CoRD: Converged RDMA Dataplane

Maksym Planeta
Exostellar / Barkhausen Institut*

Jan Bierbaum
TU Dresden

Michael Roitzsch
Barkhausen Institut

Hermann Härtig
TU Dresden

***Abstract*—HPC networking is often characterized by kernel bypass, which is considered mandatory for large parallel and distributed applications. However, kernel bypass comes at a price because it breaks the traditional OS architecture, requiring applications to use special APIs and limiting the OS's control over existing network connections. We make the case that kernel bypass is *not* mandatory. Rather, high-performance networking relies on multiple performance-improving techniques, with kernel bypass even being detrimental to performance under specific conditions. CoRD removes kernel bypass from RDMA networks, primarily to enable efficient OS-level control over the RDMA dataplane. This control can be used to enhance security or resource allocation policies, and, as we demonstrate in one of the use cases, can improve end-to-end application performance by up to 10%. This architecture can enable Cloud-based distributed RDMA applications and facilitate deployment of coupled HPC applications.**

***Index Terms*—RDMA, kernel bypass, high-performance networking, cloud computing, operating systems**

I. Introduction

Cloud architectures are increasingly geared towards handling large distributed workloads [1], [2], [3]. However, in a typical Cloud deployment, only small partitions offer high-performance networking [4], [5], [6], limiting where high-performance applications could run. For such partitioning to disappear, traditional and high-performance networks must converge into a unified architecture. In this paper, we demonstrate how operating systems (OSes) can aid in this integration by making high-performance networks more accessible. We believe this approach can significantly improve performance within Cloud environments and interoperability in the HPC environment.

The main reason for the difference between traditional and high-performance networks lies in the interfaces the OS provides. A traditional OS supplies a "rich and robust" [7, p. 228] abstraction layer between the applications and the underlying hardware and networking to provide application portability, implement resource scheduling, and enforce security policies. High-performance networks, often synonymous to Remote Direct Memory Access (RDMA) networks [8], [9] or *dataplane* architectures [10], [11], subvert this layered OS architecture by granting user-level applications direct access to network devices (NICs). These applications bypass OS services (including the network stack), employing their own NIC drivers and managing most of the scheduling and resource allocation themselves [11], [12]. This approach compels the OS to depend on the NIC to implement security (IOMMUs [13], [14] or VLANs [15] are insufficient) and resource-sharing policies [16], [17], [18]. As a result, the OS becomes nothing more than an over-engineered bootloader [19], [20], [21], [22].

In response to the demand for flexibility and OS control over high-performance network communication, the OS community has proposed several architectures [23], [24], [25], [26], [27], which enable fine-grained OS control over high-performance communication. These novel architectures, however, are incompatible with both the traditional socket API [28] and with widespread RDMA APIs, such as *ibverbs* [29]. And even when such compatibility exists [17], [18], [30], [31], there is a need for a translator service, which, while maintaining backward compatibility, takes precious resources away from the applications.

One of the fundamental assumptions behind kernel-bypass architectures is a popular belief that system calls are performance killers for high-performance systems [32], [33], [34], [35]. We challenge this prevalent view and argue that kernel-bypass is not indispensable for high-performance Cloud applications. Inspired by these insights, we propose CoRD, a strategically modified version of the traditional RDMA architecture that routes the dataplane (i.e., send and receive operations) through the OS kernel. This critical shift gives the kernel full control over RDMA communication, enabling it to enforce security policies [36], [37], provide virtualization [18], [38], manage resources at a fine-granular level [17], [18], and enhance application observability [39].

Unlike previous approaches [23], [24], [25], [26], [27], we prioritize backward compatibility at the API and binary levels, allowing existing network-sensitive applications to run on CoRD without modifications. This property can simplify the coupling of HPC applications [40], [41] by offloading coordination and scheduling of shared resource access to the OS. In a Cloud environment, where resource-sharing is omnipresent, CoRD can facilitate the deployment of RDMA networks by enabling RDMA-based microservices [42].

Our evaluation shows that CoRD is capable of maintaining performance for large-scale applications while enabling flexible OS-level control over the communication dataplane. This enhanced control can improve security or resource allocation policies and, as we demonstrate in one use case, improve end-to-end application performance by up to $10\,\%$. Although we observed a case of kernel bypass being significantly faster than CoRD (up to $24\,\%$), the respective application also had other requirements for its environment, like a noiseless network. Such conditions are hard to meet in current shared cloud

* The work was done while the author was at Barkhausen Institut.

environments and may be hard to guarantee in future high-performance Clouds. Therefore, Cloud applications will still be better off if CoRD schedules and manages their resources in a traditional way.

The main contributions of this paper are: 1) Design and implementation of an OS-controlled RDMA dataplane (Sections III and IV), 2) Analysis and evaluation of the kernel-bypass role for high-performance applications (Section V) and 3) Demonstration of use-case scenarios for RDMA dataplane interception (Section VI).

II. RDMA Performance

Compared to traditional socket-based networks, RDMA networks rely on three properties for high-/performance communication: *fast dataplane*, *operation offloading*, and *noiseless environment*. Each property consists of a set of techniques, which come with their own benefits and limitations and can be employed independently of the rest. These techniques can be combined in various ways, depending on the use case.

Enabling these techniques in RDMA networks requires a special API [8], [24], [29] that is very different from the traditional socket-based API. The opposite is also true: not using a performance-improving technique can make the programming model simpler but also sacrifices performance [43].

A. Fast Dataplane

RDMA networks put special attention towards optimizing dataplane operations (e.g. send and receive). One specific area for optimization is processing latency, i.e. the delay between the application issuing a message and the NIC transmitting it, and the time between the NIC receiving a packet and the application being able to process it. Compared to traditional networks, the key techniques for a fast dataplane are *kernel-bypass*, *zero-copy*, and *polling*.

Kernel-bypass removes the OS kernel as an intermediary between applications and the NIC. The kernel maps NIC MMIO registers to an application, which can then access the device directly. This technique avoids the overhead of system calls, context switches, and the kernel-level software stack. In return, the application must come with its own user-level NIC drivers and rely on the NIC for security and resource management.

Traditional `read` and `write` system calls copy the content of messages between kernel and user memory. *Zero-copy* avoids this overhead by enabling the NIC to access the application memory directly. This technique is particularly crucial for large messages, where the overhead of copying can be significant. For zero-copy to work, the application memory needs to be *pinned* to prevent the OS from moving or swapping out this memory while the NIC can access it. Moreover, either the NIC [29] or the application [11] needs to translate applications' virtual addresses to physical addresses.

Finally, with *polling*, the application continuously checks the NIC's message queues to avoid the overhead of interrupt processing. If the application requested an interrupt, the OS would receive the interrupt and only then notify the application about the message. The required context and privilege-level switches add extra overhead. In other words, polling is a way to bypass the OS kernel on the receiver's side. Unfortunately, polling is also very resource-intensive, as the application never gives up the CPU, even without useful work to do.

B. Operation Offloading

RDMA networks strive to relieve the host CPU and offload work to the NIC whose specialized hardware can perform these operations faster and more efficiently. There exists a wide range of operation offloadings, but the two main ones are *network stack offloading* and one-sided *RDMA operations*. These operations are additionally beneficial because they allow the NIC to handle a network operation without *interrupting* the host CPU.

RDMA operations, like *RDMA read* or *RDMA write*, run directly on the NIC without involving the host CPU. For example, Alice wants to send messages to Bob using *send*/*receive* (two-sided) operations. Bob must ensure sufficient buffer space is available for incoming messages before Alice's messages arrive. For that, Bob's host CPU needs to regularly post *receive buffers* to the NIC. If Bob has other important work to do, such an obligation can interrupt his computation. With the RDMA *write* (one-sided) operation, Bob's host CPU is not involved at all, but his RDMA NIC writes the message directly to the designated memory area.

Network stack offloading moves the responsibility for processing network packets from the host CPU (either in the OS or the application) to the NIC. Traditionally, the OS kernel splits application-issued messages into packets and tracks each packet (e.g. for retransmission), whereas the NIC transfers the packets over the network. Even when available, offloading in traditional networks is relatively limited (e.g., checksums or segmentation offloading). RDMA networks, on the other hand, expose a message-level interface to the applications, allowing the application to pass a message descriptor to the NIC, which runs the packet-level protocol. In particular, RDMA NICs track the state of outstanding messages and, in case of a network error, retransmit any missing packets without involving the host.

C. Noiseless Environment

Considering that the network latency in RDMA networks can be as low as 1 µs [44], any disruption or delay in the host system or within the network can significantly impact the overall application performance. Driven by this observation, HPC systems minimize contention for any system resource. Any unexpected delays are considered *noise* and are avoided at all costs.

Measures to reduce noise work at both compute node and network levels. In compute nodes, the OS kernel is configured to suppress non-essential interrupts, and applications pin each thread to a dedicated CPU core to avert context switches. A fixed CPU frequency prevents performance fluctuations, and hyper-threading is disabled to eliminate contention for CPU resources. Non-essential services are disabled, and the system is configured to avoid swapping and paging [45].

RDMA networks rely on lossless flow control to prevent packet loss and associated retransmission delays [9], [46]. Lossless connection-level flow control is reflected in the user-level RDMA API by requiring the user to provide enough buffer space to accommodate all incoming messages in advance. Although a properly-configured RDMA network does not lose messages due to congestion, network congestion is still a problem [47]. Thus, in addition to flow control, RDMA networks also employ congestion control algorithms to improve network utilization [48]. Even then, Cloud HPC sites often prefer to use dedicated networks to avoid contention from other applications [4], [49].

D. RDMA in the Cloud

All the aforementioned techniques, when used together, can deliver unparalleled network performance. However, due to difficulties in deployability and manageability, RDMA networks often forgo some techniques to achieve higher flexibility and better resource utilization.

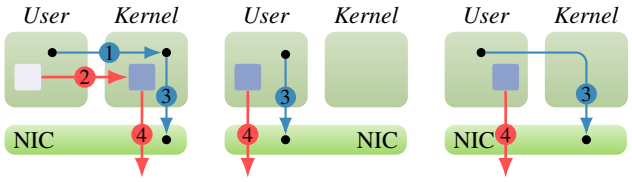
For example, Venkatesh et al. [50] forgo polling to save energy. KRCORE [26] and LITE [27] reintroduce the kernel into the dataplane to improve resource utilization. Our approach is similar to KRCORE and LITE, but we focus on keeping the interface backward compatible with existing RDMA networks. This allows us to evaluate the impact of removing kernel-bypass on unmodified large-scale real-world applications.

Operations offloading, too, is not always beneficial. First, relying on a particular implementation of a hardware feature creates vendor lock-in and has long been a concern in the TCP world [51]. Major cloud providers begin offering their own RDMA networks [5], [52], [53] with mutually incompatible offloading capabilities. These offloading capabilities are also incompatible to NVIDIA's NICs [54], [55], which are state-of-the-art in terms of performance. Second, even the standard RDMA read and write operations can be detrimental to application performance [56], [57], [58], because completing a single, complex request may require multiple RDMA operations.

Finally, a truely noiseless network environment is often not achievable. For example, physically separating RDMA networks segments from the rest of the network to avoid contention works only if the partitions are relatively small [4], [49]. RDMA networks can share the fabric with traditional networks, for example, by using the RoCE protocol [59], but then the lossless flow control is much less efficient [48]. In such environments, CoRD offers the OS an opportunity to coordinate resource allocation better than low-level hardware-centric approaches.

III. Converged RDMA Dataplane

Our proposed architecture, CoRD, integrates OS-level control into the high-performance user-level communication dataplane. This section first compares socket- and RDMA-based communication (Fig. 1). Then, it explains how CoRD modifies the RDMA dataplane to enable control over RDMA connections.



(a) Sockets (b) RDMA (c) CoRD

Figure 1: Socket, RDMA, and CoRD dataplane. To send a message over a socket, the application invokes the high-level kernel network stack 1. The kernel (a) or the application (b, c) request the NIC driver to send the packet 3, which copies the data from pinned memory ■ and sends it over the network 4.

In a traditional socket-based API, as shown in Fig. 1a, communication starts with the application making a system call 1 (e.g., `send`) instructing the kernel to dispatch a message over an existing connection (e.g., a TCP socket). Subsequently, the kernel copies 2 the message content into NIC-accessible (*pinned*) memory ■, splits the message into packets, and prepares corresponding packet metadata. Finally, the kernel *triggers* the NIC 3 to start the transmission of message packets 4. This entire operation is managed by a complex kernel-level network stack, designed to maintain scalability and improve resource utilization across a large number of user applications. However, this design philosophy inadvertently hampers network performance.

In contrast, RDMA networks (Fig. 1b) delegate a significant part of control to the application, allowing it to manage pinned memory ■ and access the NIC directly 3. The NIC retrieves the message content directly from user memory 4, thus bypassing the kernel. Note, that while this requires some network stack functionality at the application level (including a user-level NIC driver), a majority of responsibilities, such as managing concurrent users, are offloaded to the NIC. Despite the complexity added to the application, this architecture ultimately yields the lowest possible communication latency.

The existing RDMA architectures expect applications to access the NIC with the help of user-level RDMA drivers (Fig. 1b), which are dynamically linked to the application. There exist similar kernel-level RDMA drivers, which are mostly used for fast distributed storage [60], [61]. CoRD modifies the user-level RDMA architecture by prohibiting direct access to the NIC. Instead, CoRD provides its own user-level RDMA driver, which forwards data plane operations to the kernel-level RDMA drivers 3. From the application's point of view, the RDMA architecture remains the same (Fig. 1c).

Although CoRD forces each dataplane operation to go through the kernel, processing inside the kernel is kept to a minimum. In contrast to socket-based communication, CoRD only permits the enforcement of lightweight policies (e.g. for scheduling or security) to maintain its high network performance. This limitation is insignificant, however, as CoRD can afford to uphold simpler, more streamlined *CoRD policies* with a complexity similar to that of eBPF programs [62].

CoRD allows policies powerful enough to implement QoS, security, and isolation similarly to other dataplane interception techniques [17], [18]. We showcase some exemplary CoRD policies in Section VI. CoRD inevitably adds a constant per-message latency from user-kernel switching. The overhead from enforcing CoRD policies depends on the specifics of the implemented functionality.

IV. Implementation

The goal of our prototype implementation is to measure the cost of CoRD's architecture, which we expect to be higher than that of just a system call, but low enough to be tolerable for RDMA application. For that, we modify the user- and kernel-level `mlx5` device drivers for NVIDIA's ConnectX-series NICs. These drivers are used by the *ibverbs* library [29], which is the "narrow waist" of many high-performance user-level network stacks [17], which use it either directly [57], [63], [64] or through a higher-level API [65], [66], [67].

The ibverbs API defines *control- and dataplane* operations: Control-plane operations set up communication; they register device-accessible pinned memory, create communication endpoints (*queue pairs*), etc. Data-plane operations initiate message sending (`ibv_post_send`), post receive buffers (`ibv_post_recv`), or perform a non-blocking check if any of these operations have been completed (`ibv_poll_cq`). Control-plane operations require kernel support, whereas dataplane operations usually bypass the kernel. CoRD changes the user-level dataplane operations by funneling them through the kernel, so the OS can enforce policies.

To pass the dataplane operations through the kernel, we reuse the existing `ibverbs`-mechanism used by control-plane operations. This mechanism passes the operation parameters through the `ioctl` system call for processing by the kernel driver. To overcome Linux's limit on the amount of data that can be passed in a system call through the registers, the `ibverbs` library *serializes* and *deserializes* the arguments when invoking the kernel. These operations add to the system-call overhead but are not performance critical for control-plane operations. Importantly for CoRD, the `ioctl` code path allows the OS to intercept control-plane ibverbs calls and thus enforce security [36], [37], isolation [68], and resource management policies [69].

CoRD extends this existing `ioctl`-based interface to also pass arguments to dataplane operations. Now, to send or receive a message the application makes an `ioctl` system call, instead of calling into a user-level RDMA device driver. After entering the kernel, the dataplane operation reaches the kernel-level device driver, which normally provides high-performance networking within the kernel [60], [61], for further processing.

We had to make only small changes to the kernel-level control and data planes, because kernel- and user-level dataplanes have very similar implementations. The main difference comes from how the kernel-level driver allocates memory, so we enabled the kernel-level dataplane to work with user-level `ibverbs` objects and allowed message transfer of message buffers provided by user applications.

To simplify our prototype, we modified the implementation of the user-accessible ibverbs objects. As a result, a CoRD-enabled kernel can only create CoRD `ibverbs` objects, but it cannot create regular kernel-bypass objects. In future work, we plan to extend the CoRD kernel to support both CoRD and regular kernel-bypass objects.

Overall, we added or modified ~250 lines in the kernel-level driver and ~20 lines in the user-level driver. Despite kernel interposition, the ibverbs API remains unchanged and introduces no interrupts or asynchronous invocations on the data plane. In other words, without CoRD policies, the only overhead comes from crossing the user-kernel boundary and passing the operation parameters.

Our implementation depends on the NIC being able to access an application's virtual memory as applications pass message buffers to the NIC by virtual address. This feature is common for high-performance NICs [29], [70], which, in contrast to other approaches (e.g., `io_uring` [71]), relieves the kernel from virtual-to-device address translations on the critical path. If the application passes an invalid address, the NIC returns an error but does not access any memory that was not previously provided to the application by the kernel.

V. Performance Evaluation

Our goal is not only to simplify the usage of existing RDMA applications in shared environments but also to enable existing Cloud applications to adopt RDMA networks. However, in our evaluation, we only study the performance of existing RDMA microbenchmark and end-to-end applications after introducing CoRD. We do not evaluate how non-RDMA applications benefit from RDMA networks because we believe that the existing works [72], [73], [74] already make a convincing case for us. Therefore, we focus on CoRD's performance impact in adverse circumstances.

To demonstrate the flexibility of CoRD, we evaluate it with two systems. First, *Oracle*, comprising eight BM.Optimized3.36 nodes in the Oracle Cloud, each with two hyperthreading-enabled Intel 6354 18-core CPUs and $100\,\mathrm{Gbit/s}$ NVIDIA ConnectX-5 Ex RoCE NICs. The system runs vanilla Linux 6.2-rc7 with or without our patch to support CoRD in the `mlx5` driver. We disable Turbo Boost, pin all the benchmark processes to dedicated cores, and set the CPU power governor to the highest performance mode.

Our second target is the two-node *Azure* system deployed in the Azure Cloud. We use virtualized HB120 instances with two 64-core AMD EPYC 7V73X CPUs (only 120 cores passed to VM) and virtualized $200\,\mathrm{Gbit/s}$ NVIDIA ConnectX-6 InfiniBand NICs. In both systems KPTI [75], [76], an expensive kernel-level Meltdown attack [77] mitigation, is disabled by the kernel because modern CPUs do not need it. Specifically, because Azure's CPUs are not vulnerable to Meltdown. We run the benchmarks in the same way as on the Oracle system, except for not disabling dynamic frequency scaling due to the cloud provider policy.

Table I: Latency of point-to-point operations (expressed as baseline + CoRD's overhead) in µs on the Oracle system.

Size	RC			UD
	Read	Write	Send	Send
2^0	3.1 + 1.4	1.7 + 1.1	1.7 + 1.7	1.7 + 1.5
2^6	3.1 + 1.4	1.7 + 1.1	1.7 + 1.9	1.7 + 1.9
2^{12}	3.9 + 1.4	3.0 + 1.0	3.0 + 1.7	3.0 + 1.5
2^{15}	7.0 + 1.3	6.1 + 1.1	6.1 + 1.7	—
2^{18}	25.8 + 1.4	24.9 + 1.1	24.8 + 1.7	—
2^{20}	89.9 + 1.7	89.2 + 1.1	88.9 + 1.7	—

Table II: Throughput of point-to-point operations (expressed as baseline / CoRD) in Gbit/s on the Oracle system.

Size	RC			UD
	Read	Write	Send	Send
2^0	0.04 / 0.01	0.04 / 0.01	0.04 / 0.01	0.04 / 0.01
2^6	2.6 / 0.7	2.6 / 0.7	2.7 / 0.7	2.6 / 0.6
2^{12}	94 / 43	98 / 43	98 / 43	98 / 37
2^{15}	98 / 98	98 / 98	98 / 98	—
2^{18}	98 / 98	98 / 98	98 / 98	—
2^{20}	98 / 98	98 / 98	98 / 98	—

A. Microbenchmarks

First, we measure the overhead CoRD adds to the point-to-point latency, using the perftest 4.5 benchmark suite [78]. Out of several communication modes (*transports*) supported by RDMA networks, we chose the two most popular ones: Reliable Connection (RC) or Unreliable Datagram (UD). Like TCP, RC is reliable, ordered, and connection-based; whereas UD, akin to UDP, lacks ordering guarantees and retransmission service but does not necessitate connection establishment. Both RC and UD support two-sided Send/Receive communication, while only RC supports one-sided RDMA Read/Write and atomic operations. The message size ranges from 1 B to 1 MiB, except for UD, which supports only up to 4 KiB messages. The latency overhead is the difference between the baseline and CoRD-enabled communication.

Table I shows that CoRD adds a constant overhead to the baseline latency independent of the message size. The overhead is slightly different for different operations and ranges from 1.0 µs to 1.9 µs. In relative numbers, a send operation can experience as much as 100 % overhead when sending small messages and as little as 1 % for large messages.

The per-message overhead lowers the throughput, as measured by the perftest bandwidth benchmark (Table II). CoRD exhibits low performance because transferring numerous small messages is a CPU-bound task, and CoRD introduces additional overhead on the CPU side. As a result, CoRD achieves maximum throughput of 98 Gbit/s only with 32 KiB messages, whereas baseline RDMA communication reaches such performance already with 4 KiB messages.

For each message, CoRD adds overhead at the entry and exit points of every dataplane operation. Therefore, sender and receiver contribute to latency overhead independently. Figure 2 illustrates how CoRD contributes to the absolute

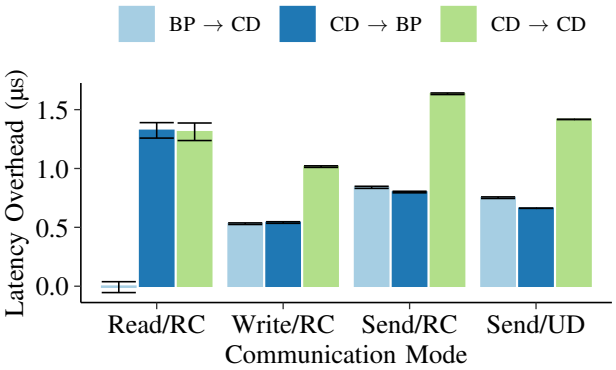


Figure 2: Latency overhead on Oracle when communicating over different transports (RC/UD) using one-sided (Read/Write) or two-sided (Send) communication. Client and server can independently run bypass (BP) or CoRD (CD), "→" indicates the direction of communication (from client to server).

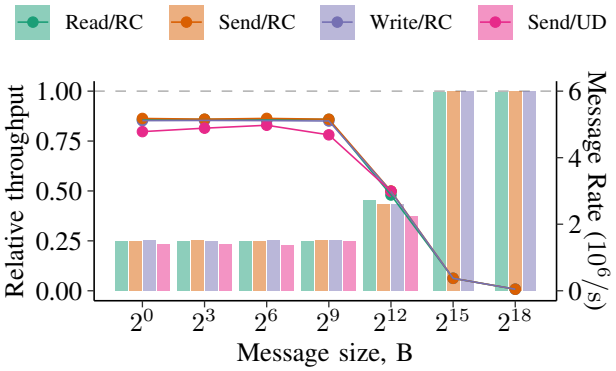


Figure 3: CoRD's throughput on Oracle relative to bypass communication. One-sided (Read/Write) or two-sided (Send) operations go over RC or UD. Overlayed lines show the message rate (right axis) in the bypass configuration.

latency overhead on each side compared to bypass-to-bypass communication when sending 4 KiB messages. The trend is similar for other message sizes.

When CoRD runs only on the sender, RDMA read has no overhead because the sender's CPU does not participate in the operation. In contrast, with RDMA write, perftest uses two writes to exchange data: one from the client to the server for synchronization and another for the server to fetch the data from the client. Except for RDMA read, enabling CoRD incurs equal overhead on each side.

Constant overhead per message significantly reduces maximum throughput for large bursts of small messages. Figure 3 corroborates this statement by showing CoRD throughput relative to baseline RDMA for different message sizes. On the other hand, with larger messages, bandwidth degradation becomes insignificant. This behaviour is similar for all types of communication (RC/UD, Send/Read/Write) as the per-message overhead is similar. Specifically, for 32 KiB messages exchanged using send operations, perftest measured ~370k messages per second and only 1% bandwidth degradation.

Table III: Latency (baseline + CoRD's overhead, µs) and throughput (baseline / CoRD, Gbit/s) for RC operations on the Azure system. The performance of Write is very similar to the of Read. Send over UD is very similar to Send over RC.

Size	Latency		Throughput	
	Read	Send	Read	Send
2^0	3.2 + 1.5	1.7 + 2.1	0.04 / 0.01	0.04 / 0.01
2^6	3.2 + 1.4	1.7 + 2.1	2.3 / 0.6	2.3 / 0.6
2^{12}	4.0 + 1.3	2.9 + 1.7	103 / 39	138 / 39
2^{14}	5.0 + 1.4	3.9 + 1.7	179 / 136	196 / 131
2^{16}	7.6 + 1.3	6.4 + 1.7	191 / 192	198 / 197
2^{18}	15.7 + 1.4	14.4 + 1.7	195 / 194	198 / 197
2^{20}	48.1 + 1.7	46.6 + 1.7	197 / 197	198 / 198

The Azure system exhibits similar behaviour (Table III), except for a few minor differences: The latency variation is higher as a direct result of enabled dynamic frequency scaling. The per-message overhead is slightly larger, especially for small messages, because the CoRD implementation we tested on Azure lacked support for *inline* messages[1], whereas baseline supported them. Moreover, for some message sizes (e.g. 4 KiB), CoRD shows a higher relative throughput on Oracle than on Azure because Azure's network offers double the throughput, making the constant per-message overhead more significant. Nevertheless, bandwidth overhead becomes negligible for large messages also on Azure (see Table III).

Overall, CoRD adds 1 µs to 1.7 µs of latency overhead depending on the operation type. The results for both of our systems (InfiniBand and RoCE) are similar and rather depend on the CPU's type and configuration, because CoRD adds work on the CPU side. In relative terms, the overhead varies widely from 1 % to 100 % depending on the message size, therefore it is only possible to judge the viability of CoRD for a specific application.

B. Collective Communication

In many cases, point-to-point operations serve as building blocks for more complex *collective* communication operations. A collective operation exchanges information among many processes of a distributed application, sometimes aggregating and processing the data during the operation. Theoretical LogP-based models [79], [80], [81] predict that CPU overhead accumulates proportionally to the number of point-to-point operations along the longest chain of hops in a particular operation. Therefore, it is essential to understand how CoRD performs with such operations.

To study the performance of collective operations, we measured the latency of MPI [67] collectives. MPI libraries offer dozens of highly optimized collective operations and are easy to scale for large numbers of communicating processes. We used the OSU Micro-Benchmarks [82] `7.1-1`, a popular MPI microbenchmark suite, together with OpenMPI [83] version `4.1.5rc4`, a popular open-source MPI library. Each run of the benchmark executed an MPI operation with a specific message size over 1000 iterations with 100 warm-up iterations. We report the average latency and standard deviation from 3000 iterations, measured across three distinct runs.

[1]Inline messages store the message content in the message descriptor instead of a registered memory region, thus reducing pointer chasing on the NIC side.

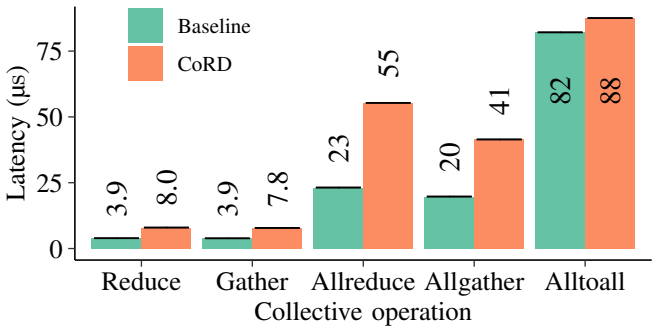


Figure 4: Latency of MPI collectives with 1-byte operands.

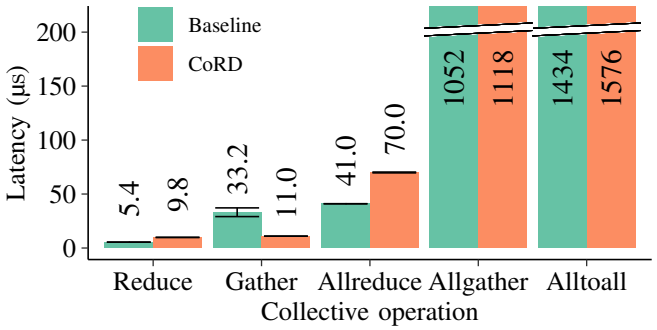


Figure 5: Latency of MPI collectives with 1 KiB operands.

Figure 4 compares the latency of MPI collective operations with 1-byte operands that ran on 288 processes across 8 nodes on the Oracle system. To amplify network effects, we disabled shared memory communication between the processes running on the same node. The communication intensity of operations varies, with Reduce transferring the least amount of data, and All-to-all transferring the most.

With CoRD, latency doubles for all presented operations except All-to-all, which is on par with the baseline RDMA communication. Considering CoRD incurs overhead multiple times on the communication critical path, a naive extrapolation of point-to-point overhead would predict higher latency than we measured. This means much of CoRD's overhead can be masked when applied to complex communication patterns.

When looking at 1024-byte messages (see Fig. 5), CoRD was very close to the baseline performance in two cases. This effect is similar to point-to-point communication, where CoRD's overhead diminishes with larger message sizes. Further investigation of larger message sizes (figure omitted due to space constraints) shows that, starting from 16 KiB operands, CoRD catches up with baseline communication for virtually all operations. Moreover, with the Gather operation (see Fig. 5), CoRD outperformed the baseline. We attribute this anomalous result to network congestion and elaborate on this issue more in Section VI-C.

Overall, we have demonstrated that, despite relatively high point-to-point overhead, CoRD's performance does not degrade with complex communication patterns. We consider our results to be applicable for a cost-optimized Cloud-operator-run RDMA network and expect CoRD to perform much worse when scaled to hundreds of compute nodes on a flagship supercomputer. Nevertheless, we demonstrate that in a very typical case of having less than 1000 communication processes, CoRD can match the performance of the baseline RDMA network.

C. Real-world Applications

To estimate the effect on real-world applications, we measure the performance of several MPI applications using the Open MPI library. For RDMA communication, Open MPI uses the `ibverbs` library, which provides a low-level interface to the InfiniBand network. We believe that MPI applications serve as suitable benchmarks because they scale effectively large numbers of processes and can fully utilize the network's potential. As a general example, we use the NAS Parallel Benchmarks (NPB) [84], a popular MPI benchmark suite. Gromacs [85], our second benchmark, is a molecular dynamics simulation software known for its sensitivity to latency, making it an extreme example and worst-case scenario for CoRD.

On the InfiniBand system (Azure), we compare communication over baseline RDMA, CoRD, and IPoIB. On the RoCE system (Oracle), we compare communication over baseline RDMA, CoRD, and TCP/IP. To amplify the influence of the network, we prevent the MPI library from using shared memory for communication. These results are pessimistic for CoRD but highlight the differences better. We chose IPoIB and TCP/IP for comparison because these protocols can communicate using high-performance NICs (InfiniBand and RoCE, respectively) while also offering fine-grained control over dataplane operations, making them functionally equivalent competitors to CoRD. Each NPB benchmark has limitations on the number of processes allowed for a run, which in our case ranged from 128 to 240 on Azure, from 256 to 278 on Oracle without hyperthreading, and from 512 to 576 on Oracle with hyperthreading. Gromacs does not impose such limitations, allowing us to utilize all available cores in each system configuration.

When running the NPB benchmarks on Azure (see Fig. 6), CoRD has nearly zero overhead over baseline kernel-bypass communication whereas IPoIB is up to $2\times$ slower. IPoIB is the slowest with the IS (integer sorting) and SP (matrix factorization) benchmarks. These two benchmarks are simultaneously data-intensive (each process sends $72\,\mathrm{Gbit/s}$ and $34\,\mathrm{Gbit/s}$, respectively) and message-intensive ($\approx 1300\,\mathrm{messages/second}$ per process). IPoIB implements a full-fledged TCP/IP software stack running on top of InfiniBand, negating many of the performance advantages high-performance networks offer. EP (embarrassingly parallel), which communicates very little, and CG (conjugate gradient), which communicates using few large messages, see a slight performance boost with CoRD. Similarly to CG, we observe CoRD marginally outperforming

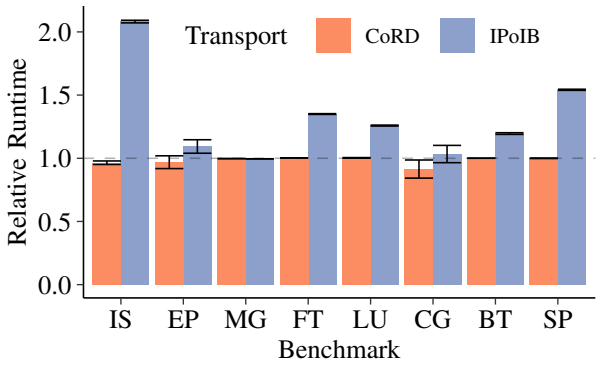


Figure 6: Runtime of NPB on Azure, relative to baseline.

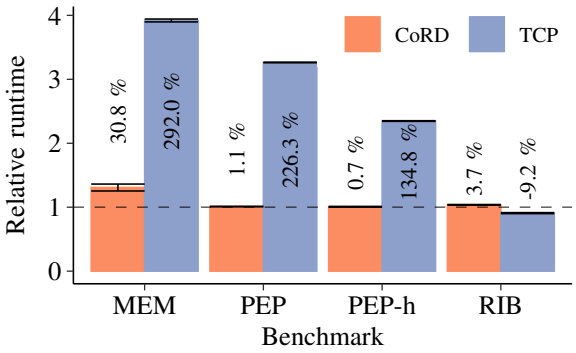


Figure 7: Runtime of Gromacs on Oracle. Annotations show the runtime overhead of CoRD and TCP communication compared to the baseline.

kernel bypass in large-message bandwidth microbenchmarks. This behavior is due to baseline RDMA experiencing more congestion than CoRD, a phenomenon that we elaborate on in Section VI-C. Overall, our implementation of CoRD has small or negligible overhead for the NPB benchmarks on both systems.

To evaluate the performance impact of CoRD on real-world applications, we measured the runtime of three Gromacs benchmarks [86] on the Oracle system. Gromacs uses input files to describe different molecular dynamics simulations. Figure 7 shows the relative runtime of the Gromacs runs with CoRD and TCP/IP communication compared to the baseline kernel-bypass version. In this experiment, TCP/IP communication shows up to $4\times$ slowdown, whereas CoRD has only $30\,\%$ overhead in the worst case.

The difference in performance between the benchmarks is due to different problem sizes (Table IV): The MEM benchmark has the smallest problem size, running only for $18\,\mathrm{s}$. As a result, MEM spends relatively more time communicating, sending on average $56\,000$ packets per process and second. On the scale of a node (72 processes), this results in 4 million packets per second, or $34\,\mathrm{Gbit/s}$ of data sent ($1/3$ of the NIC's line rate). Less communication-intensive benchmarks reduce the runtime overhead to $1\,\%$ to $3\,\%$.

Table IV: Packet rate (Ptks) and throughput (TP) of Gromacs on the baseline Oracle configuration.

	Runtime s	Per Process Pkts k/s	Per Process TP Mbit/s	Per Node Pkts M/s	Per Node TP Gbit/s
MEM	18	56.1	478	4	34
PEP	222	7.1	166	0.5	12
PEP-h	218	6.7	161	0.5	12
RIB	240	5.4	92	0.4	6.7

When measuring application benchmarks, we observed that the overhead of CoRD still ranges wildly from 1% to 30%. This range is much smaller than for the microbenchmarks, and for most experiments it remained within the single-digit-percent range. Our results also represent the worst-case scenario, because we disabled shared memory communication, which is the most efficient way to communicate between processes on the same node.

VI. Use Cases

CoRD enables the OS to intercept and control RDMA dataplane operations for various purposes. This section presents three use cases for CoRD: a traffic monitoring tool, a rate limiter, and a simple congestion control mechanism.

A. Traffic Monitoring

In traditional RDMA networks, the NIC is responsible for sending/receiving packets and reporting statistics about the traffic. However, a NIC typically reports only aggregated statistics for all processes running on the same node. This limitation makes it difficult to analyse and understand the traffic patterns of individual processes or applications.

Some per-process traffic statistics are available through the iproute2 tools [87]. Unfortunately, these statistics do not include the number of bytes or packets sent. Applications may use the `ibv_read_counters` API to query statistical information on their traffic [88]. However, this API is designed for use by the application itself, and the OS does not have easy access to these statistics. Moreover, both approaches work only for high-end NVIDIA NICs and not for NICs from other vendors.

To address this shortcoming, we added a Linux kernel tracepoint [89] to the `ib_core` driver right before it invokes the device-specific function to send a message. Such a tracepoint allows a user-defined function to be attached to a specific place in the Linux kernel. When the tracepoint is not in use, the overhead is virtually zero. We developed a simple user-level service that uses the Aya [90] framework to attach an eBPF [62] program to the newly defined tracepoint. This program records the number of bytes and packets sent by each process.

Figure 8 shows the monitoring results from one of the four nodes running the Gromacs benchmark with the MEM data set and dynamic load balancing enabled. In total, our tool observed 72 threads belonging to 36 different MPI processes. Each MPI process has two threads: a main thread and an asynchronous helper thread. The figure shows two groups

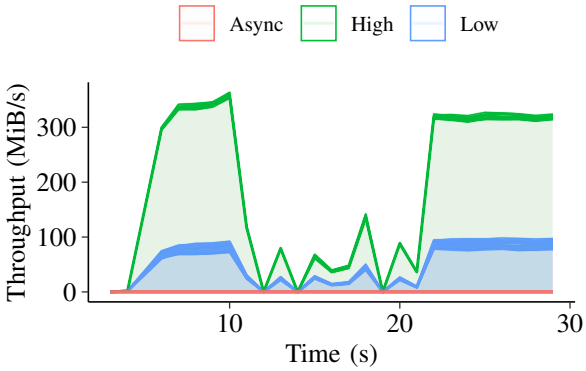


Figure 8: The per-thread traffic of Gromacs is concentrated in 12 high-traffic and 24 low-traffic threads. The 36 asynchronous MPI threads produce little to no traffic.

of main threads: one group with 12 high-traffic threads and another group with 24 low-traffic threads. This separation aligns with Gromacs's architecture, which assigns processes to two dedicated roles [85]. A sudden drop in traffic from the period from 11 s to 21 s corresponds to work rebalancing among the processes.

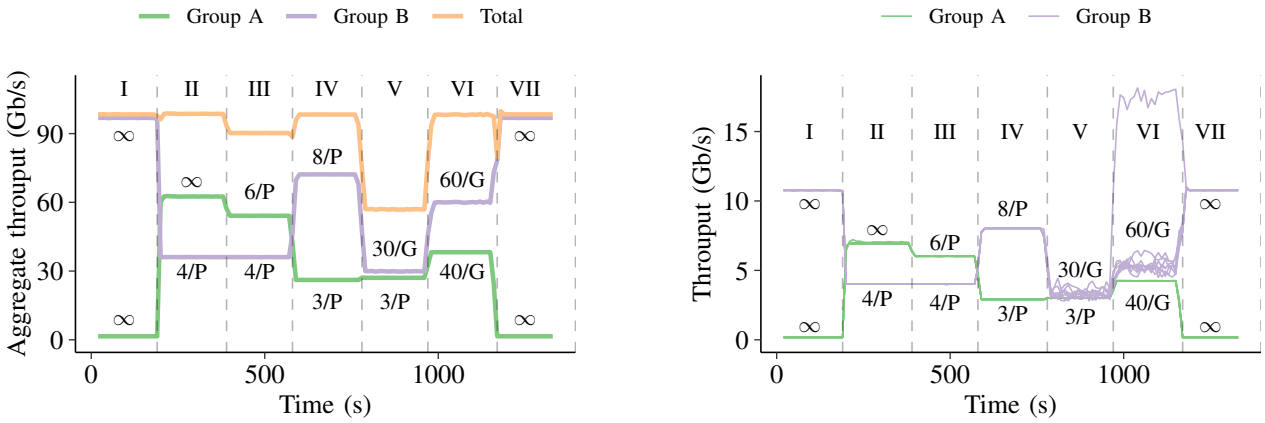
Compared to hardware-offloaded monitoring, our approach can deploy arbitrary monitoring logic without modifying the application. If necessary, we can attach eBPF programs to other existing tracepoints or create ones. CoRD-based monitoring is aware of high-level OS concepts, like processes and threads, making it more flexible than hardware-offloaded monitoring. Finally, CoRD is portable because it does not depend on any specific hardware support.

B. Rate Limiting

When multiple applications share the same node, it can be desirable to enforce a limit on the amount of traffic each process can send at once. Existing RDMA solutions offer two main approaches to rate limiting. In the first approach, the application itself implements rate limiting by relying on the InfiniBand verbs API [29] to enforce a per-flow maximum packet rate. In the second approach, the OS employs hardware virtualization to enforce rate limits [91].

The disadvantage of the first approach is that the OS has no influence over what the application sets as the rate limit. Therefore, it is hard for the OS to manage and coordinate multiple applications running on the same node. The second approach, on the other hand, is coarse-grained because it allows only for a single limit per VM or application instance. Moreover, similar to the traffic monitoring case, some rate-limiting features are only available for advanced NVIDIA NICs [92].

In contrast, CoRD allows for a fine-grained, vendor-agnostic rate-limiting implementation. To support this claim, we implemented a straightforward rate-limiting mechanism as a CoRD policy. Our rate limiter enforces a specific throughput limit at the level of Linux cgroups [69]. This interface allows the OS to enforce a limit on the traffic each process or process group can send.



(a) Aggregate throughput of process groups.

(b) Throughput of individual processes.

Figure 9: Rate limiting on the Oracle system. Without rate-limiting (∞), the two groups of processes use the RDMA network in an imbalanced manner (phases I and VII). CoRD allows the OS to impose rate limiting either on a per-process basis (x/P, phases II–V) or on a per-cgroup basis (x/G, phases V and VI). With per-cgroup rate limiting, the throughput of individual processes is noisy because our prototype implementation does not guarantee fairness.

For each cgroup, the rate limiter maintains a counter of the number of bytes sent by all processes belonging to the cgroup within a $200\,\mathrm{ms}$ sliding window. If the process attempting to send a new message is about to exceed the limit allocated for its cgroup, the rate limiter puts the process to sleep for a short period of time. The sleep time is calculated based on the current throughput of the cgroup and the amount of data the process is trying to send. The cgroup interface allows for the creation of hierarchical resource limits, so the rate limiter needs to ensure that the limit is enforced for every cgroup in the hierarchy.

Previously, we stated that the policies must have minimal overhead for CoRD to be competitive. In contrast to traffic monitoring, the rate limiter has a latency overhead of around $300\,\mathrm{ns}$, even when rate limiting is not enforced. We attribute this overhead mainly to the requirement of grabbing a mutex to access the cgroup data structure. We believe that using a faster mutual exclusion mechanism would reduce this overhead, but from our point of view, the convenience of the cgroup interface outweighs the overhead.

To demonstrate how our rate limiter works, we conducted an experiment on two Oracle Cloud nodes. Each node ran 18 instances of the `ib_send_bw` benchmark, split into two groups of 9 processes. The benchmarks on one node were sending $64\,\mathrm{KiB}$ messages to the benchmarks on the other node. A process in the first group (group A) sends messages over a single queue pair (connection), whereas a process in the second group (group B) sends messages over 64 queue pairs in parallel. Every $10\,\mathrm{s}$, each process reports its observed throughput. Approximately every $180\,\mathrm{s}$, we change the configuration of the rate limiter to observe how the benchmarks react to the change.

Figure 9a shows the aggregate throughput observed by each group. The rate limiter changes configuration 6 times, creating 7 periods with different configurations. The benchmarks start by sending at full speed (phase I) with no intervention from the rate limiter. Processes in group B receive a disproportionate share of the bandwidth because the NIC implements load balancing on a per-queue-pair basis. The aggregate throughput of all the processes is $98.4\,\mathrm{Gbit/s}$, just a little short of the theoretical maximum of $100\,\mathrm{Gbit/s}$.

Next, we prioritize the processes in group A over those in group B. To achieve this, we set the rate limit to $4\,\mathrm{Gbit/s}$ for each process in group B, so that group A can claim the remaining available bandwidth. To set the rate limit for each process in group B precisely, we created an individual cgroup for each process. As a result, in phase II, processes in group A receive approximately $6.9\,\mathrm{Gbit/s}$ throughput. The aggregate throughput of all processes remains the same as in phase I.

Phase III limits each process in group A to $6\,\mathrm{Gbit/s}$. Considering that the aggregate throughput of all processes is thus limited to $9 \cdot 4\,\mathrm{Gbit/s} + 9 \cdot 6\,\mathrm{Gbit/s} = 90\,\mathrm{Gbit/s}$, the observed aggregate throughput also drops. Phase IV limits each process in group A to $3\,\mathrm{Gbit/s}$ and each process in group B to $8\,\mathrm{Gbit/s}$. In all these cases, all the processes observe the expected throughput.

In phase V, we moved all the group B processes to a single cgroup and set the rate limit for this cgroup to $30\,\mathrm{Gbit/s}$. Our current algorithm does not guarantee fairness among the processes within the same cgroup. As a result, processes sharing a cgroup may experience variation in observed throughput (see Fig. 9b). A better algorithm would also provide fairness [93], but rate limiting algorithms are beyond the scope of this paper.

In phase VI, we set the aggregate rate limit to $40\,\mathrm{Gbit/s}$ for group A and to $60\,\mathrm{Gbit/s}$ for group B. In this case, most of the processes within group A observe equal throughput, but there is even more variation in group B. Again, we do not guarantee fairness between the processes within one cgroup. Finally, in phase VII, we remove the rate limit for all processes, and the bandwidth distribution returns to the state of phase I.

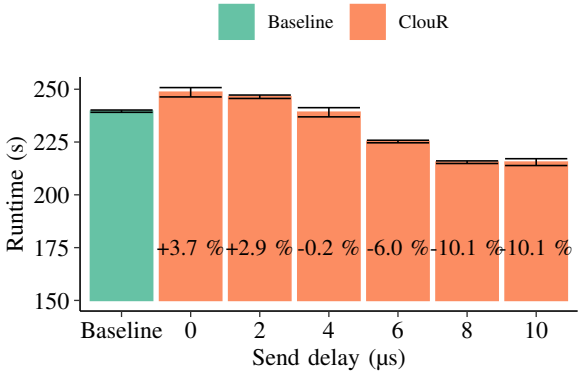


Figure 10: Reducing network congestion with CoRD

C. Congestion Control

The final use case demonstrates how CoRD can implement RDMA congestion control. Hardware-level protocols like DCQCN [48] rate limit the sender in response to congestion by delaying send operations. We simulated this behaviour, by adding a CoRD policy to delay the posting of new send requests. To have a complete protocol, we would also need to implement congestion sensing or dynamic send delay adjustment, which we omitted for our experiment.

To test potential congestion control scenarios, we ran Gromacs on the Oracle system and compared the performance with and without CoRD. CoRD can delay the posting of new send requests by a variable interval. Figure 10 shows Gromacs with the RIB dataset running on 8 nodes with hyperthreading enabled. Although the baseline RDMA configuration suffers from congestion, CoRD still runs slower without adding a delay. However, when adding a send delay to CoRD, the application performance improves by up to 10 %. This result suggests that CoRD can implement congestion control for real-world applications.

When Gromacs ran with the PEP and PEP-h datasets, we observed no significant changes in the performance of the application when adding a send delay. On the other hand, when running Gromacs with the MEM dataset, adding any send delay increased CoRD's overhead even beyond the already existing 30 %. So, these mechanisms need to be able to sense congestion and dynamically adjust the delay to be effective.

To understand how send delay affects network congestion, we measured the latency of individual MPI collective operations. These operations facilitate data exchange among multiple processes of the same MPI application. To isolate the overhead added by CoRD, we reimplemented the send delay in the user-level InfiniBand verbs driver communicating using baseline kernel-bypass.

Figure 11 shows the average latency of MPI Gather across 576 processes with a varying send delay. This operation gathers the data supplied by each participating process at a designated *root* process. The MPI library is responsible for establishing a communication graph, which reduces latency and improves network utilization.

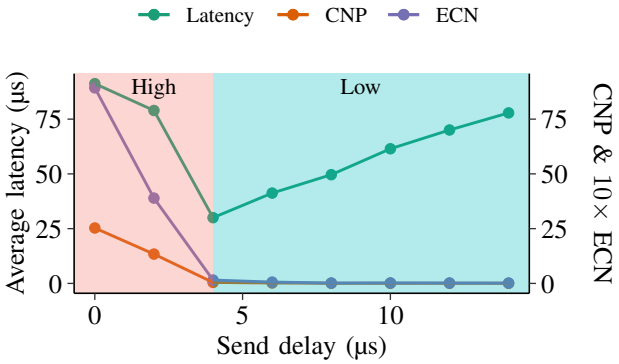


Figure 11: MPI Gather operation with varying send delay. Even in a lossless network, delaying each send operation limits each process's throughput and reduces congestion measured in CNP packets sent and ECN-marked packets received. Once congestion disappears, higher send delays are detrimental to latency.

We measure congestion by counting the average number of Explicit Congestion Notification (ECN) marked packets received by the NIC and Congestion Notification Packets (CNP) sent by the NIC. The zero-send-delay case records up to 890 ECN-marked packets, 25 CNP packets, and 91 µs latency per Gather operation, which is a clear sign of high congestion. With a 4 µs send delay, the number of ECN-marked packets drops to 15, the number of CNP packets drops to 0.5 (on average), and the latency drops to 30 µs. Adding more delay cannot reduce congestion further, resulting in an increased Gather operation latency.

CoRD with zero send delay achieves 53 µs latency, which is faster than the baseline kernel-bypass with zero send delay. The latency achieved by CoRD is the closest to 6 µs send delay implemented in the userspace for the baseline kernel, but it would be wrong to conclude that CoRD adds that much latency. CoRD adds latency before and after each dataplane operation, which is not comparable to the user-level send delay.

Although it is faster than the baseline kernel without send delays, CoRD is still slower than what is achievable with the optimal send delay. This result suggests that CoRD can implement RDMA congestion control, but also has low enough overhead for having a window for useful send delay values.

A possible criticism of using CoRD for such a task is that it is too slow to react timely to congestion events [94]. Our results suggest that there are high-level sources of congestion, for which CoRD is quick enough. This role becomes even more important when applications on the same host fail to coordinate their communication patterns, preventing NICs and switches from alleviating congestion.

VII. Discussion and Conclusion

Modern cloud systems primarily rely on socket-based networking to manage a myriad of distributed applications. Key to this operation is the precise control over application communication channels using tools such as Linux packet

filtering [95] or eBPF [62]. Despite persistent enhancements in high-performance TCP/IP [96], [97], it still falls significantly short of the performance of RDMA networks.

High-performance networking is diverse and complex [8], [11], [25], [29], [98], so, unsurprisingly, there are no "one size fits all" solutions [99]. Striving for optimal problem-specific solutions, researchers have abandoned polling [50], [100], zero-copy [99], [101], lossless congestion control [102], [103], hardware offloading [10], [51], [104], [105], and one-sided operations [56], [106], [107] to achieve higher flexibility and resource utilization. Thus, forgoing performance-enhancing features is not entirely new.

Nevertheless, the RDMA-networking community avoids placing the OS kernel on the data path. Instead, existing work adds another device to the path of a packet, either in the form of dedicated CPU cores [17], [25], [27], [71], [108], [109] or by offloading complex data-path interception logic to the NIC. Such offloading requires either an expensive SmartNIC [110], [111], [112], [113] or hardware modifications [43], [114], [115].

Some functionality — including the use cases we demonstrated in Section VI — can also be implemented in the application logic using, for example, based on user-level tracepoints. Despite being better for performance, the OS cannot use this approach to reliably enforce policies or coordinate shared resource access among multiple applications running on the same system (see Section VI-B). Moreover, coordinating coherent behaviour among independent applications may turn out to be infeasible in practice.

CoRD's performance with small messages is a major limitation. Regrettably, we cannot assess the severity of this limitation in real-world scenarios due to limited research on application characterization in this area. A study of the Mira supercomputing system analyzed the volume of data sent by point-to-point and collective MPI operations [116, Fig. 10 and 20]. Most MPI operations transmitted data ranging from kilobytes to hundreds of kilobytes per operation. We measured that GROMACS sends approximately $2\,\mathrm{KiB}$ per message, placing it on the lower end of this spectrum. Although some applications do transmit very small messages, typical data sizes fall within ranges where CoRD's overhead remains modest.

We consider per-core message rate as the most important application characteristic to ensure low overhead from CoRD. For instance, if an application sends $100\,000$ messages per core every second, and CoRD adds $1\,\mu\mathrm{s}$ overhead per message, the application can suffer no more than $10\,\%$ overhead. Applications with complex user logic (e.g., GROMACS; unlike ToR-switches) spend significant time computing, naturally limiting the maximum per-core message rate. Based on the Mira study [116], real-world per-core message rates are likely much lower than our hypothetical example. Future high-performance networks are also more likely to coincide with higher per-node core counts, which should keep CoRD's overhead the same, but making multi-tenancy more relevant.

Our work is based on the idea that the problem with system calls is not the cost of user-kernel transitions in hardware but rather an obsolete and inefficient API; in this case, POSIX. This idea aligns with the existing body of work [24], [26], [117], [118], [119], and our results support the observation of kernel bypass being dispensable. We propose a small incremental change to the existing RDMA network architecture, which, in the context of a Cloud environment, has a very small negative performance impact but allows for a substantial increase in OS control over network communication.

In the future, we strive for a smaller per-message overhead and to incorporate more elaborate policies. Our early results suggest that going below $1\,\mu\mathrm{s}$ is achievable. If successful, CoRD can be applied in additional domains like high-performance storage [120], [121], where APIs are built on similar concepts.

Acknowledgments

The research and the work presented in this paper has been supported by the German priority program 1648 "Software for Exascale Computing" via the research project FFMK [122] (HA-2461/10-2), by the German Research Foundation (DFG) within the Collaborative Research Center HAEC and by the Center for Advancing Electronics Dresden (cfaed). The authors acknowledge support from Oracle for Research for providing cloud computing resources in project 172 and by the Federal Ministry of Education and Research of Germany in the programme of "Souverän. Digital. Vernetzt." (joint project 6G-life, project ID 16KISK001K). We also thank the anonymous reviewers of HotOS, WORDS, Eurosys, and IPDPS for their valuable comments and suggestions.

References

[1] J. Carreira, P. Fonseca, A. Tumanov, A. Zhang, and R. Katz, "Cirrus: A Serverless Framework for End-to-end ML Workflows," in *Proceedings of the ACM Symposium on Cloud Computing*, ser. SoCC '19, New York, NY, USA: Association for Computing Machinery, Nov. 20, 2019, pp. 13–24, ISBN: 978-1-4503-6973-2. DOI: 10.1145/3357223.3362711.

[2] L. Feng, P. Kudva, D. Da Silva, and J. Hu, "Exploring Serverless Computing for Neural Network Training," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, Jul. 2018, pp. 334–341. DOI: 10.1109/CLOUD.2018.00049.

[3] Q. Pu, Shivaram Venkataraman, and Ion Stoica, "Shuffling, Fast and Slow: Scalable Analytics on Serverless Infrastructure," in *Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation*, Boston, MA, USA, 2019.

[4] Microsoft. "High performance computing VM sizes," Linux virtual machines in Azure, Accessed: Apr. 14, 2020. [Online]. Available: https://docs.microsoft.com/en-us/azure/virtual-machines/sizes-hpc.

[5] Amazon Web Services, Inc. "Elastic Fabric Adapter," Accessed: Apr. 14, 2020. [Online]. Available: https://aws.amazon.com/hpc/efa/.

[6] Oracle. "Compute Shapes," Oracle Cloud Infrastructure, Accessed: Jun. 22, 2023. [Online]. Available: https://docs.oracle.com/en-us/iaas/Content/Compute/References/computeshapes.htm.

[7] M. Abranches, O. Michel, and E. Keller, "Getting back what was lost in the era of high-speed software packet processing," in *Proceedings of the 21st ACM Workshop on Hot Topics in Networks*, Austin Texas: ACM, Nov. 14, 2022, pp. 228–234, ISBN: 978-1-4503-9899-2. DOI: 10.1145/3563766.3564114.

[8] Cray Inc, *XC Series GNI and DMAPP API User Guide CLE70UP02 (S-2446)*, 2019.

[9] InfiniBand Trade Association, *InfiniBand Architecture Specification*, 1.3. InfiniBand Trade Association, Mar. 3, 2015, vol. 1. [Online]. Available: https://cw.infinibandta.org/document/dl/8567.

[10] S. Peter et al., "Arrakis: The Operating System is the Control Plane," in *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation, 2014*, ser. OSDI '14, Broomfield, CO, USA: USENIX Association, 2014, pp. 2–17, ISBN: 978-1-931971-16-4.

[11] DPDK Project. "Data Plane Development Kit," DPDK, Accessed: Sep. 26, 2019. [Online]. Available: https://www.dpdk.org/.

[12] OpenMPI. "Open MPI v5.0.x — Open MPI 5.0.x documentation," Accessed: Dec. 8, 2022. [Online]. Available: https://docs.open-mpi.org/en/v5.0.x/.

[13] B. Rothenberger, K. Taranov, A. Perrig, and T. Hoefler, "ReDMArk: Bypassing RDMA Security Mechanisms," in *30th USENIX Security Symposium*, Vancouver, B.C., 2021. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/rothenberger.

[14] K. Taranov, B. Rothenberger, A. Perrig, and T. Hoefler, "sRDMA – Efficient NIC-based Authentication and Encryption for Remote Direct Memory Access," presented at the 2020 USENIX Annual Technical Conference (USENIX ATC 20), 2020, pp. 691–704, ISBN: 978-1-939133-14-4. Accessed: Apr. 29, 2021. [Online]. Available: https://www.usenix.org/conference/atc20/presentation/taranov.

[15] A. K. Simpson, A. Szekeres, J. Nelson, and I. Zhang, "Securing RDMA for High-Performance Datacenter Storage Systems," in *12th USENIX Workshop on Hot Topics in Cloud Computing*, ser. HotCloud 20, USENIX Association, 2020.

[16] Y. Zhang, Y. Tan, B. Stephens, and M. Chowdhury, "Justitia: Software multi-tenancy in hardware kernel-bypass networks," presented at the 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22), 2022, pp. 1307–1326, ISBN: 978-1-939133-27-4. Accessed: Jul. 10, 2023. [Online]. Available: https://www.usenix.org/conference/nsdi22/presentation/zhang-yiwen.

[17] D. Kim et al., "FreeFlow: Software-based Virtual RDMA Networking for Containerized Clouds," in *Proceedings of the 16th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI, Boston, MA, USA, 2019, pp. 113–125, ISBN: 978-1-931971-49-2. DOI: 10.5555/3323234.3323245.

[18] Z. 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*, ser. SIGCOMM '20, New York, NY, USA: Association for Computing Machinery, Jul. 30, 2020, pp. 1–14, ISBN: 978-1-4503-7955-7. DOI: 10/gg9rjq.

[19] B. Gerofi, M. Takagi, A. Hori, G. Nakamura, T. Shirasawa, and Y. Ishikawa, "On the Scalability, Performance Isolation and Device Driver Transparency of the IHK/McKernel Hybrid Lightweight Kernel," in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Chicago, IL, USA: IEEE, May 2016, pp. 1041–1050, ISBN: 978-1-5090-2140-6. DOI: 10.1109/IPDPS.2016.80.

[20] M. Jamshed, Y. Moon, D. Kim, D. Han, and K. Park, "mOS: A Reusable Networking Stack for Flow Monitoring Middleboxes," in *Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation*, Boston, MA, USA, 2017.

[21] M. Giampapa, T. Gooding, T. Inglett, and R. W. Wisniewski, "Experiences with a Lightweight Supercomputer Kernel: Lessons Learned from Blue Gene's CNK," in *2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, New Orleans, LA, USA: IEEE, Nov. 2010, pp. 1–10, ISBN: 978-1-4244-7557-5. DOI: 10.1109/SC.2010.22.

[22] C. Weinhold, A. Lackorzynski, and H. Härtig, "FFMK: An HPC OS Based on the L4Re Microkernel," in *Operating Systems for Supercomputers and High Performance Computing*, B. Gerofi, Y. Ishikawa, R. Riesen, and R. W. Wisniewski, Eds., Singapore: Springer Singapore, 2019, pp. 335–357, ISBN: 9789811366246. DOI: 10.1007/978-981-13-6624-6_19.

[23] A. Belay, G. Prekas, C. Kozyrakis, A. Klimovic, S. Grossman, and E. Bugnion, "IX: A Protected Dataplane Operating System for High Throughput and Low Latency," in *11th USENIX Symposium on Operating Systems Design and Implementation*, ser. OSDI '14, Oct. 2014, pp. 49–65, ISBN: 978-1-931971-16-4.

[24] I. Zhang et al., "The Demikernel Datapath OS Architecture for Microsecond-scale Datacenter Systems," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, ser. SOSP '21, New York, NY, USA: Association for Computing Machinery, Oct. 26, 2021, pp. 195–211, ISBN: 978-1-4503-8709-5. DOI: 10/gnkjj5.

[25] M. Marty et al., "Snap: A microkernel approach to host networking," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles - SOSP '19*, Huntsville, Ontario, Canada: ACM Press, 2019, pp. 399–413, ISBN: 978-1-4503-6873-5. DOI: 10/ggcdnx.

[26] X. Wei, F. Lu, R. Chen, and H. Chen, "KRCORE: A microsecond-scale RDMA control plane for elastic computing," Dec. 28, 2021. arXiv: 2201.11578 [cs]. Accessed: Jan. 28, 2022. [Online]. Available: http://arxiv.org/abs/2201.11578.

[27] S.-Y. Tsai and Y. Zhang, "LITE Kernel RDMA Support for Datacenter Applications," in *Proceedings of the 26th Symposium on Operating Systems Principles - SOSP '17*, Shanghai, China: ACM Press, 2017, pp. 306–324, ISBN: 978-1-4503-5085-3. DOI: 10/ggscxn.

[28] POSIX 2017, "POSIX.1-2017," IEEE, 2017. DOI: 10.1109/IEEESTD.2018.8277153.

[29] Mellanox Technologies, "RDMA Aware Networks Programming User Manual," Mellanox Technologies, 1.7, 2015, p. 216.

[30] 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, Istanbul, Turkey: ACM Press, 2015, pp. 17–30, ISBN: 978-1-4503-3450-1. DOI: 10/ggscxm.

[31] A. Mouzakitis et al., "Lightweight and Generic RDMA Engine Para-Virtualization for the KVM Hypervisor," in *2017 International Conference on High Performance Computing & Simulation (HPCS)*, Genoa, Italy: IEEE, Jul. 2017, pp. 737–744, ISBN: 978-1-5386-3249-9 978-1-5386-3250-5. DOI: 10/ggscxk.

[32] L. Gerhorst, B. Herzog, S. Reif, W. Schröder-Preikschat, and T. Hönig, "AnyCall: Fast and Flexible System-Call Aggregation," in *Proceedings of the 11th Workshop on Programming Languages and Operating Systems*, ser. PLOS '21, New York, NY, USA: Association for Computing Machinery, Oct. 25, 2021, pp. 1–8, ISBN: 978-1-4503-8707-1. DOI: 10/gm5nz5.

[33] K. Elphinstone and G. Heiser, "From L3 to seL4 what have we learnt in 20 years of L4 microkernels?" In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, Farminton Pennsylvania: ACM, Nov. 3, 2013, pp. 133–150, ISBN: 978-1-4503-2388-8. DOI: 10.1145/2517349.2522720.

[34] G. M. Kurtzer, V. Sochat, and M. W. Bauer, "Singularity: Scientific containers for mobility of compute," *Plos one*, vol. 12, no. 5, e0177459, May 11, 2017, ISSN: 1932-6203. DOI: 10/f969fz.

[35] A. Madhavapeddy et al., "Unikernels: Library operating systems for the cloud," *Asplos '13 proceedings of the eighteenth international conference on architectural support for programming languages and operating systems*, 2013, ISSN: 03621340. DOI: 10.1145/2499368.2451167.

[36] M. Bauer. "Paranoid Penguin: An Introduction to Novell AppArmor," Accessed: Dec. 14, 2022. [Online]. Available: https://dl.acm.org/doi/fullHtml/10.5555/1149826.1149839.

[37] S. Smalley, C. Vance, and W. Salamon, *Implementing SELinux as a Linux Security Module*, 2002.

[38] S. Miano, F. Risso, M. V. Bernal, M. Bertrone, and Y. Lu, "A Framework for eBPF-Based Network Functions in an Era of Microservices," *Ieee transactions on network and service management*, vol. 18, no. 1, pp. 133–151, Mar. 2021, ISSN: 1932-4537. DOI: 10.1109/TNSM.2021.3055676.

[39] M. Abranches, O. Michel, E. Keller, and S. Schmid, "Efficient Network Monitoring Applications in the Kernel with eBPF and XDP," in *2021 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Chandler, AZ United States: IEEE, Nov. 2021, pp. 28–34. DOI: 10.1109/NFV-SDN53031.2021.9665095.

[40] M. Lieber, V. Grützun, R. Wolke, M. S. Müller, and W. E. Nagel, "Highly Scalable Dynamic Load Balancing in the Atmospheric Modeling System COSMO-SPECS+FD4," in *Applied Parallel and Scientific Computing*, K. Jónasson, Ed., Berlin, Heidelberg: Springer, 2012, pp. 131–141, ISBN: 978-3-642-28151-8. DOI: 10.1007/978-3-642-28151-8_13.

[41] F. Zheng et al., "GoldRush: Resource efficient in situ scientific data analytics using fine-grained interference aware execution," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, Denver Colorado: ACM, Nov. 17, 2013, pp. 1–12, ISBN: 978-1-4503-2378-9. DOI: 10.1145/2503210.2503279.

[42] M. Copik, K. Taranov, A. Calotoiu, and T. Hoefler. "rFaaS: Enabling High Performance Serverless with RDMA and Leases." arXiv: 2106.13859 [cs], Accessed: Jun. 21, 2023. [Online]. Available: http://arxiv.org/abs/2106.13859, pre-published.

[43] L. Liss, "On Demand Paging for User-level Networking," presented at the OpenFabrics Workshop, 2013. [Online]. Available: https://openfabrics.org/images/eventpresos/workshops2013/2013_Workshop_Tues_0930_liss_odp.pdf.

[44] NVIDIA, *ConnectX-6 VPI Card Product Brief*, 2020. [Online]. Available: https://www.mellanox.com/files/doc-2020/pb-connectx-6-vpi-card.pdf.

[45] F. Petrini, D. J. Kerbyson, and S. Pakin, "The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q," in *Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, Phoenix AZ USA: ACM, Nov. 15, 2003, p. 55, ISBN: 978-1-58113-695-1. DOI: 10.1145/1048935.1050204.

[46] IEEE, *802.1Qbb - Priority-based Flow Control*, Jun. 16, 2011. Accessed: Oct. 15, 2019. [Online]. Available: https://1.ieee802.org/dcb/802-1qbb/.

[47] C. Guo et al., "RDMA over Commodity Ethernet at Scale," in *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference - SIGCOMM '16*, ser. SIGCOMM '16, Florianopolis, Brazil: ACM Press, 2016, pp. 202–215, ISBN: 978-1-4503-4193-6. DOI: 10.1145/2934872.2934908.

[48] Y. Zhu et al., "Congestion Control for Large-Scale RDMA Deployments," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM, London, UK: ACM, 2015, pp. 523–536, ISBN: 978-1-4503-3542-3. DOI: 10.1145/2785956.2787484.

[49] J. Brar and P. Vincent, "First Principles: Building a high performance network in the public cloud," Dec. 12, 2022. Accessed: Feb. 7, 2024. [Online]. Available: https://www.youtube.com/watch?v=ca_OGqCVHDM.

[50] A. Venkatesh et al., "A case for application-oblivious energy-efficient MPI runtime," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Austin Texas: ACM, Nov. 15, 2015, pp. 1–12, ISBN: 978-1-4503-3723-6. DOI: 10.1145/2807591.2807658.

[51] Linux Foundation. "Networking:toe [Wiki]," Accessed: May 17, 2021. [Online]. Available: https://wiki.linuxfoundation.org/networking/toe.

[52] M. McInnes and M. McGovern. "Microsoft Azure Network Adapter (MANA) overview," Accessed: Feb. 20, 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/virtual-network/accelerated-networking-mana-overview.

[53] D. Lenoski and N. Dukkipati. "Introducing Falcon: A reliable low-latency hardware transport," Google Cloud Blog, Accessed: Feb. 20, 2024. [Online]. Available: https://cloud.google.com/blog/topics/systems/introducing-falcon-a-reliable-low-latency-hardware-transport.

[54] NVIDIA Corporation, *ConnectX-7 400G Adapters*, 2022. [Online]. Available: https://nvdam.widen.net/s/csf8rmnqwl/infiniband-ethernet-datasheet-connectx-7-ds-nv-us-2544471.

[55] NVIDIA, "NVIDIA BlueField-3 DPU," Datasheet, 2023. Accessed: Oct. 12, 2023. [Online]. Available: https://resources.nvidia.com/en-us-accelerated-networking-resource-library/datasheet-nvidia-bluefield.

[56] A. Dragojevic, D. Narayanan, and M. Castro, "RDMA Reads: To Use or Not to Use?" *Bulletin of the technical committee on data engineering*, vol. 40, no. 1, pp. 3–14, Mar. 2017.

[57] A. Kalia, M. Kaminsky, and D. G. Andersen, "FaSST: Fast, Scalable and Simple Distributed Transactions with Two-sided (RDMA) Datagram RPCs," in *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation*, Savannah, GA, USA: USENIX Association, Nov. 2–4, 2016, ISBN: ISBN 978-1-931971-33-1.

[58] A. Kalia, M. Kaminsky, and D. G. Andersen, "Datacenter RPCs can be General and Fast," in *16th USENIX Symposium on Networked Systems Design and Implementation*, Boston, MA: USENIX Association, 2019, ISBN: 978-1-931971-49-2. [Online]. Available: https://www.usenix.org/conference/nsdi19/presentation/kalia.

[59] InfiniBand Trade Association, *Supplement to InfiniBand Architecture Specification: RoCEv2*. InfiniBand Trade Association, Sep. 2, 2014. [Online]. Available: https://cw.infinibandta.org/document/dl/7781.

[60] M. Ko and A. Nezhinsky. "Internet Small Computer System Interface (iSCSI) Extensions for the Remote Direct Memory Access (RDMA) Specification," Accessed: Oct. 19, 2021. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7145.

[61] P. Schwan, "Lustre: Building a File System for 1,000-node Clusters," presented at the Linux Simposium, 2003.

[62] Tigera Inc. "eBPF use cases | Calico Documentation," Accessed: May 29, 2023. [Online]. Available: https://docs.tigera.io/calico/latest/operations/ebpf/use-cases-ebpf.

[63] C. Mitchell, Y. Geng, and J. Li, "Using One-Sided RDMA Reads to Build a Fast, CPU-Efficient Key-Value Store," presented at the USENIX Annual Technical Conference (USENIX ATC 13), 2013, pp. 103–114, ISBN: 978-1-931971-01-0. Accessed: Mar. 5, 2020. [Online]. Available: https://www.usenix.org/conference/atc13/technical-sessions/presentation/mitchell.

[64] A. Dragojević, D. Narayanan, O. Hodson, and M. Castro, "FaRM: Fast Remote Memory," in *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation*, Seattle, WA, USA: USENIX Association, 2014, pp. 401–414, ISBN: ISBN 978-1-931971-09-6. Accessed: Oct. 19, 2020. [Online]. Available: https://www.usenix.org/conference/nsdi14/technical-sessions/dragojevi%C4%87.

[65] L. V. Kale and S. Krishnan, "A Portable Concurrent Object Oriented System Based On C++," in *Conference on Object-Oriented Programming Systems, Languages and Applications*, ser. OOPSLA '93, Washington, DC, USA: ACM Press, 1993, pp. 91–108.

[66] M. Bauer, S. Treichler, E. Slaughter, and A. Aiken, "Legion: Expressing locality and independence with logical regions," in *International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC, Salt Lake City, UT: IEEE, Nov. 2012, pp. 1–11, ISBN: 978-1-4673-0806-9. DOI: 10/gf9t42.

[67] MPI Forum, "MPI: A Message-Passing Interface Standard," Message Passing Interface Forum, 3.1, Jun. 4, 2015, p. 868.

[68] P. Pandit, *Rdma-system - RDMA subsystem configuration*, 2020. Accessed: Dec. 14, 2022. [Online]. Available: https://manpages.ubuntu.com/manpages/focal/man8/rdma-system.8.html.

[69] T. Heo. "Control Group v2," Accessed: Oct. 17, 2019. [Online]. Available: https://www.kernel.org/doc/Documentation/cgroup-v2.txt.

[70] Intel Corporation, "Intel® Data Streaming Accelerator Architecture Specification," Intel Corporation, 2.0, Sep. 2022.

[71] J. Corbet. "Ringing in a new asynchronous I/O API," LWN.net, Accessed: Dec. 14, 2022. [Online]. Available: https://lwn.net/Articles/776703/.

[72] K. Taranov, R. Bruno, G. Alonso, and T. Hoefler, "Naos: Serialization-free RDMA networking in Java," in *USENIX ATC*, 2021, p. 15.

[73] K. Al-Attar, A. Shafi, M. Abduljabbar, H. Subramoni, and D. K. Panda, "Spark Meets MPI: Towards High-Performance Communication Framework for Spark using MPI," in *2022 IEEE International Conference on Cluster Computing (CLUSTER)*, Sep. 2022, pp. 71–81. DOI: 10.1109/CLUSTER51413.2022.00022.

[74] X. Lu and D. Shankar, "Scalable and Distributed Key-Value Store-based Data Management Using RDMA-Memcached," p. 12, 2011.

[75] D. Gruss, M. Lipp, M. Schwarz, R. Fellner, C. Maurice, and S. Mangard, "KASLR is Dead: Long Live KASLR," in *Engineering Secure Software and Systems*, E. Bodden, M. Payer, and E. Athanasopoulos, Eds., vol. 10379, Cham: Springer International Publishing, 2017, pp. 161–176, ISBN: 978-3-319-62104-3 978-3-319-62105-0. DOI: 10.1007/978-3-319-62105-0_11.

[76] J. Corbet. "The current state of kernel page-table isolation," LWN.net, Accessed: Jan. 20, 2023. [Online]. Available: https://lwn.net/Articles/741878/.

[77] M. Lipp et al., "Meltdown: Reading kernel memory from user space," *Communications of the acm*, vol. 63, no. 6, pp. 46–56, May 21, 2020, ISSN: 0001-0782, 1557-7317. DOI: 10.1145/3357033.

[78] perftest. "Perftest," perftest, Accessed: Apr. 13, 2020. [Online]. Available: https://github.com/linux-rdma/perftest.

[79] C. David et al., "LogP: Towards a realistic model of parallel computation," in *Proceedings of the fourth ACM SIGPLAN symposium on Principles and practice of parallel programming*, 1993. Accessed: Dec. 12, 2022. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/155332.155333.

[80] A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman, "LogGP: Incorporating long messages into the LogP model—one step closer towards a realistic model for parallel computation," in *Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures - SPAA '95*, Santa Barbara, California, United States: ACM Press, 1995, pp. 95–105, ISBN: 978-0-89791-717-9. DOI: 10.1145/215399.215427.

[81] R. M. Karp, A. Sahay, E. E. Santos, and K. E. Schauser, "Optimal broadcast and summation in the LogP model," in *Proceedings of the fifth annual ACM symposium on Parallel algorithms and architectures - SPAA '93*, Velen, Germany: ACM Press, 1993, pp. 142–153, ISBN: 978-0-89791-599-1. DOI: 10.1145/165231.165250.

[82] D. K. Panda, *OSU Micro-Benchmarks*, Ohio, USA: The Ohio State University, 2023. Accessed: Dec. 4, 2023. [Online]. Available: http://mvapich.cse.ohio-state.edu/benchmarks/.

[83] E. Gabriel et al., "Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, vol. 3241, Berlin, Heidelberg: Springer, 2004, pp. 97–104, ISBN: 978-3-540-30218-6. DOI: 10/bxhsv5.

[84] D. H. Bailey et al., "The NAS Parallel Benchmarks," 1994.

[85] B. Hess, C. Kutzner, David van der Spoel, and E. Lindahl, "GROMACS 4: Algorithms for Highly Efficient, Load-Balanced, and Scalable Molecular Simulation," *Journal of chemical theory and computation*, vol. 4, no. 3, pp. 435–447, Mar. 2008, ISSN: 1549-9618, 1549-9626. DOI: 10/b7nkp6.

[86] Bert L. de Groot and Helmut Grubmüller. "A free GROMACS benchmark set: Input files for GROMACS performance evaluations," Accessed: May 20, 2024. [Online]. Available: https://www.mpinat.mpg.de/grubmueller/bench.

[87] S. Hemminger et al. "Iproute2/iproute2.git - Iproute2 routing commands and utilities," Accessed: Dec. 24, 2023. [Online]. Available: https://git.kernel.org/pub/scm/network/iproute2/iproute2.git.

[88] A. Rosenbaum. "[PATCH RFC v2] Introduce verbs API for traffic and event counters — Linux RDMA and InfiniBand development," Accessed: Dec. 24, 2023. [Online]. Available: https://www.spinics.net/lists/linux-rdma/msg58579.html.

[89] S. Rostedt. "Using the TRACE_EVENT() macro (Part 1)," LWN.net, Accessed: Dec. 24, 2023. [Online]. Available: https://lwn.net/Articles/379903/.

[90] A. Decina, D. Tucker, T. Duberstein, M. Rostecki, A. Stoycos, and W. Findlay. "Aya," Accessed: Dec. 24, 2023. [Online]. Available: https://aya-rs.dev/.

[91] NVIDIA, *NVIDIA® OpenFabrics Enterprise Distribution for Linux Documentation*, v23.10-1.1.9.0 LTS. 2023. Accessed: Jan. 10, 2024. [Online]. Available: https : / / docs . nvidia . com / networking / display / MLNXOFEDv23101190LTS.

[92] A. Rosenbaum and B. Wang. "Ibv_modify_qp_rate_limit(3)," Accessed: Jan. 10, 2024. [Online]. Available: https://man7.org/linux/man-pages/man3/ibv_modify_qp_rate_limit.3.html.

[93] S. Radhakrishnan, Y. Geng, V. Jeyakumar, A. Kabbani, G. Porter, and A. Vahdat, "SENIC: Scalable NIC for End-Host Rate Limiting," presented at the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14), 2014, pp. 475–488, ISBN: 978-1-931971-09-6. Accessed: Jan. 10, 2024. [Online]. Available: https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/radhakrishnan.

[94] L. Barroso, M. Marty, D. Patterson, and P. Ranganathan, "Attack of the killer microseconds," *Communications of the acm*, vol. 60, no. 4, pp. 48–54, Mar. 24, 2017, ISSN: 0001-0782. DOI: 10/f94xjp.

[95] "Weave Net: Network containers across any environment," Accessed: Oct. 28, 2019. [Online]. Available: https://www.weave.works/docs/net/latest/overview/.

[96] Q. Cai, M. Vuppalapati, J. Hwang, C. Kozyrakis, and R. Agarwal, "Towards μ s tail latency and terabit Ethernet: Disaggregating the host network stack," in *Proceedings of the ACM SIGCOMM 2022 Conference*, Amsterdam Netherlands: ACM, Aug. 22, 2022, pp. 767–779, ISBN: 978-1-4503-9420-8. DOI: 10.1145/3544216.3544230.

[97] Y. Sun, Q. Qu, C. Zhao, A. Krishnamurthy, H. Chang, and Y. Xiong. "TSoR: TCP Socket over RDMA Container Network for Cloud Native Computing." arXiv: 2305.10621 [cs], Accessed: May 29, 2023. [Online]. Available: http://arxiv.org/abs/2305.10621, pre-published.

[98] B. W. Barrett et al., "The Portals 4.2 Network Programming Interface," Sandia National Laboratories, Apr. 2017, p. 158.

[99] K. Taranov, F. Fischer, and T. Hoefler. "Efficient RDMA Communication Protocols." arXiv: 2212.09134 [cs], Accessed: Jan. 18, 2023. [Online]. Available: http : //arxiv.org/abs/2212.09134, pre-published.

[100] J. Bierbaum, M. Planeta, and H. Härtig, "Towards Efficient Oversubscription: On the Cost and Benefit of Event-Based Communication in MPI," in *Runtime and Operating Systems for Supercomputers*, Dallas, Texas, USA: IEEE, 2022.

[101] W. Huang, G. Santhanaraman, and Q. Gao, "Design and Implementation of High Performance MVAPICH2 (MPI2 over InfiniBand)," in *Sixth IEEE International Symposium on Cluster Computing and the Grid*, ser. CCGRID '06, 2006, p. 10.

[102] L. Liss, "The Linux SoftRoCE Driver," presented at the 13th Annual OpenFabrics Alliance Workshop (Austin, TX. U.S.A), Mar. 2017. Accessed: Aug. 11, 2020. [Online]. Available: https://youtu.be/NumH5YeVjHU?t=45.

[103] D. Garcia, P. Culley, R. Recio, J. Hilland, and B. Metzler. "A Remote Direct Memory Access Protocol Specification," Accessed: Apr. 2, 2020. [Online]. Available: https://tools.ietf.org/html/rfc5040.

[104] A. Kaufmann, T. Stamler, S. Peter, N. K. Sharma, A. Krishnamurthy, and T. Anderson, "TAS: TCP Acceleration as an OS Service," in *Proceedings of the Fourteenth EuroSys Conference 2019*, Dresden Germany: ACM, Mar. 25, 2019, pp. 1–16, ISBN: 978-1-4503-6281-8. DOI: 10/ggzhd6.

[105] T. Høiland-Jørgensen et al., "The eXpress data path: Fast programmable packet processing in the operating system kernel," in *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies*, Heraklion Greece: ACM, Dec. 4, 2018, pp. 54–66, ISBN: 978-1-4503-6080-7. DOI: 10/gh5x29.

[106] M. Su, M. Zhang, K. Chen, Z. Guo, and Y. Wu, "RFP: When RPC is Faster than Server-Bypass with RDMA," in *Proceedings of the Twelfth European Conference on Computer Systems*, ser. EuroSys '17, New York, NY, USA: Association for Computing Machinery, Apr. 23, 2017, pp. 1–15, ISBN: 978-1-4503-4938-3. DOI: 10.1145/3064176.3064189.

[107] A. Kalia, M. Kaminsky, and D. G. Andersen, "Using RDMA efficiently for key-value services," in *Proceedings of the 2014 ACM conference on SIGCOMM - SIGCOMM '14*, Chicago, Illinois, USA: ACM Press, 2014, pp. 295–306, ISBN: 978-1-4503-2836-4. DOI: 10.1145/2619239.2626299.

[108] S. Boyd-Wickizer et al., "Corey: An Operating System for Many Cores," in *8th USENIX Symposium on Operating Systems Design and Implementation*, San Diego, California, USA: USENIX Association, 2008.

[109] A. Ousterhout, J. Fried, J. Behrens, A. Belay, and H. Balakrishnan, "Shenango: Achieving High CPU Efficiency for Latency-sensitive Datacenter Workloads," in *Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation*, Boston, MA, USA, 2019.

[110] D. Firestone et al., "Azure Accelerated Networking: SmartNICs in the Public Cloud," in *15th USENIX Symposium on Networked Systems Design and Implementation*, ser. NSDI'18, Berkeley, Calif: USENIX Association, 2018, pp. 51–64, ISBN: 978-1-931971-43-0.

[111] T. Hoefler, S. Di Girolamo, K. Taranov, R. E. Grant, and R. Brightwell, "sPIN: High-performance streaming processing in the network," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '17*, Denver, Colorado: ACM Press, 2017, pp. 1–16, ISBN: 978-1-4503-5114-0. DOI: 10.1145/3126908.3126970.

[112] H. Sadok et al., "We need kernel interposition over the network dataplane," presented at the HotOS, Ann Arbor, MI, USA, 2021.

[113] L. Vilanova et al., "Slashing the disaggregation tax in heterogeneous data centers with FractOS," in *Proceedings of the Seventeenth European Conference on Computer Systems*, Rennes France: ACM, Mar. 28, 2022, pp. 352–367, ISBN: 978-1-4503-9162-7. DOI: 10.1145/3492321.3519569.

[114] I. Lesokhin et al., "Page Fault Support for Network Controllers," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, Xi'an China: ACM, Apr. 4, 2017, pp. 449–466, ISBN: 978-1-4503-4465-4. DOI: 10/ghm5x7.

[115] M. Planeta, J. Bierbaum, L. S. D. Antony, T. Hoefler, and H. Härtig, "MigrOS: Transparent Operating Systems Live Migration Support for Containerised RDMA-applications," in *USENIX ATC 2021*, Jul. 14, 2021, pp. 47–63, ISBN: 978-1-939133-23-6. [Online]. Available: https://www.usenix.org/conference/atc21/presentation/planeta.

[116] S. Chunduri, S. Parker, P. Balaji, K. Harms, and K. Kumaran, "Characterization of MPI usage on a production supercomputer," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, ser. SC '18, Dallas, Texas: IEEE Press, Jul. 26, 2019, pp. 1–15. DOI: 10.1109/SC.2018.00033.

[117] T. Miemietz, M. Planeta, V. Reusch, J. Bierbaum, M. Roitzsch, and H. Härtig, "Fast Privileged Function Calls," in *Systems for Post-Moore Architectures*, Rennes, France, 2022. [Online]. Available: https://www.barkhauseninstitut.org/fileadmin/user_upload/Publikationen/2022/202204_Miemietz_SPMA_FastCalls.pdf.

[118] P. Enberg, A. Rao, J. Crowcroft, and S. Tarkoma. "Transcending POSIX: The End of an Era?" USENIX, Accessed: Jan. 26, 2023. [Online]. Available: https://www.usenix.org/publications/loginonline/transcending-posix-end-era.

[119] V. Atlidakis, J. Andrus, R. Geambasu, D. Mitropoulos, and J. Nieh, "POSIX abstractions in modern operating systems: The old, the new, and the missing," in *Proceedings of the Eleventh European Conference on Computer Systems*, London United Kingdom: ACM, Apr. 18, 2016, pp. 1–17, ISBN: 978-1-4503-4240-7. DOI: 10.1145/2901318.2901350.

[120] Z. Yang et al., "SPDK: A Development Kit to Build High Performance Storage Applications," in *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, Dec. 2017, pp. 154–161. DOI: 10.1109/CloudCom.2017.14.

[121] Intel Corporation, "oneAPI Specification 1.2-rev-1," Specification. Accessed: Dec. 21, 2022. [Online]. Available: https://spec.oneapi.io/versions/1.2-rev-1/.

[122] "FFMK Website," FFMK, Accessed: Oct. 22, 2020. [Online]. Available: https://ffmk.tudos.org.