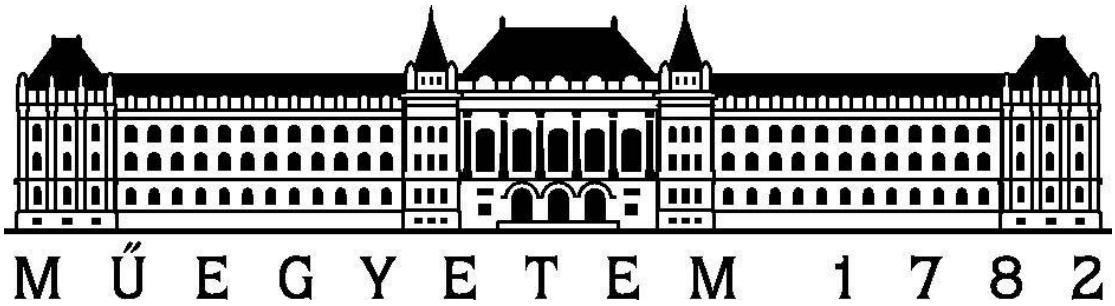


Budapest University of Technology and Economics

Faculty of Electrical Engineering and Informatics

Department of Telecommunications and Media Informatics

Dávid Balla

Minimizing the latency of the RDMA based communications

Supervisors

Markosz Maliosz Dr.

Csaba Simon Dr.

Dániel Géhberger

Budapest 2017

Table of Contents

1 Introduction

Requirements against modern IT infrastructures are high availability and low response times. Such systems usually run in a distributed way. For this reason, the networking infrastructure that interconnects the components is one of the most important building block of the system. Because, even if the components are performing well one by one, the underlying network can cause bottleneck in the whole system.

In this paper I show a solution for enhancing networking performance by using Remote Direct Memory Access (RDMA). I show how RDMA influences network latency, and how can we gain more performance with the use of subtle programming steps.

In the next Section I present the motivation of my work and provide an overview of the related literature, then in Section 3 I review the networking technologies related to my work. In the Section 4 I provide a detailed description of the RDMA approach. In Section 5 I present the results of my research, including my proposed mechanisms and the measurement results. Finally, I summarize these results.

2 Motivation

When RDMA appeared first in the world of computer networking a lot of research and development project has started to utilize its possibilities and exploiting the gains of low latency and high bandwidths supported by the technology. Since RDMA is only supported by special Network Interface Cards (NICs) and in the early 2000s the requirements of typical use cases were not so high for networking infrastructures, these projects have been soon stopped. It can be said that RDMA has appeared before the time has come for leveraging its features.

Nowadays, the demand has increased for networking infrastructures that have high availability and the ability of serving huge amount of tasks. Such environments need to support low response times and high bandwidths, a typical use case being the telecommunication networks with high Quality of Service (QoS) requirements. Today, in modern data center servers are required to handle massive amounts of data. Latency-sensitive applications like virtualized telecom and industrial IoT systems require a service with very low latencies to become cloud-native. Recent proposals that confer determinism to standard Ethernet, e.g., Time-Sensitive Networking (TSN) are not mature enough yet, and there are very few networking devices that support such features.

Today, Ethernet is the default option to solve the networking needs of the data centers. Nevertheless, as for now, Ethernet cannot support the increasing QoS requirement. The research community revisited RDMA as a known option to achieve low latency networking. The first widely used RDMA networking architecture was Infiniband, but it was designed to be deployed in dedicated networks. This was a major impediment in the wide adoption of the technology, and the most successful contender proved to be the RDMA over Converged Ethernet (RoCE), which combines the ubiquity of Ethernet and the certain advantages of RDMA.

RDMA based improvement of specific distributed applications has been actively studied in the recent years. Due to lack of space I cannot offer a detailed review of all the relevant articles, but I will highlight several contributions covering various fields. HERD is an RDMA based key-value system, which supports up to 26 million transactions per second with 5 us average latency. HERD shows how effective systems can be implemented by using RDMA [26]. As RDMA technologies has evolved, HERD has been improved as well, increasing both the CPU efficiency and the throughput performance [27]. The uRDMA project has started to implement RDMA over DPDK [28], answering need of providing RDMA over commodity hardware. FaRM shows an example how RDMA can be used to implement distributed memory, too. By replacing TCP/IP with RDMA, FaRM improved both its latency and throughput performance by

an order of a magnitude. RDMA also can be used to support low latency file transfers, by leveraging the possibility of copying data from the server's memory to the client's page cache [29]. This solution minimizes memory copies that are otherwise executed in the client interface or the operating system kernel.

The application of RDMA to speed up the networking process is a hot topic among the designers of Data Center communication. Besides providing a good review of the historical RDMA technologies, [30] also proposes a software RDMA solution that can cope with high-performance computation loads. This trend is sensed by the networking equipment manufacturers, too. A good example of a recent high-performance network adapter family is presented in [31], where the authors also presents a good background and motivation for the adoption of this technology. [32] presents a detailed overview of RDMA based intra-data center communication solution, which is deployed in the data centers of one of the leading public cloud service provider.

3 Technologies used

3.1 Traditional networking solutions

When creating computer networking systems the first thoughts are about traditional networking devices like switches, routers and cables used to connect servers or storage nodes. This is a network centric, bottom-up approach. According to this view the overwhelming part of the design is about the underlying networking architecture. Applications running in such environments are highly different by the nature of the traffic, for example storage, web, inter-process communication, etc.

Traditional local networking solutions are mostly implemented by Ethernet devices and links and build communication channels over IP. The majority of applications running over IP are using socket based communication, such as TCP or UDP sockets. An application using such communication types are relying on the main memory of the operating system for creating an anonymous buffer pool which is integral for the networking stack. Each in/outbound packets are copied via these buffers as they are traversing through the networking stack. When remote applications communicate with each other by transferring data, for every send or receive operation at least two copies are required, one for copying the data to the buffer of the transport protocol, and another for delivering the data to the application's virtual memory space as shown in Figure 1.

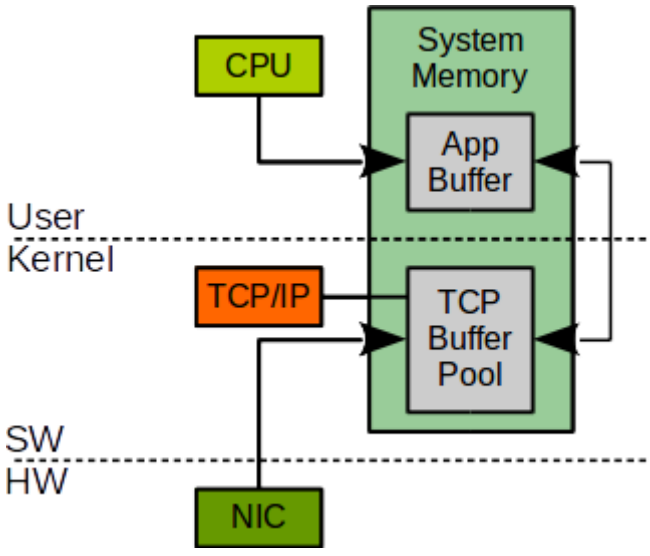


Figure 1 – Extra copying cycles using socket based communication [1]

These extra copies, induced by the transport layer, are very expensive both for latency and bandwidth, not to mention the CPU time consumed by the copy operations. This results in pure overhead spending the time with copying data through the buffers.

The load of the memory originated from the network traffic is largely dependent from the aggregated traffic that has been created by the hosted applications. As the number of processors in computers increases and the number of threads executed per core increases the offered network load can increase proportionally.

Further, each incoming packet moves from the NIC to a ring buffer and an interrupt is asserted, the driver serves the interrupt, receives the packet, and pushes it to the networking stack. The packet then traverses through the networking stack and arrives in a socket. Lastly, the data is copied from the socket buffer located in the kernel space to the memory space of the application in the user space.

3.2 Solutions applied in data centers

Traditional local area networking is predominantly based on Ethernet. Ethernet does not provide any guarantees that data is delivered, i.e. the delivery is best-effort, that can lead to packet loss typically due to congestion or corrupted packets. Higher layer protocols can handle these packet losses, taking TCP for an example. TCP is a reliable connection oriented transport protocol, which guarantees reliability by retransmitting packets if loss occurs. Combining the latency coming from the buffer copy operations happened as the data traverses the networking stack, and the latency caused by packet retransmissions can result in serious latency increase and bandwidth drops.

Data Center Bridging (DCB) is a solution for implementing lossless Ethernet by providing hardware-based bandwidth allocation to specific types of traffic. It enhances Ethernet transport by several protocols such as Priority based Flow Control (PFC), Enhanced Transmission Selection, and Congestion Notification [2][3].

3.3 Data Plane Development Kit

Data Plane Development Kit (DPDK) is a Linux Foundation Project, developed by hundreds of contributors, supported by strong leading members, and used in a growing ecosystem. DPDK runs mostly in Linux userland, it consists of libraries and drivers for fast packet processing [4]. DPDK achieves fast packet processing by controlling the NIC from user space as opposed to kernel space, by using DMA operations to move incoming traffic to application buffers, thus DPDK can avoid expensive kernel to user space copying. The User-space IO (UIO) driver is responsible for mapping the device to user space. DPDK polls the memory for receiving packets from the hardware instead of interrupt based processing, the Poll Mode Driver (PMD) implements this task. By this DPDK can achieve much faster packet processing than the classical solution [5].

3.4 Solutions for virtualized environments

3.4.1 Using software based layer 2 forwarding

There are different networking solutions for environments running virtual machines (VM) or containers. The basic solution supported by every cloud platform is using a virtual software switch that interconnects the physical network interface and the VMs (Figure 2), like LinuxBridge, part of the Linux kernel, or Open Virtual Switch (OVS). In this networking setup VMs running in the cloud on different hosts communicate via the software switches, the physical interfaces of the compute nodes and the underlaying physical network. However, if we would like to implement some kind of latency sensitive communication this solution has some drawbacks, since software based L2 forwarding consumes CPU time, and adds overhead in latency. Using OVS latencies coming from the network stack can be eliminated by using DPDK based extensions [7].

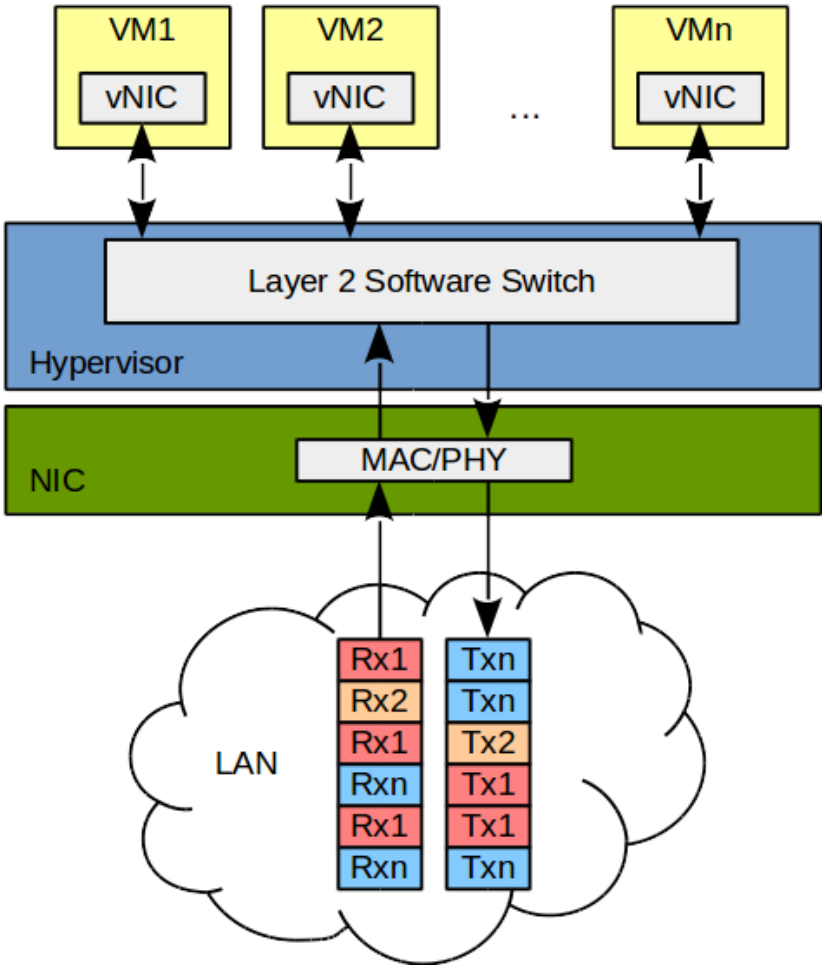


Figure 2 – Software based L2 forwarding [6]

For eliminating these overheads using software switches Intel introduced hardware based solutions for virtualization. As the number of VMs are increasing on a host the required traffic is increasing nearly proportionally. This increase in traffic has a serious impact on CPU usage, and can cause bottlenecks regarding the latency of network communication.

3.4.2 Virtual Machine Device Queues

Intel introduced Virtual Machine Device Queues (VMDQ). By using VMDQ the arriving packets are sorted to destination queues using MAC address and VLAN tag filtering, the sorter then places the packets to the receive queue belonging to the respected VM. The hypervisor's software bridge then routes the packets to the respective VMs, thus VMDQ performs the heavy lifting work by sorting the packets. Figure 3 shows this mechanism [6].

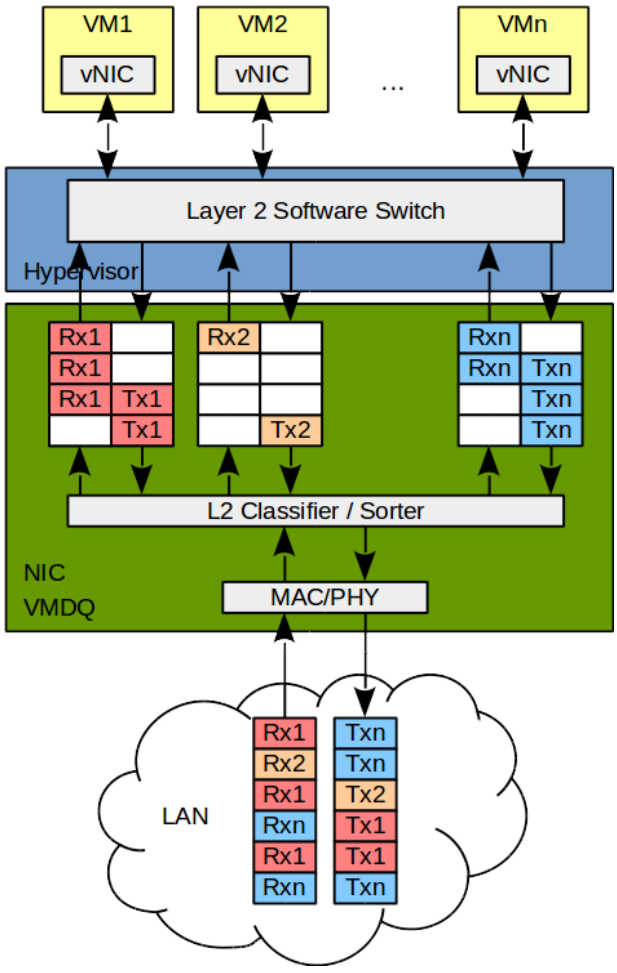


Figure 3 – VMDQ packet sorting [6]

3.4.3 Single Root I/O Virtualization

Intel later introduced further latency eliminations by presenting Single Root I/O Virtualization (SR-IOV). SRIOV allows a single PCIe (PCI Express) physical device under a single root port, to appear to be multiple separate physical devices to the host OS or the hypervisor. SR-IOV operates by introducing Physical Functions (PF) and Virtual Functions (VF). PFs are full PCIe devices that include the SR-IOV extended capability for managing and configuring SR-IOV functionality. VFs are lightweight PCIe devices that contain the resources for data transferring but they offer less configurational opportunities [8].

By using the features of an SR-IOV capable NIC one can assign Virtual Functions (VF) to the VMs using PCI passthrough. Thus, VMs can use dedicated hardware resources for network communication, and are able to bypass the hypervisor's software based L2 forwarding services (Figure 4).

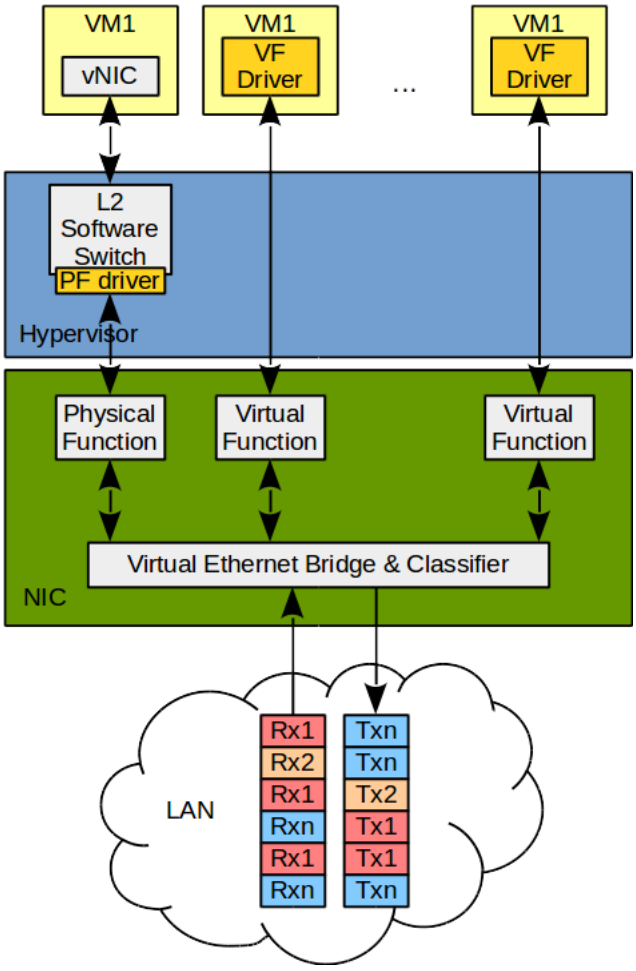


Figure 4 – SR-IOV [9]

The previous solutions are enough for eliminating overheads regarding latencies

coming from software based L2 forwarding, for eliminating the latencies coming from the usage of the networking stack DPDK based extensions or custom solutions should be used.

4 Remote Direct Memory Access

Direct Memory Access (DMA) is a feature of a device to send or receive data directly to or from the main memory, without any interaction of the computer's CPU. Remote Direct Memory Access (RDMA) is the ability of accessing memory on a remote computer without any CPU interruption on the remote system.

4.1 Differences between traditional IO and RDMA

The basic idea behind RDMA is the kernel bypass which means the elimination of extra copying cycles through the networking stack that was described before. Logically RDMA creates a direct communication channel between the buffers of the remote applications.

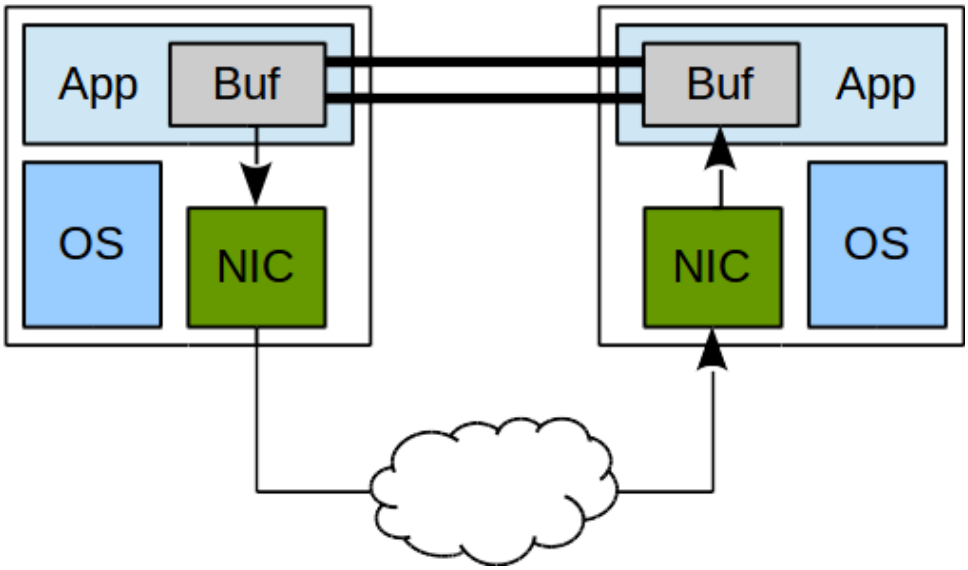


Figure 5 – RDMA creates a channel between the virtual address space of remote applications [10]

The key approach is that the architecture offers availability to applications to directly access the hardware resources. This means that the application does not rely on the operating system for transferring messages, in consequence of this, RDMA implements zero copy as well, since each inbound packets are directly copied to the virtual memory space of the application and every outbound packets to the NIC's physical buffer. The differences between the usage of traditional networking stack and RDMA is shown in Figure 6.

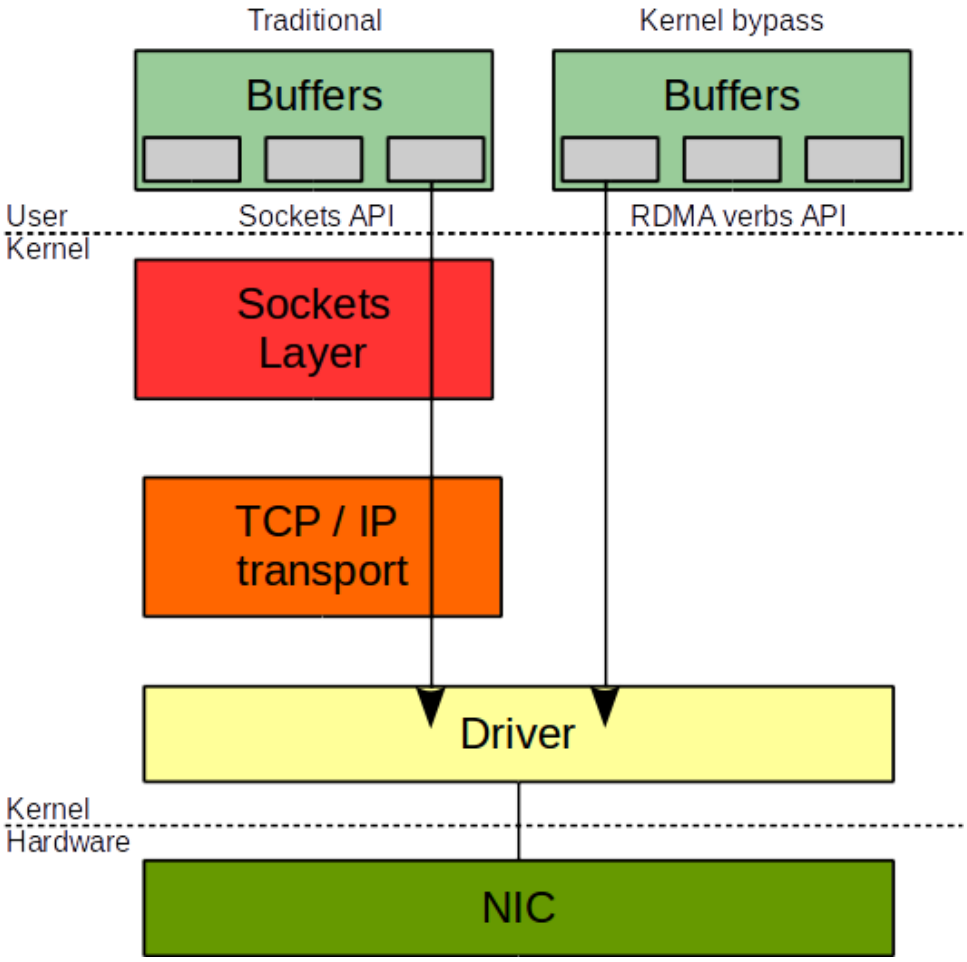


Figure 6 – Comparison of socket and RDMA based communication [11]

4.2 RDMA communication

Applications using RDMA can communicate by using queues. Three types of queues exist, Send Queue, Receive Queue and Completion Queue (SQ, RQ, CQ). SQs and RQs are always used together as Queue Pairs (QP). QPs are much like sockets in traditional TCP / UDP communication, the difference between sockets and QPs is that QPs are mapped to hardware resources, while sockets are implemented purely in software. For sending or receiving messages Work Requests (WR) should be inserted in the corresponding queues of the QPs. There are two types of WRs, Send Requests (SR) are for transferring data to the receiving side of the communication, while Receive Requests (RR) are for catching incoming messages. Inserting a SR into the SQ results in sending a message, which is fragmented to frames by the RDMA hardware. Figure 7 depicts how QPs are connected to each other [12].

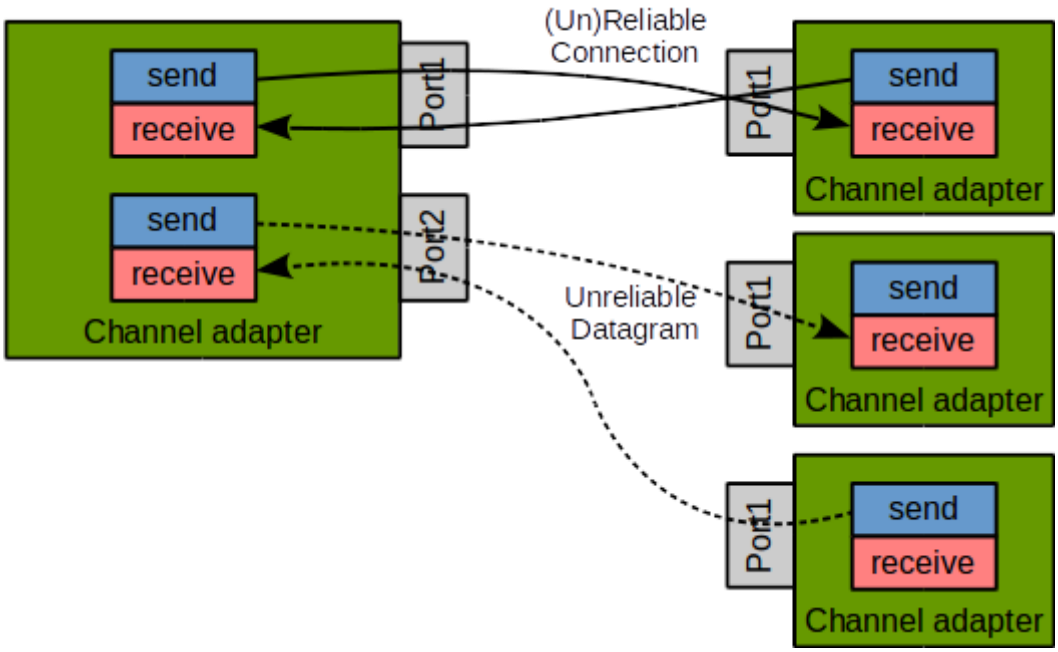


Figure 7 – RDMA communication is implemented by using QPs [12]

One can be sure that a WR has completed by gathering the Work Completions (WC) from the CQ. WCs contain data about the state of the completed WR. Work completions can be retrieved from the CQ by polling it, or by using events. On the other hand, polling the CQ consumes much more CPU resources than using events, on the other hand, using events cannot guarantee the immediate access to the WC elements [13]. The states of WRs are shown in in Figure 8.

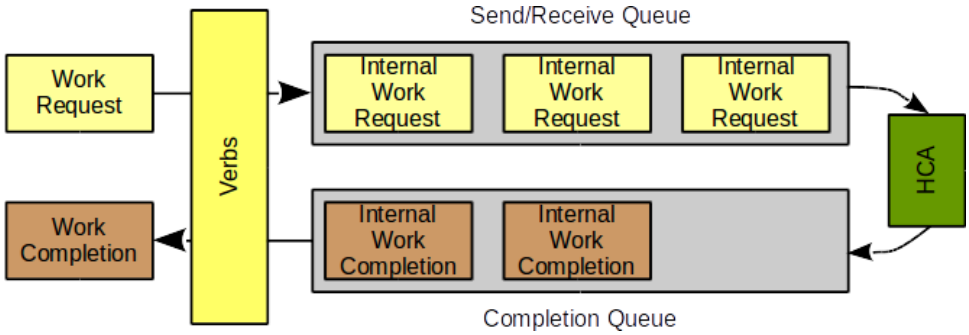


Figure 8 – States of a WR from creation to completion [12]

When creating WRs the memory region where the data will be sent from or received to should be specified by the developer of the application. In case of RR only the address of the destination buffer, and its length must be specified. SRs are a bit more complex. Three main types of operations can be used for SRs. If one uses send - receive operations, then SRs should be posted on the side of the communication where the data will be sent from, and RR are posted on the receiver side. In this case the checking of completion queues is mandatory on both sides, for making sure that the data transfer

has ended. However, RDMA supports WRITE and READ operations as well. When using RDMA WRITE, the sender side is able to directly write data into the virtual memory region of the receiver application. This permits to eliminate the CQ checks on the receiver side. RDMA READ also happens in the background on the receiver side. RDMA READ can be used if the sender side would like to read from the memory of the receiver application. For being able to directly write to or read from the remote memory, the application should be aware of the starting address and the length of the remote buffer. Figure 9. depicts the differences between send – receive (a) and RDMA WRITE (b) operations.

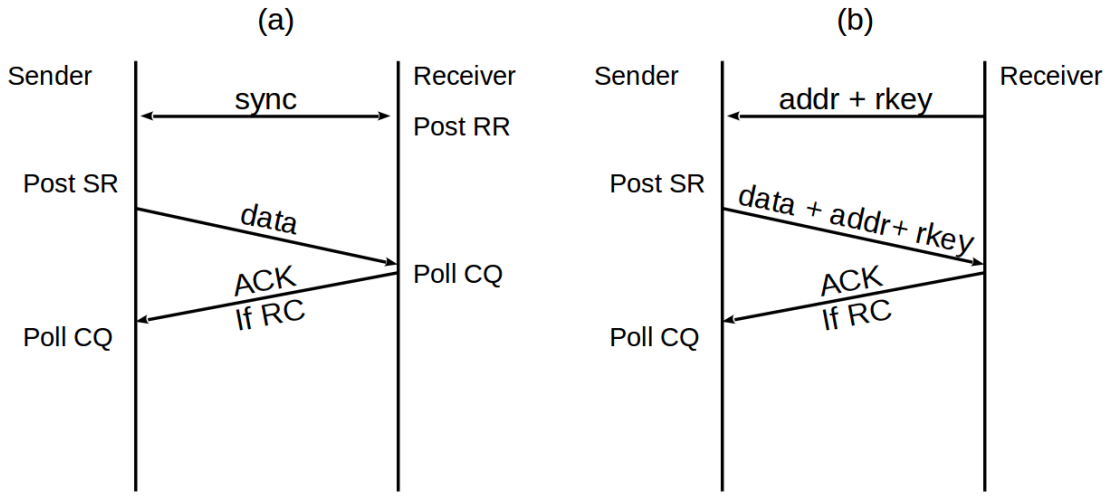


Figure 9 – Send – Receive operations compared to RDMA WRITE [14]

RDMA defines three types of connection modes, Unreliable Datagram (UD), Unreliable Connection (UC), Reliable Connection (RC).

Using RDMA UD is much like UDP in socket transport. UD is a connectionless transport type, a QP can send to or receive from messages from every other QP in the network. This solution scales the best, since there is no need to create as much QPs as much communicating parties are in the fabric. However, UD has its drawbacks as there is no guarantee that the packets are received in order or not got corrupted. Further, the maximum message size must equal to the Maximum Transmission Unit (MTU) of the transport path. In case of UD only send-receive operations can be used. Since UD is connectionless, the destination should be set for every message, this can be done by creating Address Handles (AH) and passing it to the corresponding SRs. UD is a good choice if the fabric size is large, and the reliability by the fabric is not a requirement.

In case of using UC, a QP is connected to exactly one UC QP in an unreliable way. UC does not guarantee the arriving of packets in order, and if a packet is corrupted there will be no retransmission. Corrupted and out of sequence packets are silently dropped.

If a frame is dropped, the whole message is dropped. This can lead to serious latency problems. The supported message size is up to 2 GB. UC enables the usage of RDMA WRITE besides send - receive operations. UC QPs can be used if the fabric size is not oversized and the reliability of the connection is not a requirement.

One RC QP is connected to exactly one RC QP in a reliable way. RC QPs guarantee that the messages are received at the remote side at most once and in order, without any corruption. RDMA RC uses selective acknowledgements, for properly received packets the receiver side answers with an ACK message, but if the sequence number of the arriving packet number is out of order, or the packet is corrupted the receiver side sends a NACK message to the sender side. For a NACK message the sender will retransmit the corresponding packets. One can set timeout values for waiting ACKs / NACKs to be received. In this case the maximum supported message size is up to 2 GB. RC QPs support RDMA READ operations in addition to the ones supported by UC. RC QPs are in use if the fabric size is not too large and there is a need for reliable data transfer. Table 1. summarizes the capabilities of each RDMA connection types.

Opcode	UD	UC	RC
Opcode: SEND (w/o immediate)	Supported	Supported	Supported
Opcode: RDMA Write (w/o immediate)	Not supported	Supported	Supported
Opcode: RDMA Read	Not supported	Not supported	Supported
Opcode: Atomic operations	Not supported	Not supported	Supported
Connection type	Datagram (One to any/many)	Connected (one to only one)	Connected (one to only one)
Maximum message size	Maximum path MTU	2 GB	2 GB
Multicast	supported	Not supported	Not supported

Table 1. RDMA connection types and capabilities [15][19]

4.2.1 Programming interface for RDMA

For creating applications implementing RDMA communication one should use the verbs API or ibverbs, which is a unified library for developing applications using RDMA. The verbs API can be used for IPC, storage and network communication as well.

4.3 RDMA transport technologies

There are three implementations of RDMA, using the same programming interface, these are Internet Wide Area RDMA Protocol (iWARP), Infiniband and RDMA over Converged Ethernet (RoCE).

4.3.1 InfiniBand

InfiniBand (IB) is a computing networking communications standard used in High Performance Computing (HPC) environments, developed by the InfiniBand Trade Association (IBTA). IB provides the combination of high bandwidth and low latency, and also implements RDMA.

The smallest IB architecture (IBA) is the subnet, built by Channel Adapters (CA) connected to switches (SW). Software can access Host CAs (HCA), and send data to Target CAs (TCA). In every IB subnet, at least one Subnet Manager (SM) should present in the subnet. IB communication is forwarded by Local Identifiers (LID) within a subnet [12].

SM configures the local subnet, including assigning LIDs to devices, discovering the network topology, managing forwarding tables of the SWs, and monitoring the changes occurred in the subnet. [InfiniBand Essentials Every HPC Expert Must Know] There can be multiple SMs in a subnet, while only one has an active state. Non-active SMs are keeping the copies of the information of the active one. If an active SM goes down, one of the inactive SMs takes its place.

IB uses Global Identifiers (GID) for packet forwarding over subnets. Traffic is forwarded between subnets via IB routers. An IB router reads the Global Routing Header (GRH) of the incoming packets containing GID and LID of the target, and rebuilds these packets with the proper LID on the next subnet [16].

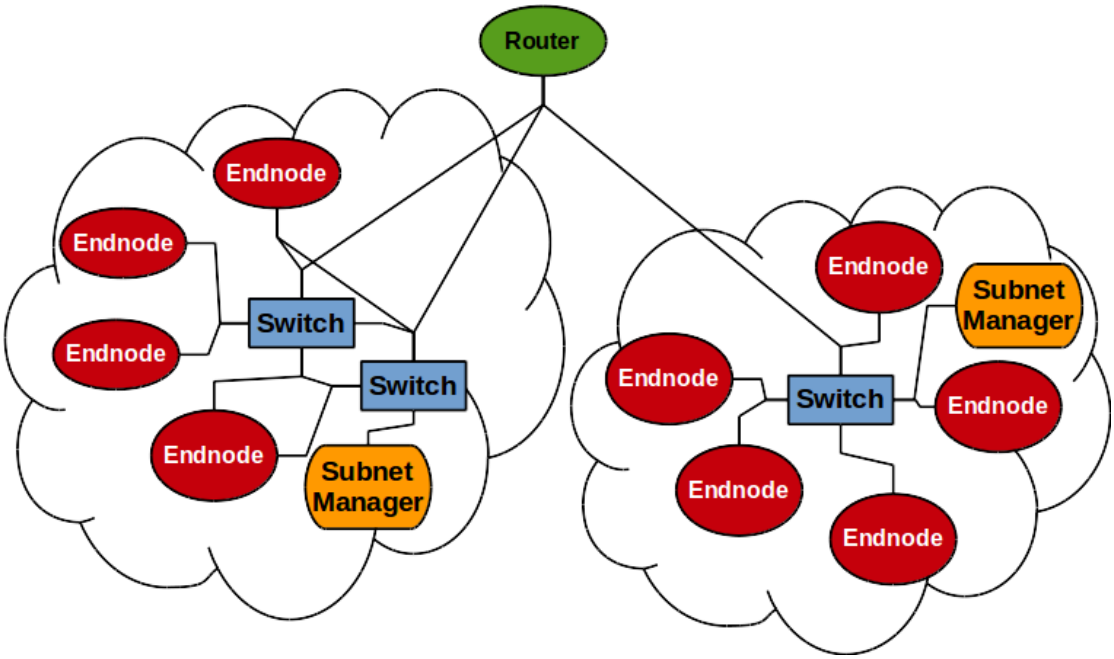


Figure 10 – Components of an IB fabric[16]

IB also enables building IP connections between hosts, using IP over IB. Using IP over IB does not require any modification of packets used by IP or the layers built on top of IP. IP packets are encapsulated in IB frames, as shown in Figuere X. The only difference between IP over Ethernet is the payload size, since IB defines different MTU values than Ethernet. The size of an IB packet using IP over IB can be the MTU - 4 bytes due to the IB frame of which payload field starts with a four bytes long header [17].

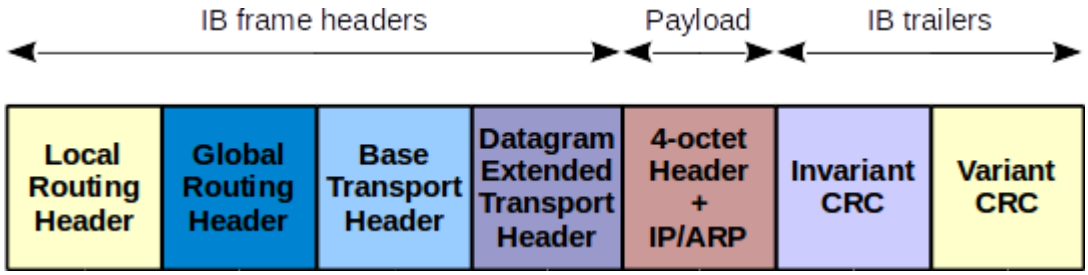


Figure 11 – IB frame containing IP packet [17]

The differences between IP over IB and IP over Ethernet can be seen in Figure 12, it also shows that no modifications are required in the upper layers.

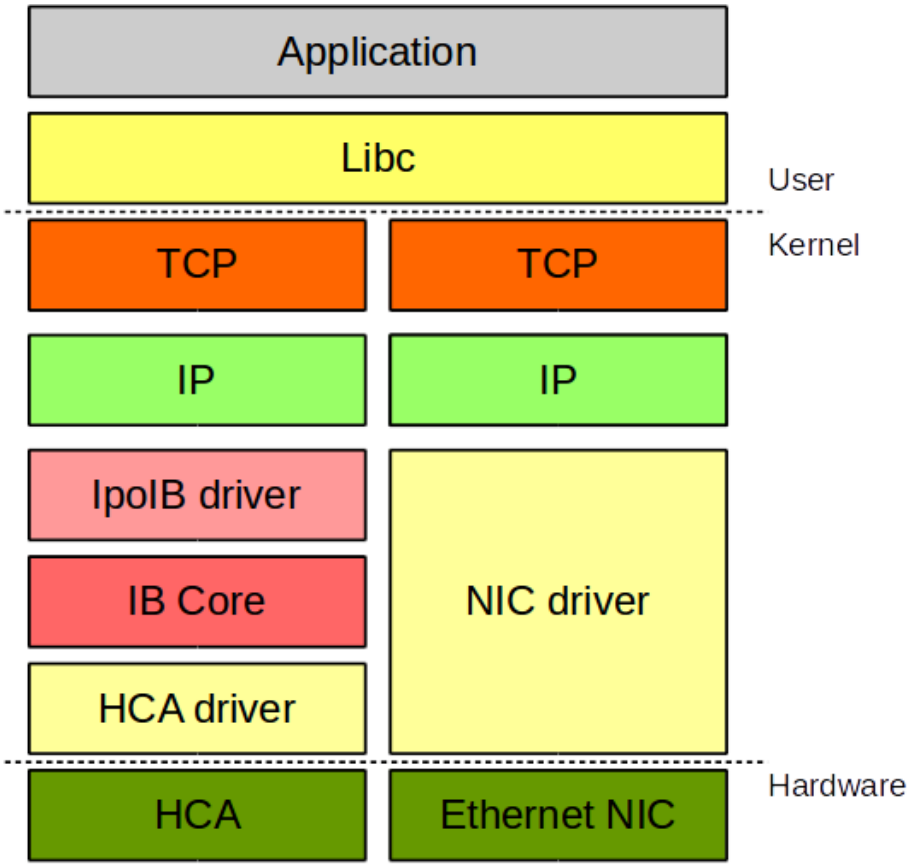


Figure 12 – IP over IB stack compared to IP over Ethernet [18]

The IB chicken and egg problem. QP communication requires knowing the needed data of the remote QPs. Sharing data related to accessing the remote QP must be done out of band since QPs cannot communicate without knowing the required data [19]. To handle this, IP over IB provides out of band channels like TCP or UDP connections.

4.3.2 RoCE

RDMA over Converged Ethernet (RoCE) implements RDMA and provides the opportunity for building flexible heterogeneous fabrics. Since RoCE uses Ethernet for link layer RDMA traffic can traverse through ubiquitous Ethernet networks. RoCE implements two versions, RoCE v1 and RoCE v2.

RoCE v1

RoCE v1 encapsulates InfiniBand traffic in Ethernet frames. This version enables RDMA communication between hosts in the same subnet. RoCE v1 uses ethertype 0x8915, this also means that the frame length is limited to standard Ethernet MTU size. The modification done for RoCE v1 compared to IB is depicted in Figure 13.

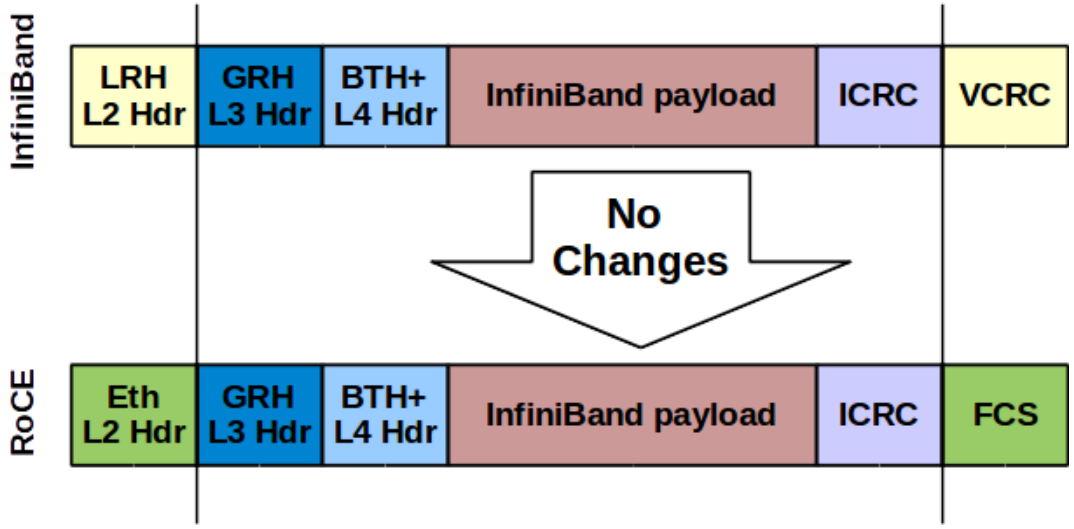


Figure 13 – Differences between IB and RoCE frames [20]

RoCE v2

RoCE v2 or Rouatable RoCE (RRoCE) overcomes the problem of v1 being bounded to a single broadcast domain. RoCE v2 encapsulates IB payload into UDP packets and enables routing RDMA traffic across IPv4 and IPv6 subnets. Differences between RoCE v1 and v2 can be seen in Figure 14.

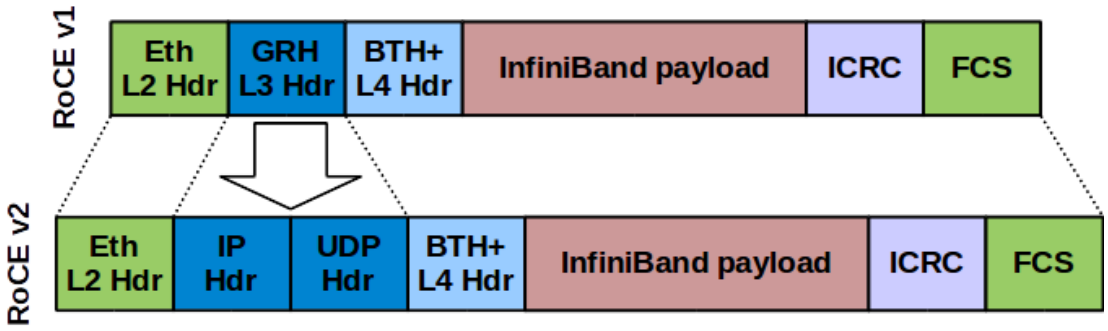


Figure 14 – Differences between RoCE v1 and v2 frames [21]

4.3.3 iWarp

iWarp builds RDMA on top of TCP/IP. iWarp leverages the advantages of this ubiquitous stack. Without RDMA the TCP/IP stack is implemented by software and puts significant load on the CPU. As NICs are evolving and bandwidth capabilities are getting higher the usage of CPU is increasing proportionally. For eliminating the overwhelmed usage of CPU iWarp introduced an extension with which it can enable Ethernet to offload transport processing from CPU to a specialized hardware, which

can save 40% of CPU utilization on networking tasks. Transport offload can be implemented by a standalone transport offload engine on the network interface. By moving transport processing to the NIC also offloads the system memory from intermediate TCP/IP protocol stack copies by moving these operations to hardware space [22].

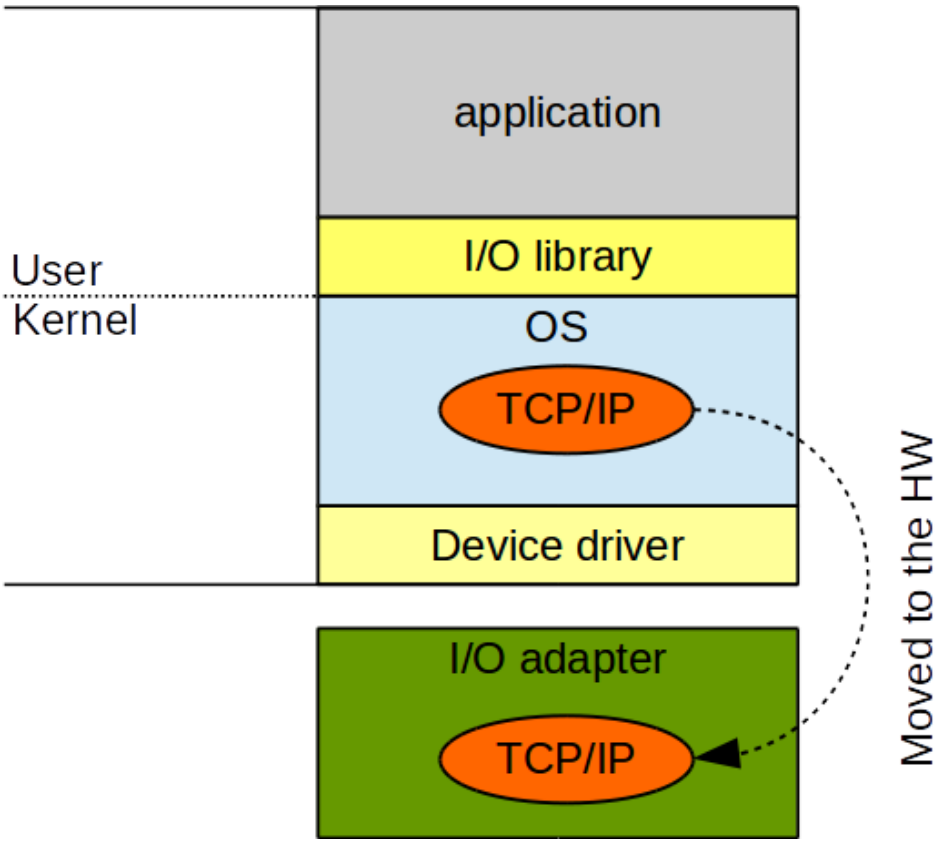


Figure 15 – Moving TCP from the kernel to the hardware [22]

Implementing RDMA over TCP enables packet forwarding through traditional networks without any hardware based support. On the other hand the implementation only enables RDMA RC communication, since TCP implements a connection oriented and reliable type of connection.

4.4 Technology comparison

The DPDK project provides an alternative to RDMA that can be used with commodity hardware [23]. DPDK similarly to RDMA implements kernel bypassing by enabling managing the hardware resources of the NIC from user space. Both solutions exploit the advantages of zero-copy and kernel bypassing for earning gains in networking performance. In contrast with RDMA, DPDK managed interfaces cannot use the networking stack, since the hardware is taken away from the kernel. In case of InfiniBand the IP over IB driver implements the necessary environment for being able

to use the networking stack, while RoCE and iWarp can directly use the networking stack supported by the kernel. Therefore, DPDK managed NICs cannot be used for tasks requiring the kernel's network stack, unless implementing it on top of a DPDK application [5]. Using RDMA based solutions gives the ease of using the hardware for low latency tasks which requires the bypassing of the kernel, at the same time with applications running that need to use the kernel's networking stack.

5 Measurement based latency evaluation

5.1 Measurement environment and benchmarking tool

I have created a test environment for measuring and comparing the latencies of traditional socket based and RDMA based communications. The two most important performance metrics of networking capacity are throughput (bandwidth) and latency. During my work I focused on evaluating and optimizing the latency of RDMA based communication. The main reason for this is that while achievable throughput is specified by the NIC vendors, the networking latency is not. Moreover, the throughput is not only depends on the NIC itself, but it is also influenced by the hardware environment (e.g., CPU power, disk and bus I/O capacity) it is deployed to. It is true that throughput may be influence by the virtualization environment of the datacenter, too, but that is an operational aspect that was out of the scope of my research. When considering latency, it is also influenced by the infrastructure environment. The difference compared to throughput is that it is not the performance of the host machine that counts, rather the external networking environment (switches, routers, interfering traffic).

In order to give a non-biased result, I used a simple, interference-free measurement scenario, my results being able to serve as benchmark for later analysis. Measurement based analysis might be biased by vendor-specific implementation details. That is the reason I used NICs from the same vendor. For the measurements I have used Mellanox ConnectX-3 IB and ConnectX-4 RoCE QDR NICs. Connectx-4 NICs have been communicating through a switch, while Connectx-3 NICs have been connected directly. Still, the measured latencies on the two setups can be compared, because the switch was previously tested and the switching latency established by measurements is 625 nanoseconds. Note that the end to end latencies measured with the RoCE NICs were above 3 microseconds, thus the sub-microsecond switching delay did not dominate the other delay components.

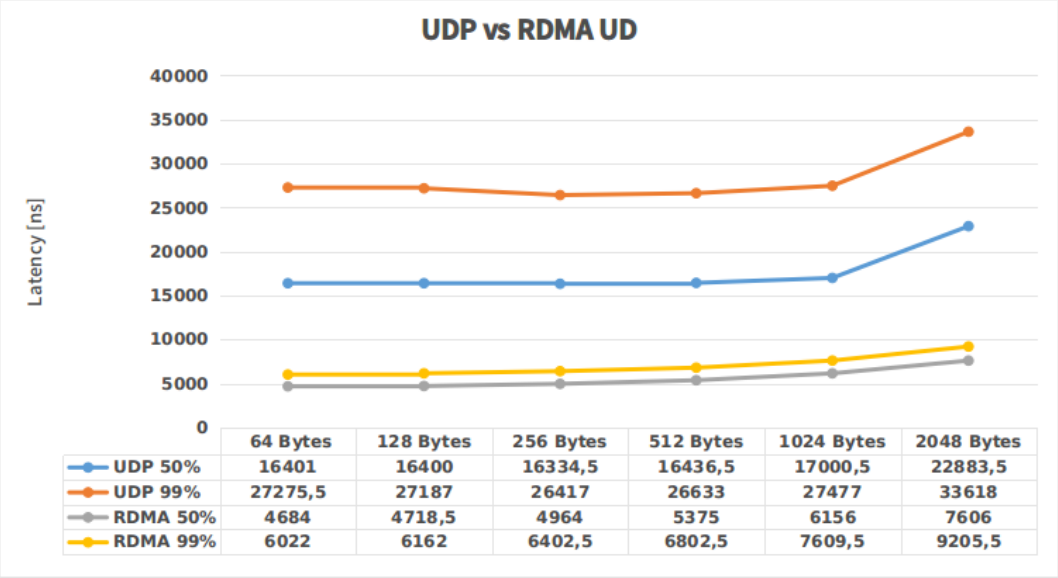
I have implemented an application that measures round trip times (RTT) for being able to compare the latencies of different types of network communications, and supports performing measurements with variable length of data. I considered it important to develop an own measurement tool instead of using third party utilities, for having full control over the application. I have performed measurements by pinning the application to exactly one CPU core avoiding the scheduling of the application to other cores and preventing further overhead latencies being added to the results.

In a local network, latencies of both RDMA and socket communication are relatively

low, in the range of microseconds to tens of microseconds. For this reason, the usage of a high-resolution clock is required. Using the Time Stamp Counter (TSC) of the CPU is an appropriate choice for measuring communication latencies in such networking architectures. TSC is a high-resolution counter inside the CPU which counts CPU cycles from which the duration of a set of instructions can be determined. TSC can be read through an assembler instruction, so the overhead is lower than calling any other pre-implemented standard functions for time measurement. When executing TSC-related instructions we should bear in mind that most of the modern CPUs enable out of order execution. Hence a dedicated instruction, called Read TSC and Processor ID (RDTSCP) should be used for reading the TSC, because RDTSCP disables out of order execution in the CPU. If RDTSCP is not supported by the processor, then RDTSC can be used instead. Nevertheless, since it is not a serializing instruction, it should be used with a previously executed serializing instruction, such as CPUID. This latter avoids the out of order instruction execution of the RDTSC instruction [24].

5.2 Differences between latencies of RDMA and socket based I/O

My goal was to measure the latency gap between RDMA and traditional socket based communication. I have compared the latency values of UDP to its RDMA based counterpart, RDMA UD. I chose UDP instead of TCP, since UDP communication does not require acknowledgement messages causing overheads in latency. Figure 16 shows the differences between using UDP and RDMA UD. The differences are coming from the copies through the networking stack; according to the values using RDMA can save at least two third of the UDP latency.



	64 Bytes	128 Bytes	256 Bytes	512 Bytes	1024 Bytes	2048 Bytes
UDP 50%	16401	16400	16334,5	16436,5	17000,5	22883,5
UDP 99%	27275,5	27187	26417	26633	27477	33618
RDMA 50%	4684	4718,5	4964	5375	6156	7606
RDMA 99%	6022	6162	6402,5	6802,5	7609,5	9205,5

Figure 16 – Latency differences between RDMA UD and UDP, showing the 50th and 99th percentiles of the latency series

I have performed measurements for examining what happens if the payload of L2 exceeds the MTU and also examining the differences of UDP (the best performing with sockets because of no ACKs) and RDMA RC (expected to perform with highest latency because of ACKs). In case of UDP we can see a definite jump in the data series (Figure 17). The reason for the jump is the software based implementation of the networking stack which implements packet fragmentation (MTU is 1500 for UDP and 1024 for RDMA). RDMA NICs are designed for performing packet fragmentation in hardware if using RC / UC QPs, thus we see only a slight increase in latency.

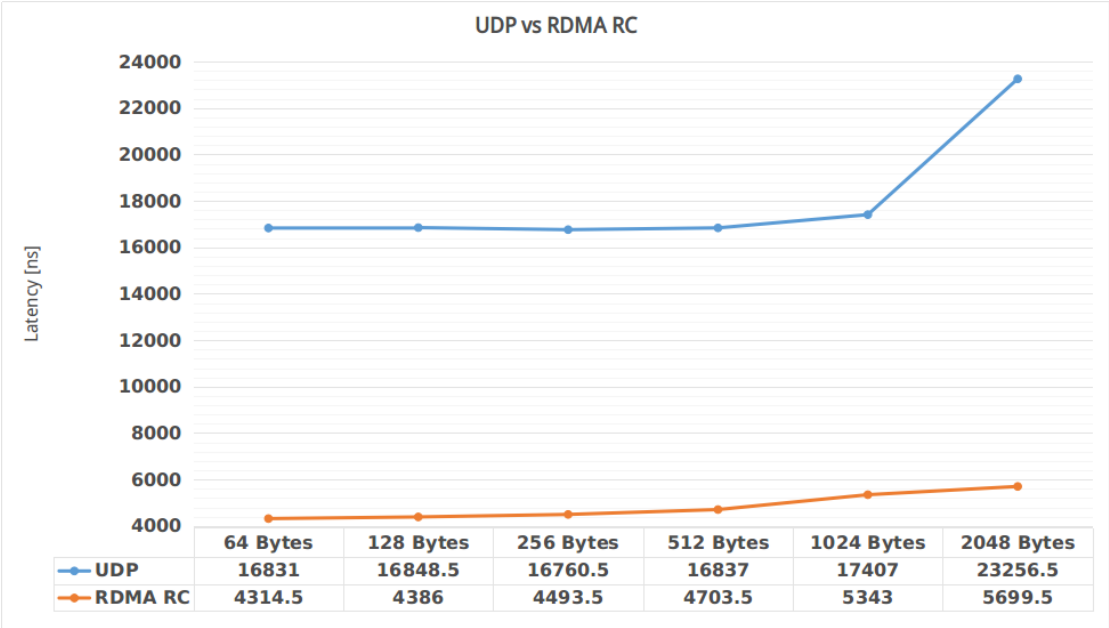
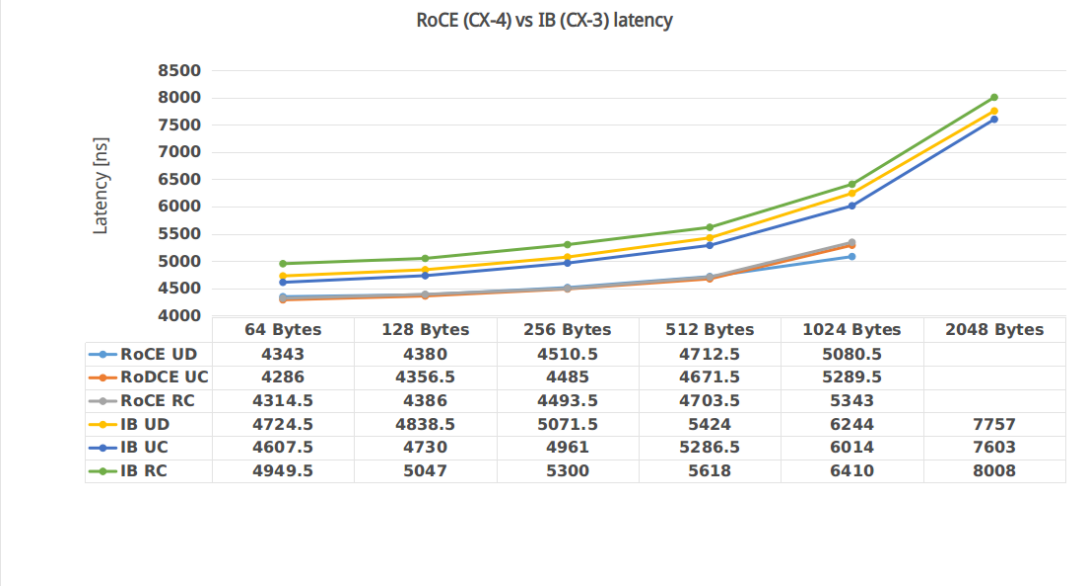


Figure 17 – Latency for different packet sizes

5.3 Latency differences between RDMA communication modes

I have also measured the differences between RDMA communication modes both with IB and RoCE cards. I have performed measurements without exceeding the MTU of the link-layers supported by the different technologies. As shown in Figure 18 RoCE performs slightly better than IB, which can be explained by the fact that ConnectX-3 IB cards are not as recent as CX-4 RoCE cards. Examining the latency values, we can observe only small differences between the different communication modes, see Figure 18. This means that RoCE offers reliable communication at the speed of unreliable connection, which is an important feature in standard Ethernet networks.



	64 Bytes	128 Bytes	256 Bytes	512 Bytes	1024 Bytes	2048 Bytes
RoCE UD	4343	4380	4510.5	4712.5	5080.5	
RoDCE UC	4286	4356.5	4485	4671.5	5289.5	
RoCE RC	4314.5	4386	4493.5	4703.5	5343	
IB UD	4724.5	4838.5	5071.5	5424	6244	7757
IB UC	4607.5	4730	4961	5286.5	6014	7603
IB RC	4949.5	5047	5300	5618	6410	8008

Figure 18 – Differences between RoCE and IB latencies

5.4 Mechanisms to enhance the performance of RDMA

In my experiments presented so far, I focused on the performance analysis of the RDMA networking solution, using the default configuration options, which already offered an impressive performance compared to standard Ethernet based solutions. However, when I analyzed the details of the communication logs, I realized that there are several possibilities to further improve the latency. In what follows I propose several mechanisms that I found effective, and also discuss their advantages.

5.4.1 Using aggregated posts for RRs

One can post RRs to the RQ by calling ibv_post_recv verb. This verb requires passing a QP and a RR for parameter, and puts the RR into the RQ of the QP. One can call this verb repeatedly for posting multiple RRs, which can result in higher latency values due to the repeated function calls. Posting multiple RRs is also feasible by calling ibv_post_recv. The structure of RR implements a linked list element, hence by linking RR elements after each other we can post more than one RR at a time, which eliminates repeated function calls. Figure 19 shows that aggregated send does not have a huge impact on lower percentiles, but on Figure 20 we can see that repeated function calls can result higher outliers in latency. In the followings I used the **aggregated posts for RRs** in all measurements.

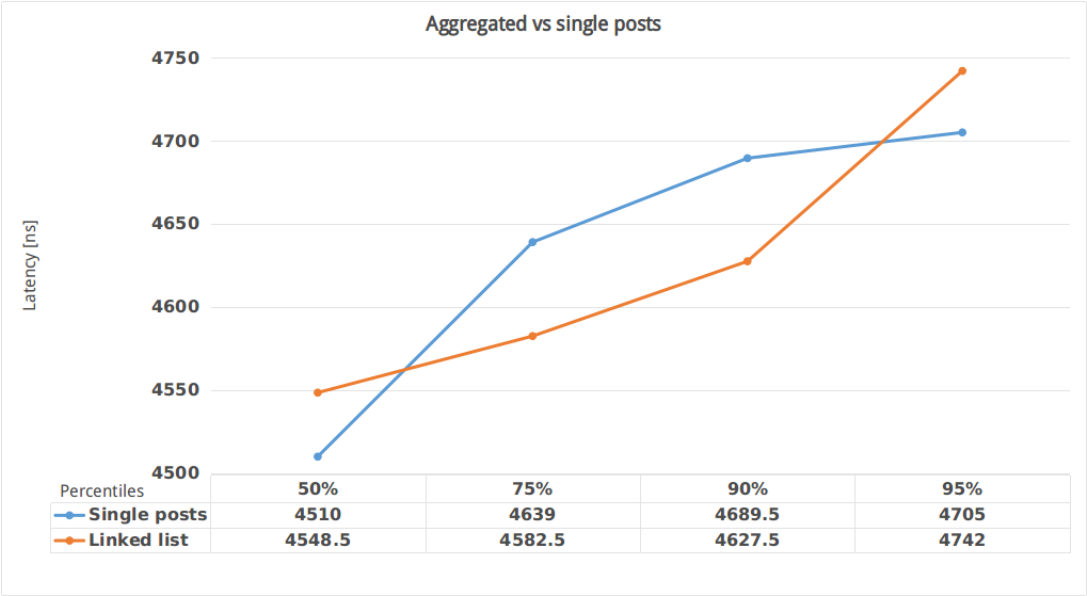


Figure 19 – Aggregated RR posting cannot influence spectacularly the latency values when talking about lower percentiles

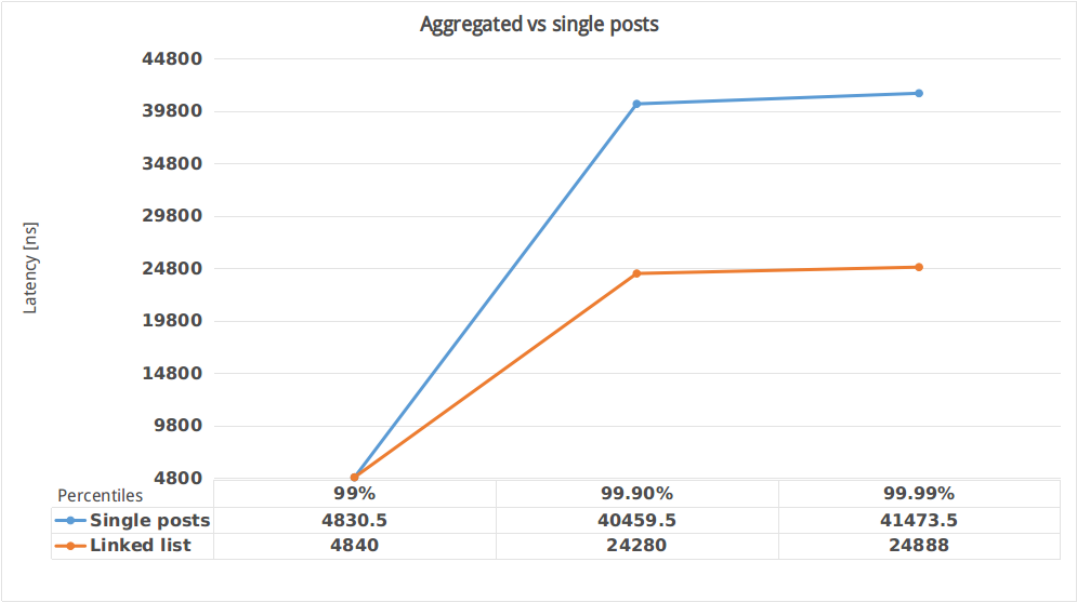


Figure 20 – Repeated function calls can result in high outliers for higher percentiles

5.4.2 Using inlined sends

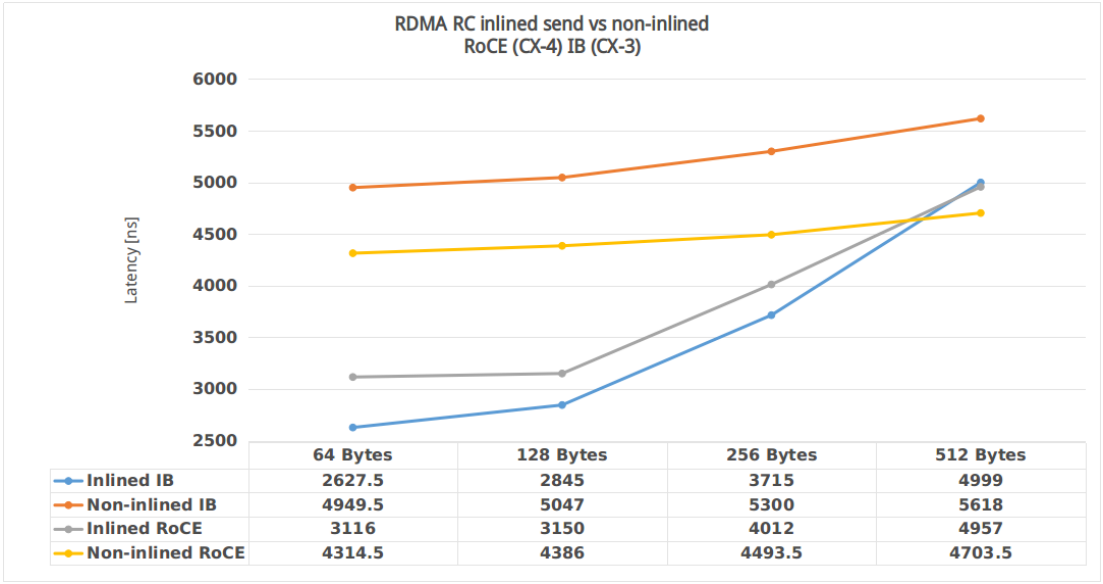
For understanding inlined data and inlined sends first we need to have an overview of SRs and the mechanism of posting them to the SQ. As it was discussed in 4.1 one can post SRs for sending data to the remote side. For sending data over RDMA the user should first register an un-swappable memory region where the data will be sent from. A SR should contain the starting address of the data within the registered memory region, and the length of the data to send. The RDMA hardware reads the posted SR for

getting the address and the length of the data, then turns to the memory for retrieving it. In contrast when using inlined sends the hardware does not need to look back to the physical memory because the data is nested – inlined – into the SR. This gives the ease to the user for not having to use registered memory, and also provides lower latencies by leaving out the reading of data from the physical memory. However, inlined sends have their disadvantages as well. One can only send a limited amount of data inlined. The maximum size of the inlined data depends not only on the hardware but the type of the connection mode. In our case the maximum sizes of the inlined data are shown in Table 2.

	UD	RC / UC
RoCE	956 Bytes	828 Bytes
IB	884 Bytes	912 Bytes

Table 2 – Maximum size of data to send inline

Figure 21 shows the changes in latency due to the application of SRs using inlined data. Using inlined sends can save about 1.2 - 2.3 us in RDMA communication. It also can be seen in Figure 21 that in case of RoCE over 256 bytes the latency values are better for non-inlined data. Using inlined send is a good choice for sending messages shorter than about 400 bytes, according to the crossing point in the figure. The measurement results are shown in the following sections has been generated by using **inlined send** operations.



	64 Bytes	128 Bytes	256 Bytes	512 Bytes
Inlined IB	2627.5	2845	3715	4999
Non-inlined IB	4949.5	5047	5300	5618
Inlined RoCE	3116	3150	4012	4957
Non-inlined RoCE	4314.5	4386	4493.5	4703.5

Figure 21 – Changes in RDMA performance by using inlined send

5.4.3 RDMA WRITE compared to send – receive mode

As it was discussed in chapter 4.1 sending data through an RDMA network can be done by using send and receive requests or using one-sided operations by RDMA WRITE. The first case requires checking the CQs on both the sender and the receiver side, while RDMA WRITE only requires CQ polling on the sender side. For general use RDMA WRITE is used for sending large data. One example for using RDMA WRITE is writing 2 GB to the memory of the receiver and notifying the receiver side by posting an empty send request to a pre-posted RR of the receiver. After the receiver gets the WC about the receive request it can be sure that all the data has arrived. Figure 22 shows this mechanism for easier understanding.

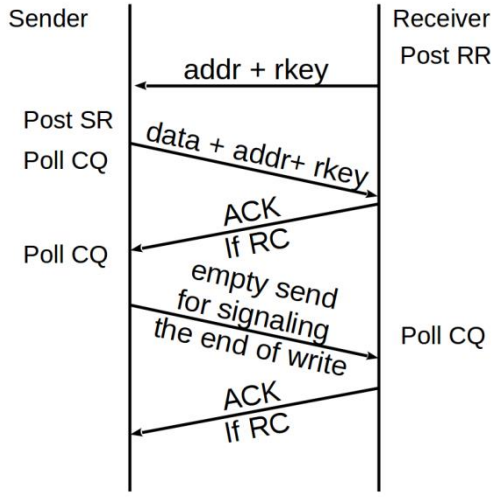


Figure 22 – RDMA WRITE with a signaling send

RDMA WRITE can be used without any signaling send request, that indicates the end of the WRITE. To achieve this, a specific message format can be used, since the incoming messages are being written to the memory using DMA. DMA is not an atomic operation therefore corrupt data can be read from the memory. Using specialized message structures can help deciding if the data has successfully written into the memory. A simple implementation for such message structure can be seen in Figure 23. The first part of the message is the length of the payload, the last n bytes are used as a trailer defining a pattern. If the trailer is correct, then the whole packet has arrived successfully.

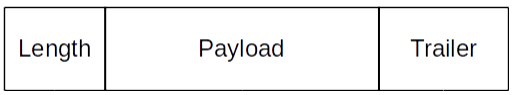


Figure 23 – Message format for using RDMA write without the use signaling sends

In Figure 24 we can see that using RDMA WRITE instead of send – receive does not have a heavy impact on the majority of the results, while Figure 25 clearly shows that

WRITE can influence the values of the higher percentiles of the data series, but this is not a spectacular difference.

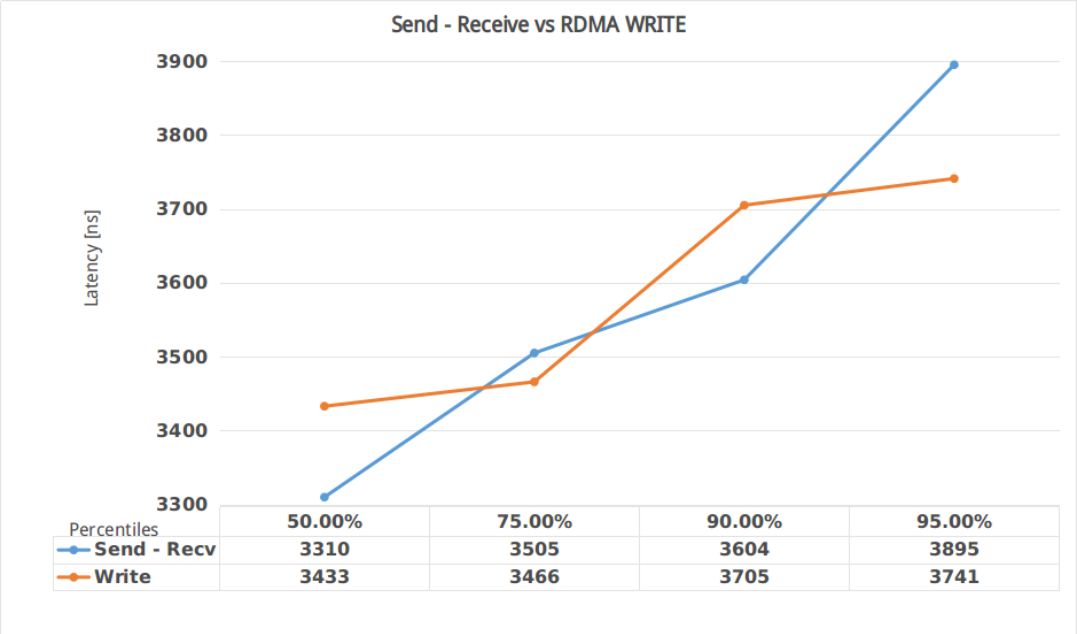


Figure 24 – Write has no influence on the majority of the latency values

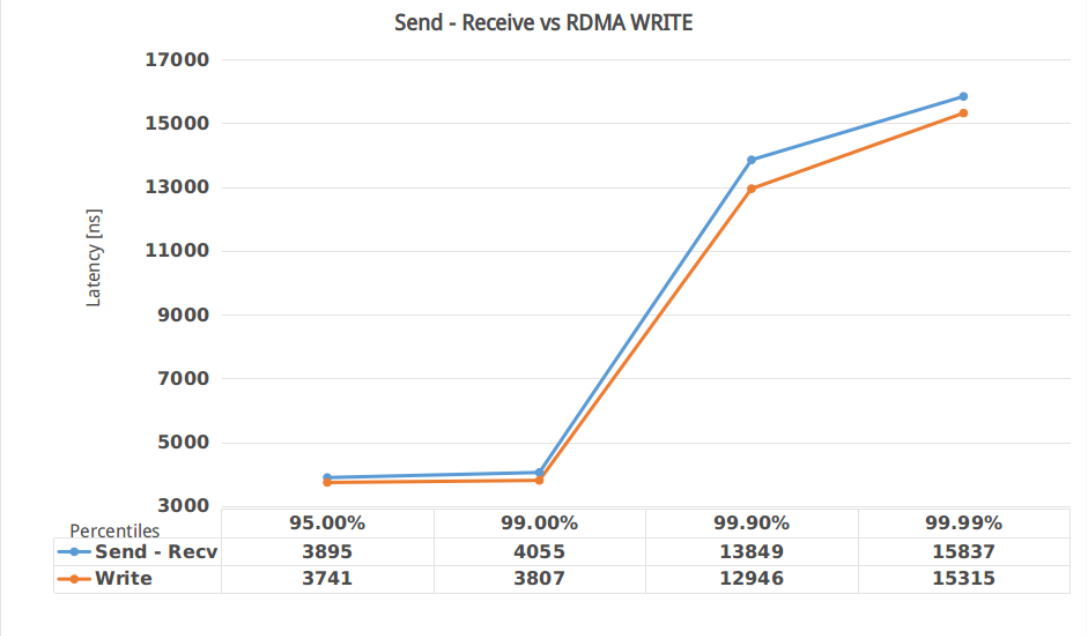


Figure 25 – The impact of using RDMA WRITE on high percentiles of the latency values

5.4.4 Introducing RDMA WRITE-like send-receive

Using RDMA WRITE and leveraging the gains of single sided polling is only available when using connected QPs. Following the method in the previous section (checking the memory for incoming messages and examining the message trailers) for making sure a

message has arrived, we can eliminate checking the CQ on the receiver side after the arrival of every single message. However, with this method -- because of the RRs posted on the receiver side -- we must retrieve WCs by polling the CQ (or receiving events about completions). This method eliminates many individual calls on the CQ after each incoming message. Measuring with this method can result in such RTT series, where the high percentiles are showing huge latency values because of the rare WC retrievals which consumes much CPU time than the ones for getting a single WC. For implementing this behavior the receiver side needs to post a huge amount of RRs to its RQ, and should poll on the memory region that is used for receiving data. If a new packet is found in the memory a RR must have been used. After polling the number of messages from the memory that equals to the number of posted RRs all the WCs should be retrieved from the CQ. This is shown in figure 26.

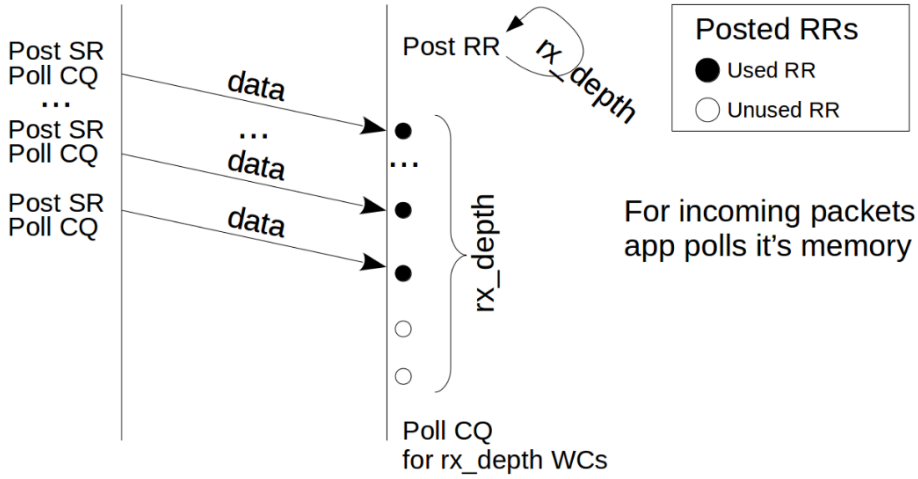


Figure 26 – implementing RDMA WRITE-like operation by using UD

I have been performing measurements with this solution, Figure 27 shows that elimination of WC retrieving from the CQ on the receiver side decreases the lower percentiles of the latency values when using UD QPs, while Figure 28 shows the outliers for the higher percentiles resulted by aggregated WC retrievals. This solution is a good choice if we want to decrease the average latency for most of the transfers, up to 99%, however the worst case latency is higher.

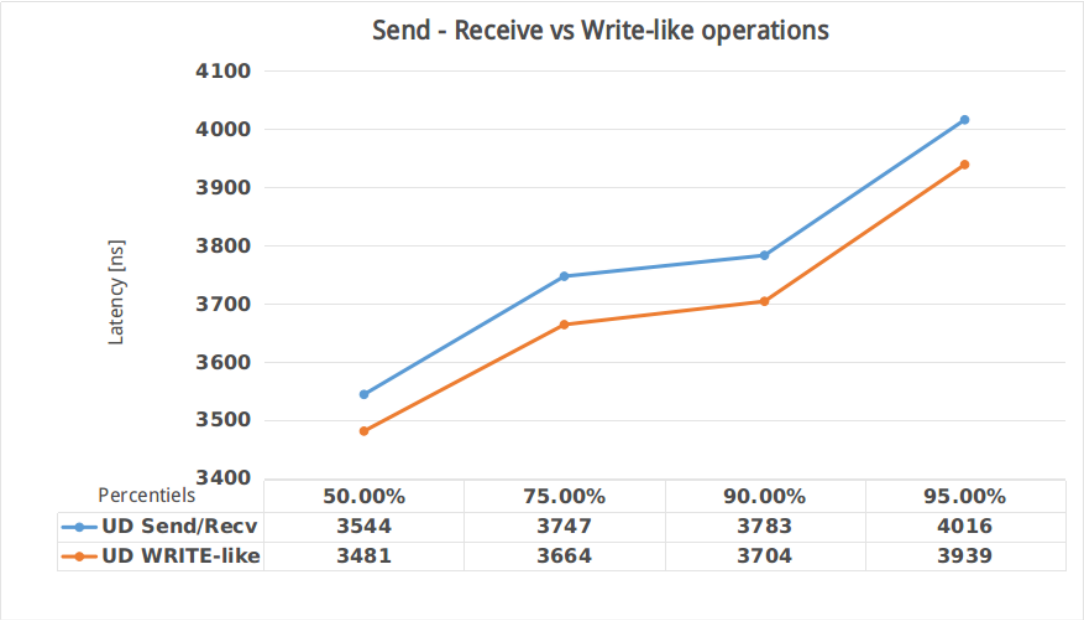


Figure 27 - Using WRITE-like send-receive performs slightly better for the majority of time

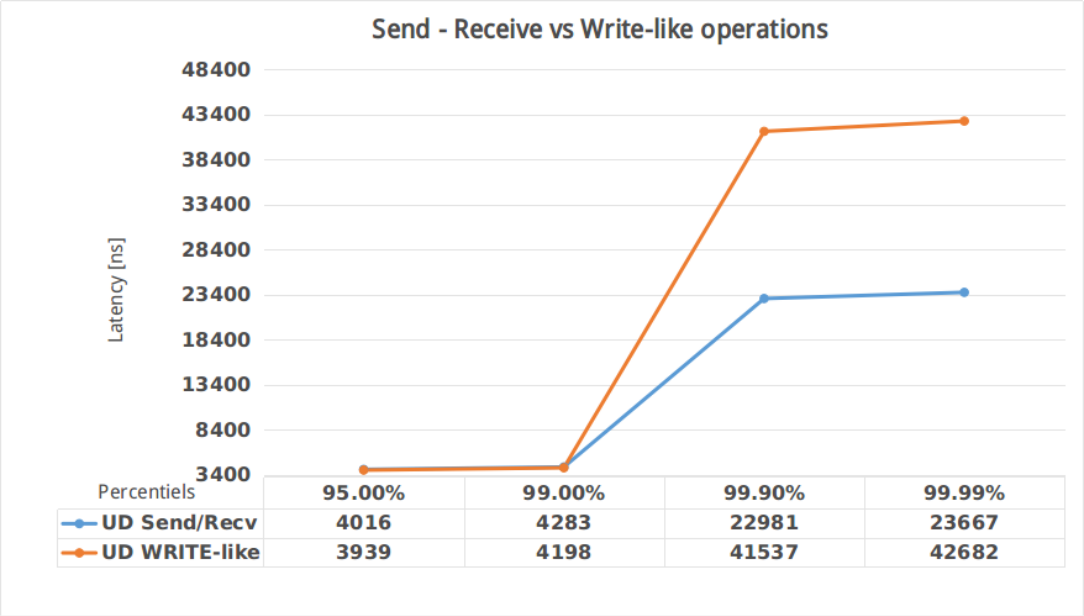


Figure 28 – WRITE-like send results in higher outliers than send – receive

5.4.5 RDMA RC timeouts for ACK / NACK

Using RDMA RD, one can set the minimum timeout that a QP waits for ACK/NACK from remote QP before retransmitting the packet. The verbs API defines 31 levels of timeouts. Level zero is special value which means waiting infinite time, this can be used for debugging. For any other value of *timeout*, the time calculation is: $4.096*2^{timeout}$ us [25].

I have implemented an application generating packet loss on the traffic, and I used the

setup shown in Figure 29 to use it in my experiments. The application drops the EtherType = 0x8915 packets with a low *P* probability, and was deployed on node B. This dropper application was implemented in DPDK, because it offers the possibility to inspect the frames, while they are still on the NIC. The application cannot be used with RoCE cards, since these cards are directly processing Ethernet frames used for RDMA communication. Thus, RoCE NICs do not forward them to the upper layers, in my case to the DPDK dropper application. Therefore, I had to use an Intel network interface card.

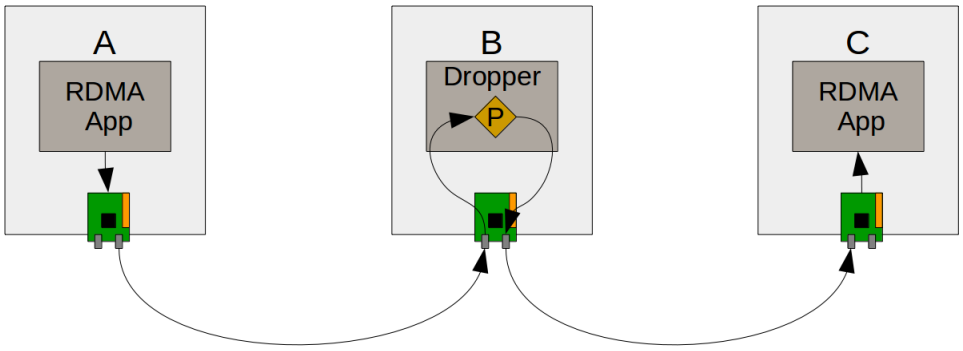


Figure 29 – The architecture for packet dropping

I measured the latencies coming from the timeouts waiting for NACK packets. The results are shown in Figure 30. The values plotted are coming from the higher percentiles of the data series. It can be seen that the reference and empirical values are overlapping only from level 16. If the desired application should implement some latency sensitive functions then using timeouts is not the appropriate choice.

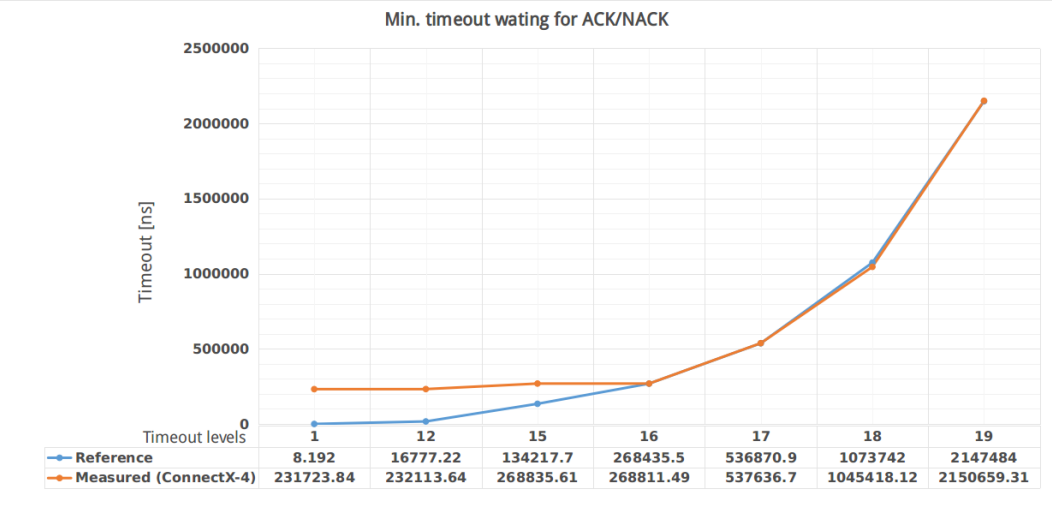


Figure 30 – Differences between the reference and the measured values

5.5 Discussion of the results

In Section 5.4 I have proposed several optimization methods to enhance the RDMA networking latency. I provide an overview of the proposed methods in Table 3, and summarize their benefit in the followings.

	RC/UC	UD
Aggregated post of RRs	Decreased outlier values	Decreased outlier values
Inlined send	~50% gain on performance for small messages	~50% gain on performance for small messages
RDMA WRITE	Decreased outlier values	-
WRITE-like send	Lower average latency, but higher outliers	Lower average latency, but higher outliers
Using Timeouts for ACK / NACK	Higher latencies	-

Table 3 – Combining optimization techniques with RDMA communication modes

When posting multiple RRs, aggregated posts should be used, since this solution results in lower worst-case values than posting RRs one by one. Taking the advantages of inlined send, one can halve the latencies for short messages. In case of subnets that are not oversized one should use RDMA UC or RC, since connection oriented RDMA supports RDMA WRITE operations, which has an impact on the rare but high latency values. In case of a huge subnet the best choice is RDMA UD, since there is no need to manage a huge number of RDMA connections. If the application is running over RDMA UD networks and the desired behavior is to react as fast as possible for most of the time and slower response time is allowed for the worst cases, the WRITE-like send - receive operations (described in 5.4.4) should be used. In case of using RC communication, using timeouts can result in high latency values.

6 Conclusion

RDMA was introduced to replace traditional socket based communication in order to enhance the performance of latency sensitive network applications. I have shown through a series of measurement based experiments, that indeed, the use of RDMA results in a significant gain in networking performance, regardless to the communication modes (i.e., reliable, unreliable or unreliable datagram). Nevertheless, the latency of RDMA based communication can be further reduced. During my research I proposed several mechanisms to achieve this goal.

I implemented and analyzed the effectiveness of these mechanisms, highlighting the gains achievable by their introduction. Selected features of these proposals were included by a major telecommunication equipment vendor to enhance the networking performance of its telco grade shared memory platform. My future work is to propose further refinements and adapt the RDMA based networking solution to meet the strict QoS requirements of telecommunication services.

References

[1] M. Risca, D. Malik, A. Kessler, *Trading Floor Architecture*, Cisco Press, 2008.

[2] Microsoft, *Data Center Bridging (DCB) Overview*, 2013.

[3] D. MacMuchael, T. Hudek, *Network Driver Design Guide*, 2017.

[4] *DPDK introduction,* available from: https://dpdk.org

[5] P Kutch, B. Johnson, *SR-IOV for NFV Solutions*, 2017.

[6] Intel, *VMDqTechnology white paper*, 2008.

[7] G. Robin, *Open vSwitch* with DPDK Overview,* Intel press*,* 2016.

[8] VMvare, *VMware vSphere ESXi and vCenter Server 5.1 Documentation*, 2016.

[9] J. Savill, *Are VMDq and SR-IOV performing the same function?*, 2012.

[10] P. Grun, *Introduction to InfiniBand for End Users, InfiniBand Trade Association*, 2010

[11] A. Ranadive, B. Davda, *Toward a Paravirtual vRDMA Device for VMware ESXi Guests*, VMTJ Winter Vol. 2., 2012.

[12] S. Brunner, *InfiniBand: An Introduction + Simple IB verbs program with RDMA Write,* available online from: ZHAW Zurich University of Applied Sciences blog, 2013.

[13] B. Dotan, *ibv_create_comp_channel(),* available online from: RDMAmojo blog, 2012.

[14] B. Dotan, *Verbs programming tutorial, Open SHMEM*, 2014.

[15] D. Barak, *ibv_post_send()*, available online from: RDMAmojo blog, 2013.

[16] *Introduction to InfiniBand,* Mellanox Technologies White Paper rev1.90, 2009.

[17] J. Chu, V. Kashyap, *Transmission of IP over InfiniBand (IPoIB),* IETF RFC 4391, 2006.

[18] S. Das, *Accelerate Your Applications with Hard RT and Reliable I/O over InfiniBand*, Mellanox Technologies, 2006.

[19] B. Dotan, *RDMAmojo blog - Connecting Queue Pairs*, 2014.

[20] A. Ayoub, *RDMA on vSphere: Update and future directions,* OpenFabric Workshop, 2012.

[21] *RDMA over Converged Ethernet,* Dell Networking Technical Brief, 2015.

[22] *Understanding iWARP* - Eliminating Overhead and Latency in multi-Gb Ethernet Networks*,* NetEffect White Paper, 2016.

[23] P. MacArthur, *Userspace RDMA Verbs on Commodity Hardware using DPDK*, IEEE 25th Annual Symposium on High-Performance Interconnects (HOTI), 2017.

[24] D. Balla, *RDMA based inter-host communication enhancement for cloud infrastructure,* BME Mesterpróba, 2017.

[25] D. Barak, *ibv_modify_qp()*, available online from: RDMA mojo blog, 2013.

[26] A. Kalia, M. Kaminsky, D.G. Andersen, *Using RDMA Efficiently for Key-Value Services*, ACM SIGCOMM, 2014.

[27] A. Kalia, M. Kaminsky, D.G. Andersen, *Design Guidelines for High Performance RDMA Systems*, USENIX Annual Technical Conference, 2017.

[28] A. Dragojevic, D. Narayanan, O. Hodson, M. Castro, *FaRM: Fast Remote Memory*, USENIX Conference on Networked Systems Design and Implementation, 2014.

[29] S. Sasaki, K. Takahashi, Y. Oyama, *RDMA-Based Direct Transfer of File Data to Remote Page Cache*, IEEE International Conference on Cluster Computing, 2015.

[30] M. Miao, et al., *SoftRDMA: Rekindling High Performance Software RDMA over Commodity Ethernet,* First ACM Asia-Pacific Workshop on Networking, 2017.
[31] B. Wheeler, *Universal RDMA: A Unique Approach for Heterogeneous Data-Center Acceleration*, Lineley Group Research Report, 2017.
[32] C. Guo et al., *RDMA over commodity ethernet at scale,* ACM SIGCOMM, 2016.