-REDIS surpasses the performance of HTTP across all data sizes. While ideally RDMA should outperform RPC as well, the utilization of ROCE (Virtualized RDMA) in this experiment limited its ability to do so for larger data sizes.

REFERENCES

[1] Zzaoen aoen. 2018. Rdma Accelerating Redis. https://github.com/zzaoen/RdmaAcceleratingRedis.
[2] Jinyang Li Christopher Mitchell, Yifeng Geng. 2013. Using one-sided RDMA reads to build a fast, CPU-efficient key-value store. *USENIX conference on Annual Technical Conference* 24, 3 (2013), 103–114. https://doi.org/10.5555/2535461.2535475
[3] Hongqiang Harry Liu3 Yibo Zhu2 Jitu Padhye2 Shachar Raindel2 Chuanxiong Guo4 Vyas Sekar1 Srinivasan Seshan Daehyeok Kim1, Tianlong Yu1. 2019. FreeFlow: Software-based Virtual RDMA Networking for Containerized Clouds. 24, 3 (2019).
[4] Xing Wei Chengcheng Huang Xuan Zhou Aoying Zhou Xuecheng Qi, Huiqi Hu. 2020. High Performance Design for Redis with Fast Event-Driven RDMA RPCs. 24, 3 (2020). https://doi.org/10.1007/978-3-030-59410-7_12
[5] Binzhang Fu Kun Tan Bei Hua Zhi-Li Zhang Kai Zheng Zhiqiang He, Dongyang Wang. 2020. RDMA for Virtual Private Cloud. 24, 3 (2020), 1–14. https://doi.org/10.1145/3387514.3405849