RADIOBLOCKS

Project ID: 101093934

High-speed data handling techniques

Deliverable:	D4.2
Lead beneficiary:	ASTRON
Submission date:	28 February 2024
Dissemination level:	Public

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101093934

Contents

RADIO BLOCKS

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101093934

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101093934

List of acronyms

ADC	Analog to Digital Converter
AI	Artificial Intelligence
ALMA	Atacama Large Millimeter/submillimeter Array
API	Application Programming Interface
CPU	Central Processing Unit
DAC	Directly Attached Copper
DAS6	Distributed ASCI Supercomputer
DDP	Direct Data Placement
DMA	Direct Memory Access
DPDK	Data Plane Development Kit
DRAM	Dynamic Random Access Memory
EHT	Event Horizon Telescope
ESRF	European Synchrotron Radiation Facility
e-VLBI	European VLBI network
EVN	European VLBI Network
FPGA	Field-Programmable Gate Array
FRB	Fast Radio Burst
GPU	Graphics Processing Unit
HBM	High Bandwidth Memory
HPC	High Performance Computing
IBTA	InfiniBand Trade Association
ICMP	Internet Control Message Protocol
IP	Internet Protocol
JIVE	Joint Institute for VLBI ERIC
KASI	Korea Astronomy and Space Science Institute
KVN	Korean VLBI Network
LOFAR	Low Frequency Array
ML	Machine Learning
MPA	Marker PDU Aligned
MPI	Message Passing Interface
ngEHT	next generation Event Horizon Telescope
NIC	Network Interface Connectors
OFED	Open Fabrics Enterprise Distribution
PCIe	Peripheral Component Interconnect express
PFC	Priority Flow Control
PKT	Packet
PMTU	Payload Max Transfer Unit
POSIX	Portable Operating System Interface
PPF	PolyPhase Filterbank
QoS	Quality of Service
QP	Queue Pair
RDMA	Remote Direct Memory Access
RoCEv2	RDMA over Converged Ethernet version 2
RR	Receive Requests
RX	Receive

RADIO BLOCKS

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101093934

SKA	Square Kilometer Array
SPEAD	Streaming Protocol for Exchanging Astronomical Data
SR	Send Requests
TCP	Transmission Control Protocol
UC	Unreliable Connection
UCX	Unified Communication X
UD	Unreliable Datagram
UDP	User Datagram Protocol
VLBI	Very Long Baseline Interferometry
WQEs	Work Queue Entries
WSRT	Westerbork Synthesis Radio Telescope

RADIO BLOCKS

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101093934

Abstract

This document describes the use cases and challenges faced for high-bandwidth data transport in large-scale distributed systems. Partners in the project provide use cases on several types of radio telescope systems as well as an industrial application. The available technologies are presented and a down-select is made on a per use case basis. DPDK, RoCEv2 and MPI are selected as the main technologies for further evaluation and implementation in the RADIOBLOCKS project. These three selected technologies are briefly assessed with hardware available in the DAS6 system and are found eligible for scale up to larger systems during the remaining time of the RADIOBLOCKS project. The document finishes with a description of what the partners aim to develop and how the partners aim to demonstrate the technology.

The document has been prepared by Steven van der Vlugt (ASTRON), John Romein (ASTRON), André Gunst (ASTRON), Mark Kettenis (JIVE), Willem-Jan Dirks (Sioux), Antsa Rasamoela (UBx) and Gie Han Tan (ESO).

1 Introduction

This document presents the application of high-bandwidth data transport in several types of radio-telescope systems as well as an industrial application. The different systems are all facing a bottleneck in Ethernet-based data transport between FPGA-based data producers and CPU + GPU based data consumers (and processors). The use cases and challenges faced for the different systems are outlined in Section 2.

The available technology is presented and described in Section 3 and a down-select of technology is made on a per use case basis in Section 4.

DPDK, RoCEv2, and MPI are selected as the main technologies for further evaluation and implementation in the RADIOBLOCKS project. These three selected technologies are briefly assessed with available technology in the DAS6 system, as described in Section 5. The selected technologies are found eligible for scale up to larger systems in the RADIOBLOCKS project. The document finishes with a description of what the partners aim to develop and how the partners aim to demonstrate the technology in Section 6.

RADIO BLOCKS

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101093934

2 Use Cases

This section describes several different use cases for high-bandwidth data transport in large-scale distributed systems. We describe two different kind of systems used in radio astronomy and one system for real-time image processing in an industrial application. Available technology will be evaluated based on the use cases.

2.1 Distributed Radio Telescope Systems

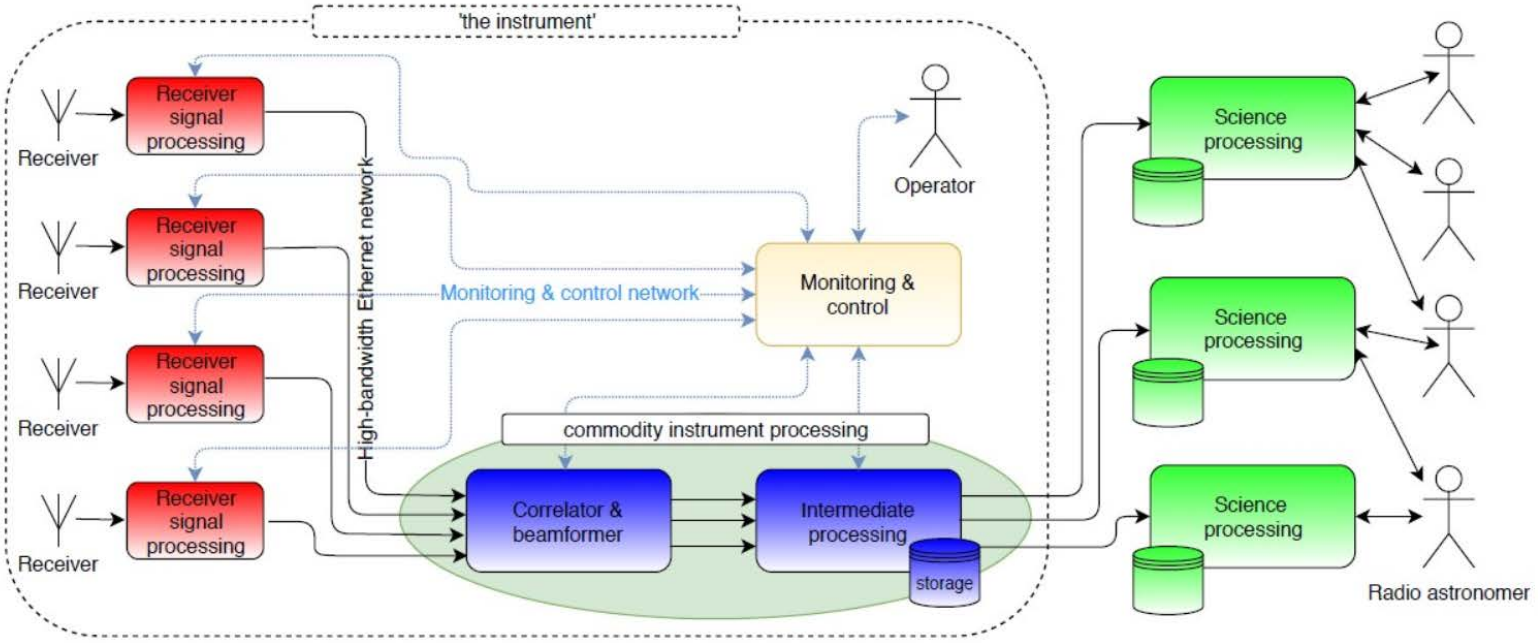

Figure 1: Top level overview of a generic distributed radio telescope [1]

In this section we consider data transport in distributed radio telescope systems such as LOFAR, WSRT, SKA, VLBI and ALMA. Modern radio telescopes generally consist of many receptors (small antennas or dishes), that are computationally combined into a single large virtual antenna using aperture synthesis. The principle driver behind this approach is to achieve higher, angular, resolution without the need of a single, large aperture, in other words antenna. Figure 1 shows an overview of a generalized distributed radio telescope system. In such a system we distinguish 5 general components:

1. antennas, receivers and digitizers; with an analog input and digitized signal as output
2. receiver signal processing; with a pre-processed stream of (channelized) data as output
3. correlator and beamformer; with frequency spectra or correlated visibilities as output
4. intermediate processing into science-ready intermediate products; with various data formats as output
5. science processing into science products; with various data formats as output

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101093934

2.1.1 Front-end to back-end communication

In this deliverable, we will mostly focus on the data transport between the receiver signal processing (2) and the correlator and beamformer (3). The receiver signal processing provides the output of the front-end[1], the data is transmitted to the correlator and beamformer, which is the first stage in the back-end. When discussing data transport in the system we will further refer to these components as front-end and back-end. More in depth information about a general distributed radio telescope system can be found in chapter 1 of [1].

In the front-end the antenna signals are digitized, filtered, and pre-processed. Although the methods of filtering and pre-processing may be different for different systems, we can generally speak of a conversion from sampled time-domain data to channelized frequency bands. The channelized data is transmitted to the back-end for further processing.

This data is transmitted in a streaming manner. During observation with the system, a continuous stream of samples is processed and transmitted to the back-end. The data transmission path shall be configurable per observation but will remain static during an observation. Due to large data rates at the front-end it is not feasible to buffer and re-transmit data in case of a transmission error.

Similarly, the first stages of data processing in the back-end have real-time requirements. If the data is not processed in time to receive new data, then part of the data has to be dropped. In addition, the front-end and back-end can be separated by several hundreds of meters up to 1000 kilometers (LOFAR). A signal arriving at one receiver will arrive at a different time at another receiver. Therefore, in order to recombine the signals in the back-end, it is essential to buffer the data before it can be processed. In current systems this data is buffered in external CPU memory and then moved to the GPU for further processing.

Processing in the back-end is a massively parallel operation, frequency channels can be processed independently. However, processing does require the same frequency channel from every antenna (front-end). This means that data either has to be redistributed. The data redistribution is often called the *corner turn* or *transpose*. The transpose can be performed inside the back-end system, or the data has to be transposed during data transport from front-end to back-end. With the latter option being greatly favored, as this means the data only has to be handled once.

Figure 2 shows a simplified example where each of the four front-ends produces four frequency channels, each node in the back-end receives and processes two channels from each front-end. This means that each front-end has to maintain four connections and each back-end node has to maintain 8 connections. If, for example, the data rate per channel is 10 Gbps, then each front-end has to support an outgoing Ethernet connection of 40 Gbps and each node in the back-end has to support an incoming connection with an aggregated data rate of 80 Gbps. The ratio between the data rate at the front-end and back-end will differ per system configuration.

We aim to build our systems with commodity components as much as possible, leveraging developments in other domains while minimizing costs and risks associated with bespoke designs. For data transport this means that we employ common technologies such as Ethernet standards, network switches, fiber optics and Directly Attached Copper (DAC) cables and PCIe Network Interface Connectors (NIC). Moreover, some systems partially rely on public Ethernet infrastructure. At the same time, we aim to use open components as much as possible.

Current transport implementations are based on UDP / IP transmission over (partially public) Ethernet. On CPU nodes, this is implemented in a software stack that covers a driver, a Linux

[1] Note that in some systems the front-end is only defined as the antenna feed, in the context of this study we define front-end up to and including antenna feed read-out and first-order processing close to the antenna. We prefer this definition because this also reflects the physical decomposition of the system.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101093934

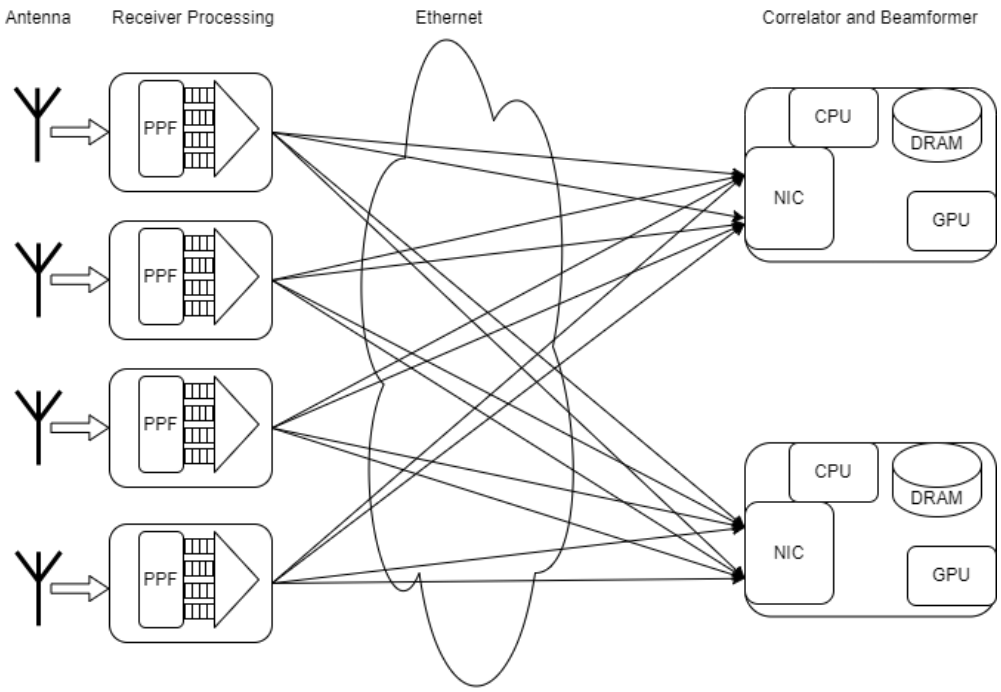

Figure 2: Simplified example of data transpose between front-end and back-end. In the front-end antenna data is sampled and filtered with a PolyPhase Filterbank (PPF) in the receiver processing, creating frequency channels. The frequency channels are distributed through the network to Correlator and Beamformer nodes in the back-end.

kernel, and a user-space application. The data is copied multiple times between these layers to achieve separation and protection, consequently imposing a significant load on the CPU. Figure 3 shows the conventional way in which the FPGA communicates data to the GPU memory in the remote host. After the data is received in the user space, it can be moved to the GPU, again requiring interaction with the kernel stack. For large Ethernet data rates, this leads to significant inefficiency (or a bottleneck) in CPU utilisation. Current and near-future network technology will allow Ethernet line rates of up to 400 Gbps, which are greatly beneficial for radio telescope systems. However, the current way of receiving and processing this (UDP-based) data is limited to approximately 40 Gbps.

In recent decades, several technologies have been developed that address this bottleneck, in this deliverable we will explore both software-based as well as hardware-based solutions:

- Software low-level control of the NIC or TCP/IP stack, reducing memory copies and Operating System overhead and optionally bypassing CPU memory, through direct data transfer to the GPU memory
- Hardware offloading through Remote Direct Memory Access (RDMA) in the NIC, bypassing the CPU and optionally bypassing CPU memory

We have a wide range of system requirements to support, ranging from distributed radio-telescope systems with a few, very high data rate receivers (e.g., ALMA) to systems with a large

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101093934

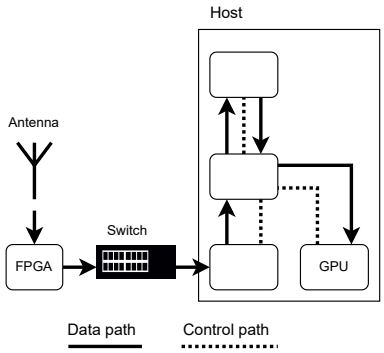

Figure 3:
Vanilla data transport through software stack

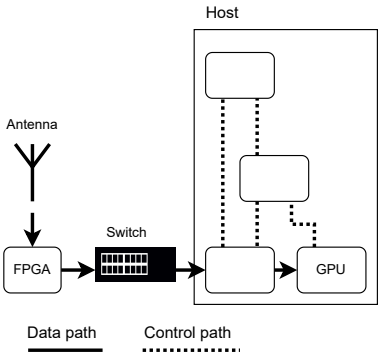

Figure 4:
RDMA data transport with zero-copy

amount of smaller, lower data rate receivers (e.g., LOFAR). For the first kind of system this means that the outgoing data rate at the front-end is very high and data is sent directly to one node or distributed over several nodes in the back-end. The ratio between front-end and back-end is in the order of 200 senders to 200 receivers (200:200). Ethernet technology used in these systems is preferably of the highest possible data rate all throughout the system. For the second kind of system the data rate at the front-end is relatively low, but the aggregated data rate at the back-end is very high. Every front-end node partitions its data to every node in the back-end. The ratio between front-end and back-end is in this case in the order of 1000:20. Ethernet technology used in the front-end might be of older generations, but at the back-end it is preferred to use the highest possible data rate that technology has to offer.

2.1.2 Inter back-end communication

In the case of the LOFAR system the Correlator and Beamformer and the intermediate processing are implemented on two different clusters. Meaning that output from the online (real time) correlator and beamformer processing pipelines has to be send to a second system for offline intermediate processing. Data rates between these systems are in the order of 100 - 200 Gbps. The interface between these systems is currently based on Infiniband and MPI. In order to reduce the diversity in technology used in our systems, we aim explore for next-generation systems if it is possible to use Ethernet here as well. While at the same time keeping the same efficiency and application abstraction as with Infiniband based solutions. This use case is largely similar to the VLBI system use case (Section 2.3 but adds that here data is transmitted from the GPU in one system to the CPU or storage in another system.

2.2 Network multi-cast

The LOFAR system follows the architecture of a general aperture synthesis radio telescope as depicted previously in Figure 1. For future upgrades of the system there is a need to sent the same front-end data to multiple back-ends at the same time. The ambition is to piggy back on ongoing observations with other systems that are specific to a science case. Such as a new cluster which is dedicated to Fast Radio Burst (FRB) research. In addition, supporting distribution of data to multiple back-ends will allow to stream data from a front-end to the back-end of the central processor and a local back-end of an international station at the same time, making

RADIO BLOCKS

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101093934

the telescope more efficient. We aim to distribute the front-end data to multiple back-ends by multi-casting the Ethernet data streams.

Figure 5 depicts the network architecture for the future LOFAR upgrade. Central to this architecture is the Core Layer, implemented with 400 GbE technology. The Core Layer allows distribution of the front-end data to the Standard Central Processor Production System (at the bottom of the image) as well as Experimental and Non-standard dedicated systems, depicted to the left and right of the Core Layer.

In the context of this deliverable it needs to be taken into account how multi-cast might impose additional or contradicting requirements on the high bandwidth data transport technology that is to be selected.

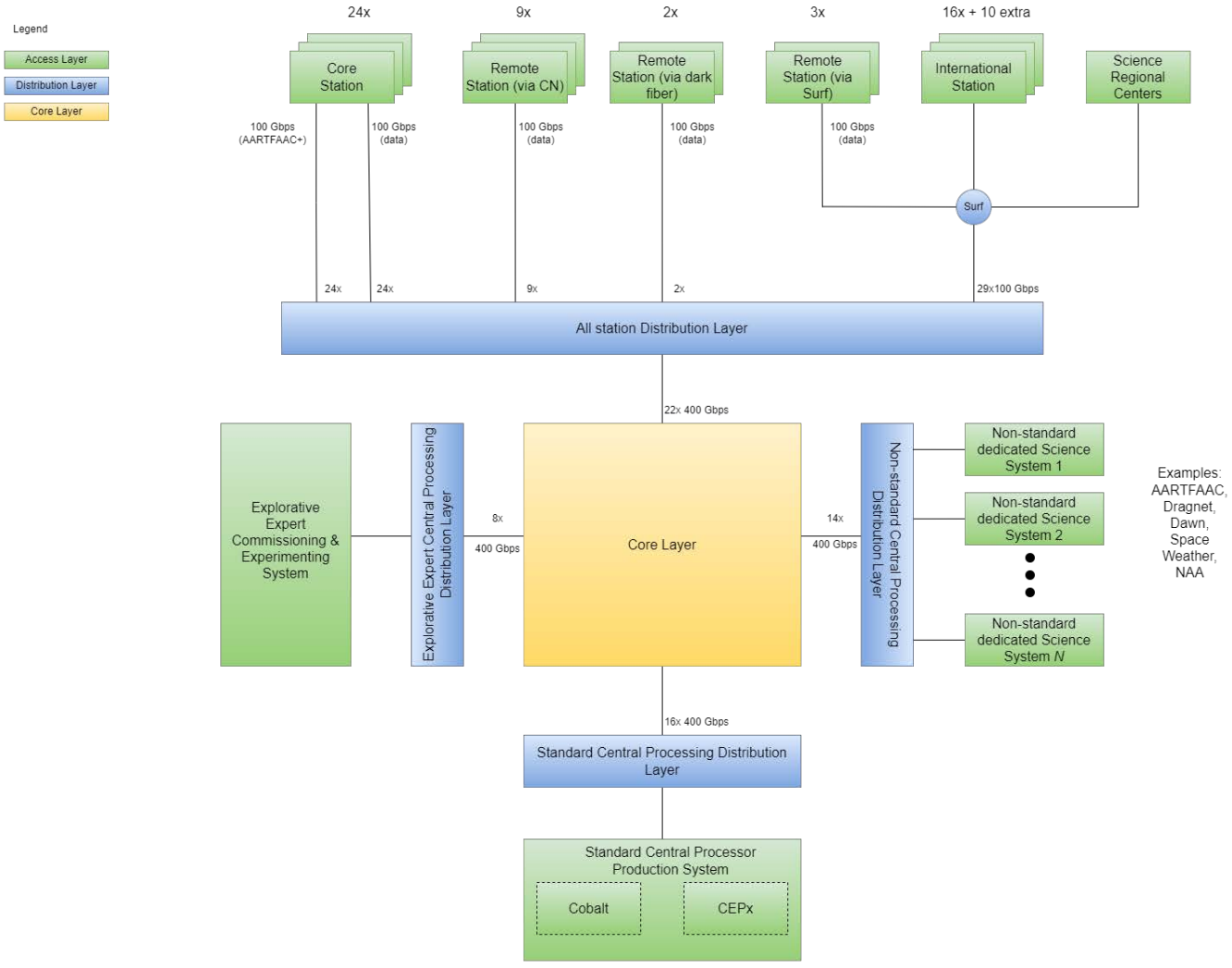

Figure 5: Network design for future update of LOFAR

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101093934

2.3 VLBI correlator

Very Long Baseline Interferometry (VLBI) is the ultimate example of a distributed radio telescope system, where the antennas are distributed on a global scale. Arguably the most well known example of a VLBI array is the Event Horizon Telescope which operates at mm-wavelengths (230 Ghz and 345 Ghz) and produced the first image of the black hole at the center of our galaxy. But several other VLBI arrays exist such as the European VLBI Network (EVN) that consists of telescopes across Europe and beyond that operate at cm wavelengths (between 350 MHz and 43 Ghz), and the Korean VLBI Network (KVN) that consists of three telescopes in South-Korea operating at both mm and cm wavelengths (between 43 Ghz and 230 GHz). Telescopes from different VLBI arrays often observe together to form even bigger arrays. A key feature of a VLBI array is that the individual antennas each operate their own high-precision clock which requires some additional synchronisation steps during correlation.

By their very nature, VLBI arrays need to transport data over large distances. While real-time e-VLBI, where data is directly streamed from telescopes to the correlator, is possible in some cases, shipping data to the correlator on disk remains the dominant operating mode for VLBI. In this scenario, data is recorded on disk at the telescope on VLBI data records (for example the Mark 6 VLBI Data Recording System [2]) and played back at the correlator through similar units. Another common approach is to use dedicated storage arrays at both the telescope and correlator and transfer data electronically over the internet. This approach works even if the available network bandwidth is lower than the recording data rate.

Since the data recorders and/or storage arrays are typically separate nodes from the systems that are used to implement the correlator (regardless of whether the correlator is a CPU or GPU correlator), this means that a VLBI correlator will have to be a distributed application where large amounts of data will have to be transferred between the storage systems and the correlator systems (see Figure 6). To prevent this data transfer from becoming a bottle-neck when accelerators are used to implement correlation, this data transfer needs to be done as efficiently as possible. When the correlator is implemented on GPUs, it is desirable to transfer the data directly from the storage systems into GPU memory.

VLBI data typically uses sampling with a very low number of bits. Many VLBI backends pack multiple 2-bit sample streams into a single data word. In order for the correlator to be scalable, it may be desirable to parallelize processing across these sample streams by sending each stream to a different correlator node. This requires a corner turn operation. Since GPUs typically do not have instructions that operate on individual bits, this corner turn is potentially better done on a CPU. This corner turn operation needs to be done for real-time e-VLBI as well, it makes sense to stream data through the storage buffers as well for that use case. This is in fact what happens in the current CPU correlator setup at JIVE. Therefore the VLBI use case would likely benefit from efficient network transport between main memory of one node to GPU memory on another node. The VLBI community standardized on Ethernet for communication between components over the last decades although Infiniband is also used at some of the correlator sites.

Current VLBI data rates vary from 4 Gbit/s per telescope for VLBI at cm wavelengths (e.g., the European VLBI Network) to 64 Gbit/s at mm wavelengths (for the Event Horizon Telescope). The EVN is working towards data rates of up to 32 Gbit/s per telescope in the near future. KASI is also planning to upgrade their VLBI system, with data rates up to 40 Gbit/s from all KVN sites to the Daejeon Correlator Center. Equipment that supports these data rate is already present at many of the telescopes in the array. For the next-generation Event Horizon Telescope (ngEHT), which is currently being planned, data rates are expected to reach 320 Gbit/s at telescopes that

[2] https://www.haystack.mit.edu/mark-6-vlbi-data-system/

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101093934

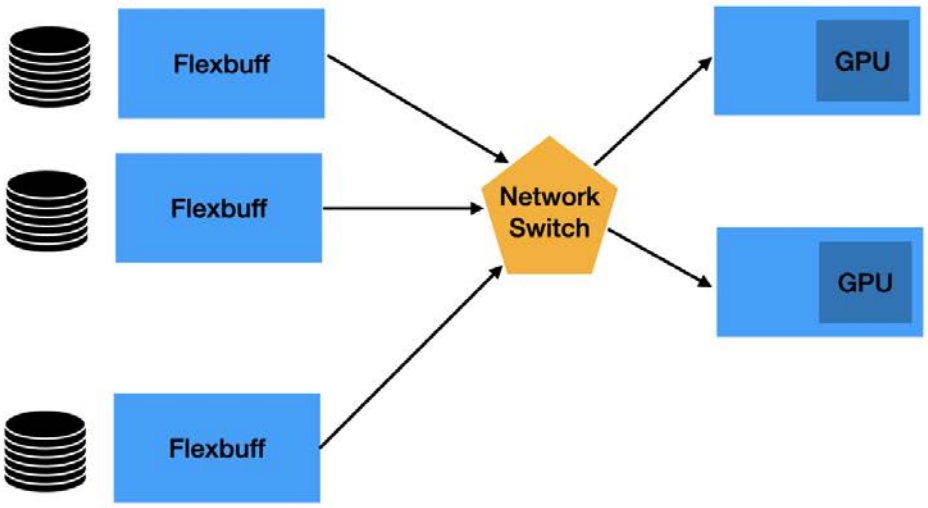

Figure 6: Simplified multi-node VLBI correlator

can observe up to three frequency bands simultaneously.

In order to support these data rates in the future, we will evaluate technology for high bandwidth data transport from CPU and or storage systems over Ethernet to CPU and or GPU systems.

2.4 Reliable large data transfer in an image inspection system

In this section, data transport in image inspection systems is considered, with flow control for reliable data transmissions.

The targeted system is used to detect artifacts in images by comparing a captured image with reference images (majority vote anomalies detection) of the same object. The complexity of this system is besides the artifact detection, also the huge amount of image processing pipelines, several steps of this processing pipeline are performed on the edge of the system. Current research is aiming for a scalable solution, efficient for a few processing pipelines to what is maximal, cost-effective, to handle in a system identification of edge conditions.

In the system represented in Figure 7, two images are used as a reference for error detection

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101093934

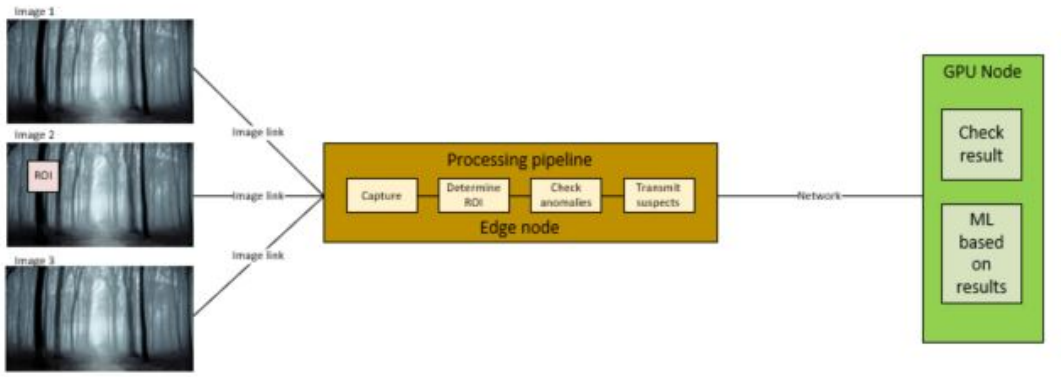

Figure 7: Image processing pipeline

on a third image. However, the number of images could vary due to system requirements.

The processing node focuses on detecting parts of the image, region of interest (ROI) which potentially have anomalies. This area is sent to a GPU cluster where a machine learning algorithm (ML) will analyze the cropped images for true anomalies. The expected total data to be transferred between edge devices and a GPU cluster will be large ($\geq$ 1 Tbps) due to the number of pipelines and resolution of the image transferred. For this system, an efficient way to move data from edge devices to the GPU cluster is essential. Just increasing the network pipeline is not a cost-effective way to process all data. After processing, the images in the pipeline are available on the output of a processing unit.

Although multiple image pipelines are now available on a single output, the connection to a GPU cluster is not defined as a static relation. This means that not all results of a processing unit will go to the same GPU in the GPU cluster. This routing shall be flexible. To allow for this flexible selection of processing GPUs, a standard Ethernet network is targeted due to its proven flexibility and scalability. No other network technology has this track record. Lots of vendors are available, which allows for an open standard. Current innovations concerning Ethernet for "AI" clusters, targeted by the Ultra Ethernet consortium, would only benefit this system.

Using a flexible routing pool of data sources and data destinations, as shown in Figure 8, will result in the most optimal allocation of resources. Also, if needed redundant clusters could be added to increase system up-time, reduce downtime and cost-of-service. However, redundancy is out of scope for a proof of concept.

Since Ethernet-based networks are common technologies, lots of initiatives are available for additional quality-of-service improvements to specific data profiles. A candidate technology should be based on proven network technologies and would efficiently help to lower latency in the network by removing data transfer CPU bottlenecks, optimizing GPU cluster utilization and performance, and allow for scalability of the system. When transferring a large amount of data over a shared network, the network should be designed optimally to avoid congestion. However, when system size increases, this is very complex to obtain for all the contributing network nodes with current technologies. Also, the network should allow for priority traffic. Not all data is rewarded the same priority, and this needs to be part of a future solution.

The type of data transmitted over the network is limited to the images captured, and marginal control data to setup and control the data flows. Therefore, the network behavior when transporting this type of data is predictable. Since images are captured at a certain rate, due to the

RADIO BLOCKS

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101093934

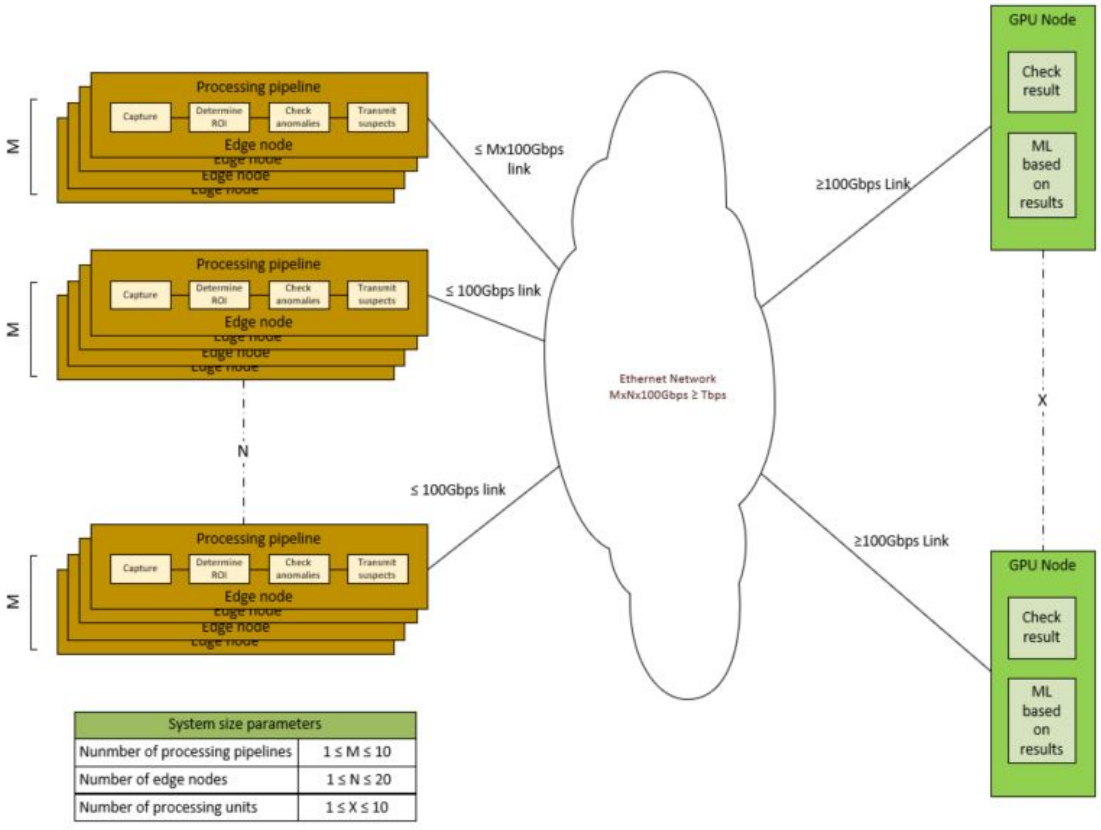

Figure 8: flexible routing pool

object being scanned, the number of still images will vary. As a result of this, the data generated will vary as well.

The end solution for this system should also allow for a Quality of service (QoS) layer to prevent needless retransmissions or large buffering. Re-transmissions would increase system cost since data needs to be buffered and will also increase system latency. To minimize the amount of data lost due to network reliability all network nodes are in scope for the QoS solution.

The main characteristics of this application are the constant flow containing large amounts of data ($\geq$ 1 Tbps) which needs to be processed efficiently by GPU-nodes. In this deliverable we aim to select an efficient protocol for enabling these system characteristics. However, besides determining a best fit for current system definition, the system shall be scalable in the future, using standard interfaces and protocols. Several standard protocols need to be analysed and compared for a best fit.

RADIO BLOCKS

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101093934

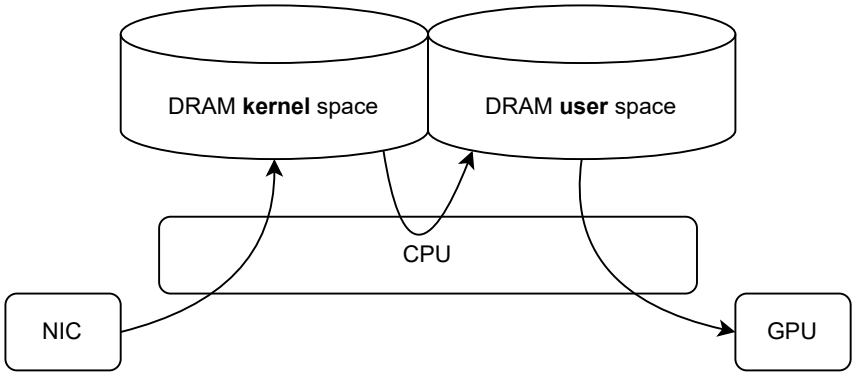

Figure 9:
Data movement from NIC to GPU through the vanilla software stack and Linux kernel

3 Available Technology

3.1 Vanilla network stack

Current transport implementations from radio telescope front-ends to back-ends are either based on proprietary physical connections and protocols, or as introduced before in Section 2.1.1 based on UDP / IP transmission over (partially public) Ethernet. We aim to use commodity technology as much as possible and favor Ethernet over proprietary implementations. Though widely available and nowadays capable of 400 - 800 gigabit line rates, Ethernet also has its limitations. Streaming, UDP based Ethernet can be well implemented on an FPGA at a front-end. In the back-end, on CPU nodes, the receiving side of this data stream is implemented in a software stack that covers: a driver, the Linux kernel mid layer and user space application. The data is copied multiple times between these layers to achieve separation and protection, consequently imposing a significant load on the CPU. Moreover, each UDP packet arriving at a node (destined to that node) triggers an interrupt (or event, through polling) to the Operating System which has to be served, resulting in a context switch.

Figure 3 shows the conventional way in which the FPGA communicates data to the GPU memory in the remote host. After the data is received in the user space, it can be moved to the GPU, again requiring interaction with the kernel stack. Figure 9 shows this situation in more detail. First, data arriving at the network interface triggers an interrupt to the CPU, data is handled by the UDP network stack and stored to kernel owned memory. Next, data has to be copied by a user application to a GPU data buffer in user owned memory and can then be moved to the GPU for further processing. The data is stored to memory twice and loaded from memory twice. Moving processed data from the GPU to another node in the network traverses these steps in direction from GPU to NIC, requiring two additional load and store operations. For large Ethernet data rates, this leads to significant inefficiency (or a bottleneck) in CPU utilisation and CPU to DRAM bandwidth.

Current and near-future network technology will allow Ethernet line rates of up to 400 Gbps (even 800 GbE is already on the horizon), which are greatly beneficial for radio telescope systems. However, the current way of receiving and processing this (UDP-based) data is limited to approximately 40 Gbps per NIC and CPU socket.

Several alternative methods have been developed to overcome the data copy overhead in

RADIO BLOCKS

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101093934

the CPU node. Subsection 3.2 will discuss general trade-offs in Software v.s. Hardware based solutions. Both alternatives will be detailed out in the succeeding subsection 3.3 and subsection 3.4.

3.2 Software and Hardware based solutions

The default UDP way of receiving Ethernet based data already has to be tuned to reach a performance of about 40 Gbps on conventional technology. Tuning parameters include packet size, interrupt priority, core pinning, numa domain co-location, memory page configurations and shared receiver queues. In order to increase performance beyond this point, adaptions will have to be made to the software architecture of the conventional software stack, these can be roughly categorized to:

- Additions to the Linux kernel. One currently already mainstream option is *io_uring*, where multiple packets can be stored in a circular buffer that is accessible by both the kernel and the user-space application. This eliminates several context switches as well as one memory copy.
- A framework such as the Data Plane Development Kit (DPDK) takes direct control of the NIC, thus skipping the Operating System Network stack. This reduces the kernel overhead and kernel to user-space data copies, but does require the user (or framework) to implement the network stack.
- Several other libraries and implementations exist where one either modifies the kernel network stack, takes over control of the interface or creates an addition to the network stack. For example, the MeerKAT system uses the SPEAD protocol and spead2 implementation[2].

Common for all software-based solutions are that the sender might keep sending data as normal UDP and memory copies are reduced by either direct user access to the interface or by sharing a buffer between the user and kernel. However, software-based solutions still require handling network interrupts or polling for every packet or collection of a few packets, thus severely loading the CPU for high network bandwidths.

On the other hand one might offload the processing and inspection of network traffic to specialized networking hardware. There are several hardware offloading alternatives available:

- Direct Memory Access through the NIC. iWarp and RoCE are protocols that are based on the proprietary Infiniband. Both protocols are encapsulated in or on top of the default TCP/UDP network protocol. Similar to Infiniband, these protocols allow the NIC to directly place received data in reserved CPU or GPU memory. However, iWarp and RoCE require changes to how data is being sent. Several NIC models support these protocols.
- FPGA-based SmartNICs, based on PCIe FPGA cards with a network interface, allow the user to design an alternative to iWarp or RoCE or even directly use TCP/UDP to receive data, and either send it to the CPU or store the data to CPU or GPU memory directly. Though this solution is more flexible than the nowadays commodity solutions with iWarp and RoCE, it does introduce additional complexity (both the FPGA firmware and driver will have to be developed and maintained), moreover an FPGA PCIe card is typically more expensive than a NIC.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101093934

Common for the hardware-based solutions is that the load on the CPU and DRAM memory is significantly reduced, thus eliminating the previously identified bottlenecks and allowing for a more energy and cost efficient system. However, (at least small) modifications are required to how data is transmitted.

As a third category we identify communication frameworks that offer abstraction, such as MPI, UCX and Holoscan, and technologies that build on top of either software or hardware based solutions.

The next three sections and their subsections will take a deeper dive on software based solutions (section 3.3), hardware based solutions (section 3.4) and communication frameworks (section 3.5).

3.3 Software based solutions

3.3.1 Data Plane Development Kit

The *Data Plane Development Kit* (DPDK) is a toolkit for sending and receiving Ethernet packets at high data rates. Intel started developing the toolkit in 2010 as an open-source project; meanwhile the toolkit is supported by all major network interface vendors and works on a wide variety of CPU architectures. The toolkit contains an extensive set of libraries that can be used by applications to accelerate packet processing, manage memory buffers, etc.

DPDK obtains its performance by bypassing the operating system in the critical communication path. The operating system is only involved in setting up the network interface, but after initialization, the network interface is essentially under full control by the application. The operating system is not involved in the receipt, sending, and processing of network packets. This basically eliminates interrupt, context switching, and kernel-space–user-space packet copying overheads; overheads that are unavoidable when communicating through the normal POSIX I/O interface of the operating system, and are prohibitive when one needs to communicate at hundreds of gigabits per second.

As DPDK operates at the Ethernet packet level, the interface is quite low level. Network protocols on top of Ethernet, such as TCP/IP and ICMP, are not natively supported by the toolkit, it is the responsibility of the application program to implement such protocols (though the toolkit provides access to protocol-specific functions like IP checksum calculations, that can be offloaded to the network interface hardware).

There is a large security risk by handing over a network interface to the application: the application can send and receive any packet on the network, while critical resources that are normally under control by the operating system are now fully exposed to the application. In practice, this means that any DPDK application needs to be run with superuser privileges (or similarly elevated rights). This makes it impractical for use in data centers that are used by multiple organizations (as security between organizations cannot be guaranteed), but for radio-astronomical instruments this is normally not a issue, as those systems are normally dedicated for use as part of the instrument only.

Recently, a highly interesting feature has been added to DPDK: GPU integration. This allows incoming packets to be received directly into GPU memory. Even better: packet headers can be received in CPU memory, and packet payloads (with sampled data) can be received in GPU memory. This way, the CPU can inspect the timestamps in packet headers, and when enough packets have been received, the CPU instructs the GPU to process the packet payloads (e.g., to filter and/or correlate the data in the packets). GPU integration is currently only supported

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101093934

for NVIDIA network interfaces, paired with NVIDIA GPUs. If there is a PCIe switch between the network interface, CPU and GPU, incoming packet payloads are copied directly from the network interface to GPU memory (using DMA), such that the bulk of the data does not pass through the CPU at all.

3.3.2 CUfile

CUfile is a recently released GPU library by NVIDIA, that allows reading and writing files directly to and from GPU memory. Our use case would be to write visibilities from GPU memory to file. We already experimented with this library, and found that asynchronous writes will only be supported in a future release of the library. This is somewhat inconvenient, but can be worked around by using synchronous writes. More problematic is that the library only supports files and not sockets, therefore it is not possible to stream visibility data (the output of the correlator) via a TCP connection to another machine with this library. This will also not be possible in a future release of the library, due to some unfortunate design choices in the Application Programming Interface. The DPDK demo correlator uses this library optionally, only if the visibilities need to be written to file, and if the CUfile library is supported on the system at all (it requires a very recent version of the CUDA toolkit, that is not supported on all systems yet).

3.4 Hardware based solutions

3.4.1 Remote Direct Memory Access

RDMA (Remote Direct Memory Access) is a method for data exchange, originating from the proprietary InfiniBand protocol and later also implemented on Ethernet as iWarp, RoCEv1, and RoCEv2. In this section, the differences between the available network transport protocol suites are listed and explored. Figure 10 shows a high-level overview of the network layering of these RDMA implementations. These network layers can give insight into the complexities, limitations, and possibilities of different RDMA implementations. One might notice that each RDMA protocol uses the Verbs API in the application layer. The Verbs API is described in subsection 3.5.1.

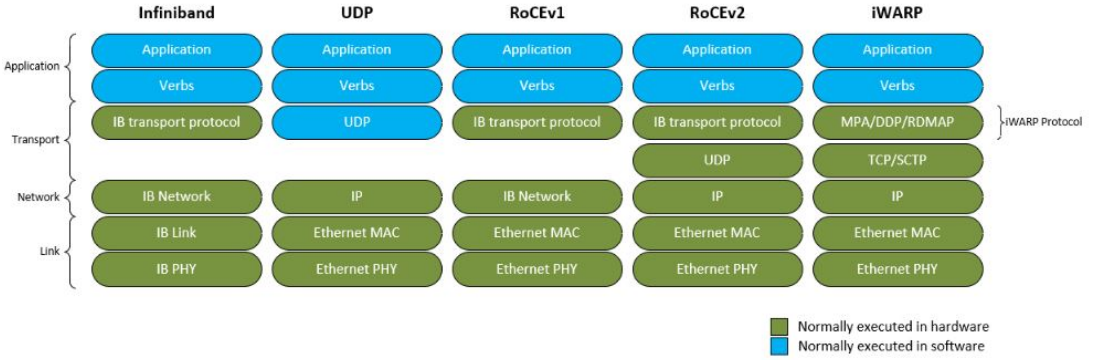

Figure 10: RDMA network layer overview

InfiniBand has been designed as a computer cluster interconnect by the InfiniBand Trade Association (IBTA) to achieve high bandwidths and low latencies. The InfiniBand network stack uses InfiniBand-specific protocols in each layer below the Verbs layer. The InfiniBand RDMA

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101093934

implementation uses InfiniBand fabric, which reduces header overheads compared to Ethernet fabric. Besides, InfiniBand fabrics and protocols are optimized for low latencies and short distances compared to Ethernet fabrics. InfiniBand hardware more expensive and make use of proprietary headers and fabrics. This makes it difficult, or even impossible, to implement on FPGAs compared to the open-source available Ethernet protocols. Since InfiniBand uses special hardware and is supported by a limited amount of vendors, its adoption remains low compared to standard Ethernet.

UDP is a widely used transport layer protocol to transport data in an unreliable connectionless manner over commodity Ethernet. The protocol does not require prior communication to set up the communication channel and does not require specialized internet hardware. The UDP network stack is shown in Figure 10. The figure shows the offloading of the IP layer as it is executed in hardware; this reflects the partial UDP/IP offloading possibilities, such as checksum offloading and fragmentation offloading. Depending on the level of hardware acceleration, the amount of CPU involvement is reduced. However, standard NICs do not offload data interpretation and therefore the CPU load will still be high. The UDP protocol might be offloaded to an FPGA based smart NIC.

RoCEv1: RDMA over Converged Ethernet version 1 (RoCEv1) is a technology that combines the RDMA and widely used Ethernet fabric. It uses InfiniBand directly on top of the Ethernet MAC, which could be seen as a layer 2 implementation. This means that RoCEv1 can operate on a standard Ethernet network, using standard network equipment. However, layer 2 networks will limit the number of endpoints allowed on a network since all nodes belong to the same domain. Moreover, RoCEv1 is not routable across subnets since it does not use the Internet Protocol layer.

RoCEv2: RDMA over Converged Ethernet version 2 (RoCEv2) uses the well-known and commonly used Ethernet link layer and the IP and UDP protocols. It differs from RoCEv1, which does not use IP routing. RoCEv2 allows network designers to transport the data traffic through commodity Ethernet cables and switches. This is a massive advantage over InfiniBand since it does not require a particular cable type and switches. To fully benefit from RoCE, one should use network adapters that support RoCE in hardware. Otherwise, one is limited to softRoCE, which leverages the CPU cores instead of the network card.

RoCE uses many mechanisms of InfiniBand in the network and application layer, as shown in Figure 10. As a result, several higher-level APIs have almost identical capabilities for RoCE and InfiniBand. The IP and UDP mechanisms contain extra services to improve the networking and routing of packets. RoCE does support these extra services or protocols, such as VLANs and QoS, to improve security and routability.

Note that UDP does not ensure a reliable transfer, which might cause issues in various application domains. This can be taken care of by the higher-level IB (InfiniBand) transport protocol implementation. Regarding the network stack, the IB transport protocol adds headers that contain information about the RDMA operation and connection. Hence the maximum theoretical throughput is reduced compared to native UDP.

In RoCE one refers to different services. The default UDP based protocol is an unreliable service, where the extended implementation that does support re-transmissions is referred to as the reliable service.

RoCEv2 is supported by the major network interface card vendors like Intel, Broadcom and Nvidia. Next to a wide support in NIC's there are also several RoCEv2 implementations for FPGA, refer to section 3.4.4 for more information.

iWarp uses several conventional lower-level network protocols such as the Ethernet link layer and IP and TCP. The first difference with RoCE is that iWarp uses TCP instead of UDP. The TCP protocol ensures a reliable connection, therefore this does not need to be taken care of at

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101093934

a higher level, as is done in RoCE. But one loses the flexibility to use an unreliable connection.

The core of iWARP consists of three protocols, the Direct Data Placement (DDP) protocol, RDMAP and marker PDU aligned (MPA) protocol. The DDP protocol handles the data interaction with the memory to allow kernel bypass data transmission. iWARP uses RDMAP to implement read-and-write services and a wire protocol. The RDMAP uses the DDP protocol to enable direct memory access mechanisms. Lastly, the Marker PDU Aligned Framing (MPA) protocol is needed as an *adaption layer* between the TCP and DDP layers.

Due to all additional layers, and associate protocols, as shown in Figure 10, the complexity of the iWarp protocol is the highest among these five architectures. iWARP is supported by the major vendor network cards.

RDMA protocol selection Based on the protocol descriptions above and an elaborate analysis in prior work [3] the RoCEv2 is deemed most suitable for the use cases in the context of this deliverable. In the next subsection we provide a short background on the RoCEv2 protocol. For a more extensive background description we refer the reader to the report by W. de Laat [3].

3.4.2 RoCEv2

RoCE flow types: RoCEv2 supports several data transport modes, all with different characteristics with respect to flow reliability and system performance. Selecting a transport mode has a direct impact on the established Queue Pairs (QP). A Queue Pair is the RoCE concept that sets up a connection between a sender and receiver. Table 2 shows a summary of all the flow types and corresponding options.

Attribute		RC	UC	UD
Reliability	Data order	Guaranteeed	Detected	No
	Data delivery	Guaranteeed	Detected	No
	Corrupt data	Recovered	Detected	Dropped
Operation support	Send	✓	✓	✓
	Receive	✓	✓	✓
	RDMA Write	✓	✓	**X**
	RDMA Read	✓	**X**	**X**
Multicast		**X**	**X**	✓
Scalability[a]		M^2 x N	M^2 x N	M x N

Table 2: Transport mode comparison

[a] M flows on N endpoints communicating with all processes on all nodes

Reliable connection (RC) is a definition of a connection where a QP is associated with only one other QP. The send queue is delivered to only one receive queue and there is no *out-of-order-delivery* of packets allowed. Detection of missing or out-of-order messages is identified by a sequence number in each message received. Also, the reception of messages is acknowledged by the receiver and if messages are missing, a re-transmission can be triggered. This transport mode is very similar to TCP on IP layer 3. When having an RC at the RDMA level, the need for TCP at the transport layer level becomes irrelevant and this allows for UDP layer 3 traffic, which simplifies the connection management for one QP. A disadvantage of this transport method is that an RDMA endpoint needs different QPs per message flow. This can inflate the number of

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101093934

QPs drastically when a higher number of flows is supported per endpoint. Also. The mechanism of re-transmission will decrease the effective bandwidth used.

Unreliable Connection (UC) is a definition of a connection where, identical to the RC, one send QP is associated with only one receive QP. However, in this transport mode, there is no *out-of-order-delivery* of packets check performed, and therefore also no re-transmissions are supported. However, the sequence counter, which is incremented upon each packed, is monitored. This allows for detection of missing messages but there will be no re-transmission triggered for the missing message. This connection type allows for a better utilization of the available bandwidth; however, error correction or concealment is shifted to the application layer. Also, for UC a disadvantage of this transport mode is the number of QPs per endpoint when the endpoint supports more than one data flow. No automatic re-transmission of missing or corrupt messages could be seen as a disadvantage, but might also be an advantage depending on the application.

Unreliable Datagram (UD) is a definition of a connection type that allows a single QP to connect with multiple destination QPs. A big advantage of this transport method is the number of QPs in the endpoint. The number of QPs is now dictated by the number of flows supported and not by the number of endpoints receiving the flow. This allows *one-to-many* distribution of a single flow. This could be compared to the multicast principle in an IP layer 3 principle and is seen as the *multicast version of RDMA*. However, other than layer 3 IP multicast, for RDMA multicast still the QP sizes need to be aligned for all QPs associated with the multicast flow, before an endpoint can start transmitting messages. For this connection type, it is identical to UC, and no re-transmission is supported. UD can still detect out-of-order or missing packets, but actions taken are the responsibility of the application. The transmitter will not be informed. For the defined connection types, not all message operations are available.

There are several less common and less applicable services or variations on services (e.g. extended RC and UC) that are left out of the scope of this analysis.

RDMA operations: The RoCE protocol defines multiple operations to exchange data between end nodes. These operations can be divided into "true" RDMA operations and non-RDMA operations. Operations involving the remote user-space application are called non-RDMA operations. These operations are also referred to as double-sided operations since they require work requests on both end nodes. RDMA operations, on the other hand, are referred to as single-sided operations, meaning the remote node is unaware of the operation. We do not discuss ATOMIC and memory binding operations as they are irrelevant to our use cases.

The **SEND (/RECEIVE)** operation is regarded as a non-RDMA operation and supported by all transport services. The operation requires the involvement of the responder because the requester's work request does not set the message's destination. In other words, the requester does not need to know the memory address it should write into. The responder should thus provide the destination of the message. Hence, the destination QP must have a corresponding receive request inside the receive queue containing the memory location before the SEND operation arrives; if not, the message will be dropped. As a result, the responder is continuously engaged in posting receive requests, causing this operation to generate more CPU load in the responder than a true RDMA operation. Figure 11 shows a simplified sequence diagram of the SEND operation. The figure indicates that the requester and responder must post send requests (SR) and receive requests (RR), respectively. The responder's acknowledgement is solely sent when using a reliable transport service.

Data transport without user-space involvement of the responder is achieved with the **RDMA WRITE** operations. This is a single-sided RDMA operation to write data from the requester to the responder. The requester specifies the remote memory address and extra keys in the send work request. Figure 12 depicts the sequence diagram for WRITE operations (without immediate

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101093934

data). One should notice that extra communication is needed to exchange memory information before the data transport starts. Hereafter, the requester can post send requests and receives completion events after transmission or acknowledgement, depending on the transport service.

The WRITE operation can be augmented with **Immediate data** allowing to add metadata to the datastream. The Immediate data consists of 32 bits placed in the receive work completion event and not directly written into the memory. Hence, requiring a receive work request to be consumed and transformed into a completion event by the NIC.

Another true RDMA operation is the **READ** operation, allowing the local node to read remote memory instantly without remote host intervention. This operation is only supported by reliable services. An abstract sequence diagram for this operation is shown in Figure 13.

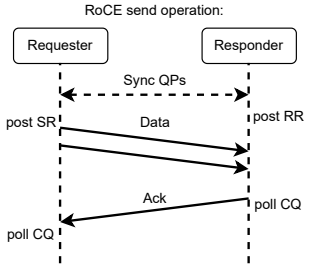

Figure 11: Send sequence diagram

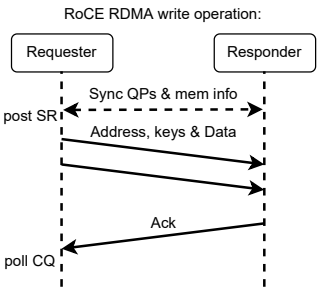

Figure 12: Write sequence diagram

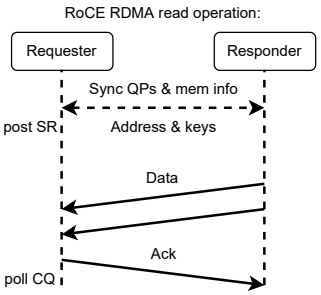

Figure 13: Read sequence diagram

Priority Flow Control and back pressure: To guarantee that no packets are dropped and/or to achieve optimal utilization of available bandwidth by avoiding re-transmissions, all network nodes, endpoints, and switches will have buffering. Switches used in the network will have ingress and egress buffering to allow messages to wait until a port is free for that particular message to be sent out.

To allow for higher and lower priority traffic the buffers are allocated on a priority-packed-based definition. This results in 8 priority queues which can hold messages from different flows. When a port is free for sending out a packet, the packets in the highest priority queue will be permitted first. When the number of flows in a system is high, it is very likely that multiple flows will end up with the same priority class and therefore will use the same priority buffer in a switch.

To avoid overflowing a buffer, which might result in dropping packets for that queue, the switch could decide to send a *Pause* message to the upstream device for that link. This *Pause* message could end up eventually at the transmitting endpoint which will then stop its message flow unit until a *Resume* message is received. This will allow the buffers in the network to drop between a save threshold before accepting new packets. A graphical representation of priority flow control (PFC) is shown in figure 14.

It is up to the transmitter application (based on the use case) to determine if stopping a flow is allowed. For file transfers or non-real-time data, this might be allowed. For applications where the source is generating a continuous flow of data, this could be an issue since there is a risk of losing data. Also, the stop message will hold all message flows on its port with that priority class. It cannot decide to stop individual flows of that priority class. There are efforts made to improve the flow-control principle in the Ethernet standard, however these are currently out of scope.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101093934

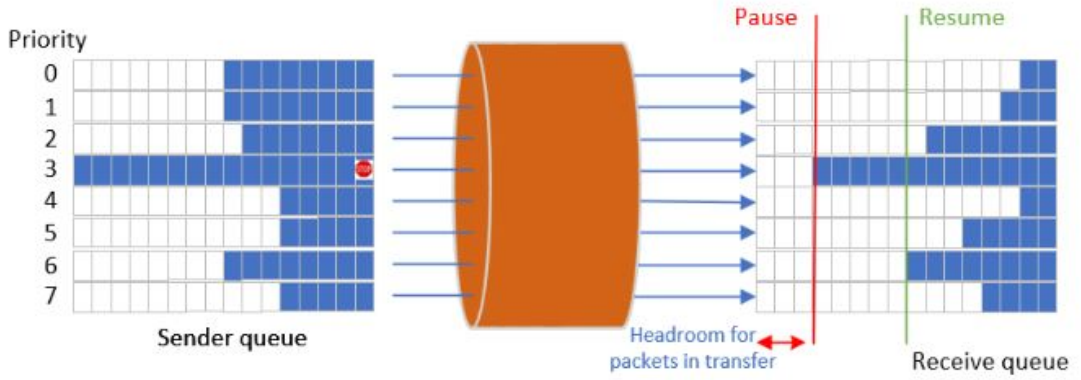

Figure 14: Priority Flow Control

Security: RDMA or RoCEv2 does not currently define special security additions since it was designed for *private use*, like private clusters. RDMA by itself is unencrypted. An endpoint can use normal credentials to provide network access and build a message flow. Encryption of the RDMA data would have a large impact on the current benefits like bypassing the CPU for user space memory access. However, RoCEv2 does set-up a connection pair with unique identifiers between two end-points. This control operation is performed on a TCP network which leverages IP-SEC. Also, compared to DPDK, RoCEv2 requires a limited set of elevated user rights on a Linux system, making it more secure to work with than DPDK. Since RDMA is shifting from private environments to the cloud, security is a key topic. Multiple efforts have been started to secure RDMA, focusing on end-to-end security or adding security to (programmable) switches to avoid NIC updates. Since these efforts have not yet resulted in updates to the standards, these principles are out of scope.

3.4.3 Smart NIC FPGAs

Alternatively, to a NIC that supports hardware accelerated RDMA, one might consider an FPGA PCIe card with high bandwidth Ethernet connectivity. There are several PCIe gen 4 accelerators that support multiple 100 or 200 GbE connections and 400 GbE capable PCIe 5 accelerators are also coming more widely available at the time of writing. The flexibility of the FPGA would allow one to implement either Ethernet based protocol and, with Direct Memory Access, move the data from the FPGA accelerator over PCIe into CPU memory.

There are several open Smart NIC FPGA frameworks such as Corundrum[4] and AMD's OpenNIC project[5], both are currently limited to 100 GbE support. In the context of astronomy telescope systems the SKA Low CBF is targeting an FPGA based Correlator and Beamformer combined with a Smart NIC FPGA solution[6] and the Green Flash project has delivered a Smart NIC FPGA based design for the adaptive optics control system in the Extremely Large Telescope[7].

The implementation of a Smart NIC FPGA for the systems considered in this document (back-end) would mean that a current UDP based sender (front-end) in the system might remain unaltered, similar as with a solution based on DPDK. However, the flexibility of an FPGA based solution comes at the cost of having to design (and maintain) the FPGA firmware and the PCIe CPU driver, which is both far from trivial. Moreover, FPGA accelerators are typically more expensive than an RDMA capable NIC. This is especially the case for devices that support 400 GbE and PCIe gen 5.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101093934

3.4.4 IP cores for FPGA

In order to either send (front-end) or receive (with a Smart NIC FPGA in the back-end) RoCEv2 one would have to implement the protocol in the FPGA firmware. At the time of writing there are several open as well as proprietary implementations available. In the next two subsections both are listed.

3.4.5 Open designs

Schelten et al. present a network-attached accelerator (NAA) framework with partial support for RoCEv2 [8, 9]. This framework implements a Reliable Connection allowing SEND and RDMA WRITE operations. They implement the READ operation via a so-called pre-RDMA operation which starts a WRITE operation on the remote node. Unfortunately, this implementation supports just a single RDMA connection and the implementation is not publicly available at the time of this study.

Multiple connections are supported in the RDMA acquisition system for high-performance applications (RASHPA) framework developed by Mansour et al. [10] for the European Synchrotron radiation Facility (ESRF). Yet, they provide an even more custom RDMA protocol over Ethernet based on RoCEv2. Besides, they report throughput results between CPU and FPGA but do not report the number of QPs nor the resource utilisation on the FPGA. This makes it difficult to compare them quantitatively.

StaR is proposed in [11] to solve the scalability problem of RDMA by transferring states to the other communication end. Despite their research showing that their proposal works, they, like the aforementioned ones, do not use a mature and general protocol.

The SKA consortium, whose smart NIC implementation was discussed earlier, implemented the RoCEv2 SEND over unreliable connection in VHDL to be used between two FPGAs. However, the receiving application[12] is open but the FPGA implementation is not.

Korolija et al. [13] implement an open-source RoCEv2[14] stack operating at 100 GbE into their Coyote Framework[15]. Coyote is an extensive framework providing secure spatial and temporal multiplexing of FPGA kernels, network interfaces (such as RDMA, TCP and UDP) and memory services to leverage host memory and the FPGAs DRAM or HBM memory. The network stack is an improvement on previous open-source FPGA projects such as Limago [16], and StRoM [17]. The framework is designed to work with AMD Alveo FPGA accelerators. The StRoM network stack has been evaluated between two FPGAs by T. Song [18]. The stack supports WRITE and READ operations over a reliable connection service and supports up to 500 connections, which should be relatively easy to enlarge as the stack consists of HLS code. This RoCE stack was designed to work between FPGAs and between FPGA and NIC. In order to evaluate FPGA to CPU/GPU communication this implementation was evaluated by W. de Laat [3]. This evaluation is summarized in Section 5.2.

3.4.6 Proprietary designs

Grovf IP-Core

Grovf Inc.[19] is an application acceleration and network offload company using FPGA programmable chips. Operating since 2017, the company has created solutions for big unstructured data processing and 100 Gbps network traffic monitoring and analysis. Currently, the company is focused on memory and storage dis-aggregation solutions over cache-coherent buses and RoCE networking technologies. The GROVF RDMA IP core

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101093934

and host drivers provide RDMA over Converged Ethernet version 2 (RoCEv2) system implementation and integration with standard Verbs API.

Highlights of the Grovf RDMA IP core:

- Fully compatible with known RNIC products and soft RoCE implementations (RoCE v2).
- 100 Gb/s Throughput, under 2.7 usec latency (roundtrip-software), under 0.5 usec on FPGA implementations
- Configurable RDMA Queue pairs, up to 8000.
- Hardware retransmission and reordering

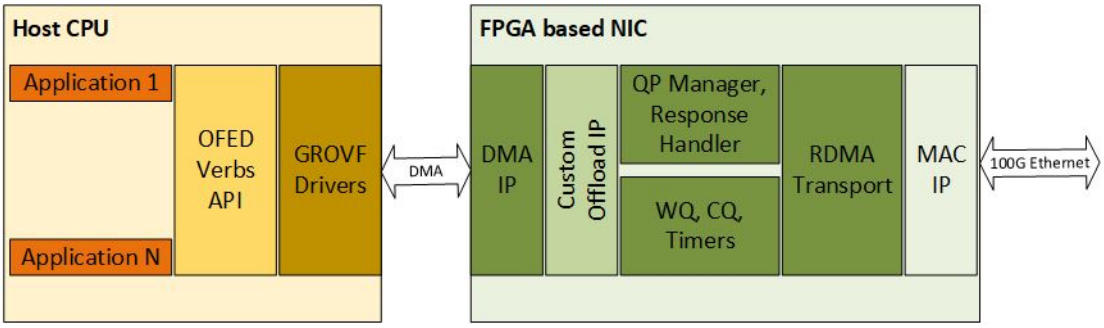

Figure 15: Grovf IP core architecture

The solution from Grovf is a soft IP implementing RDMA over Converged Ethernet protocol. It consists of FPGA IP integrated with MAC and DMA, plus the host CPU drivers. The solution complies with Channel Adapter and RoCE v2 requirements as stated in the IB specification. Below is the simplistic architectural overview of the system. The data plane and reliable communication is hardware offloaded and the implementation does not include CPU cores in FPGA.

The Grovf core is available for Intel Agilex and AMD Ultrascale+ devices.

AMD ERNIC IP-Core

AMD is the high-performance and adaptive computing leader, powering the products and services that help solve the world's most important challenges. AMD technologies advance the future of the data center, embedded, gaming, and PC markets. The ERNIC (Embedded RDMA enabled NIC) IP from AMD provides an initiator and target implementation of RDMA over Converged Ethernet version 2 (RoCEv2) enabled NIC functionality. This IP is specifically designed for embedded applications that require reliable transmission over Ethernet networks. This parameterizable soft IP core can work with a wide variety of AMD hard and soft MAC IP implementations providing a high throughput, low latency, and completely hardware offloaded reliable data transfer solution over standard Ethernet. The ERNIC IP allows simultaneous connections to multiple remote hosts running RoCEv2 traffic. A Linux driver is provided with the ERNIC IP that can run on the ARM processor (Zynq, Zynq Ultrascale+ MPSoC) and MicroBlaze processor.

RADIO BLOCKS

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101093934

Highlights of the ERNIC ip core

- Support for RDMA SEND, RDMA READ and RDMA WRITE for incoming and outgoing packets. Atomic operations are not supported. RDMA SEND with immediate and RDMA WRITE with immediate opcodes are not supported.
- Support for RDMA READ, RDMA WRITE, and RDMA SEND work requests
- Support for up to 254 connections
- Scalable design of up to 255 RDMA Queue pairs
- Supports dynamic memory registration with proprietary APIs
- Hardware handshake mechanism for efficient doorbell exchange with the user application

The QP Manager module houses the configurations for all the QPs and provides an AXI Lite interface to the processor. It also arbitrates across the various SEND Queues and caches the SEND Work Queue Entries (WQEs). These WQEs are then provided to the WQE processor module for further processing. This module also handles the QP pointer updates in the event of re-transmission.

The WQE Processor Engine reads the cached WQEs from the QP Manager module and handles the following tasks:

- Validates the incoming WQE packets for any invalid opcode
- Creates the header for the packets based on the Payload Max Transfer Unit (PMTU) and programs the scatter Gather Lists (SGLs) for the internal DMA engine
- Triggers the DMA to start the outgoing packet transfers

The WQE Processor Engine is also responsible for sending outgoing acknowledgment packets for the incoming RDMA SEND/WRITE requests and read responses for incoming RDMA READ requests.

The RX PKT Handler module receives the incoming packets. The ERNIC IP handles the following types of incoming RoCE v2 packets:

- RDMA SEND, RDMA WRITE, RDMA READ and response packets for RDMA READ (request sent from ERNIC)
- Acknowledgment packets for RDMA WRITE/RDMA SEND (request sent from ERNIC)
- Communication management (Management Datagram) packets to QP1

The RX PKT Handler module is responsible for the incoming packets. It also triggers outgoing acknowledgment packets for incoming RDMA SEND and RDMA WRITE requests and pushes the packets that pass the validation to the corresponding memory location The RDMA READ responses are channeled to the target application directly. The module handles the incoming RDMA READ requests and forwards the request to the Tx path.

The Response Handler module manages the outstanding queues. These queues hold the information about all packets sent to the remote host but have not yet been acknowledged or responded to. In addition this module triggers a re-transmission if the remote host sends a Negative Acknowledgment (NAK). If this module does not receive a response from the remote host within a specific time (timeout value), it triggers a timeout related re-transmission.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101093934

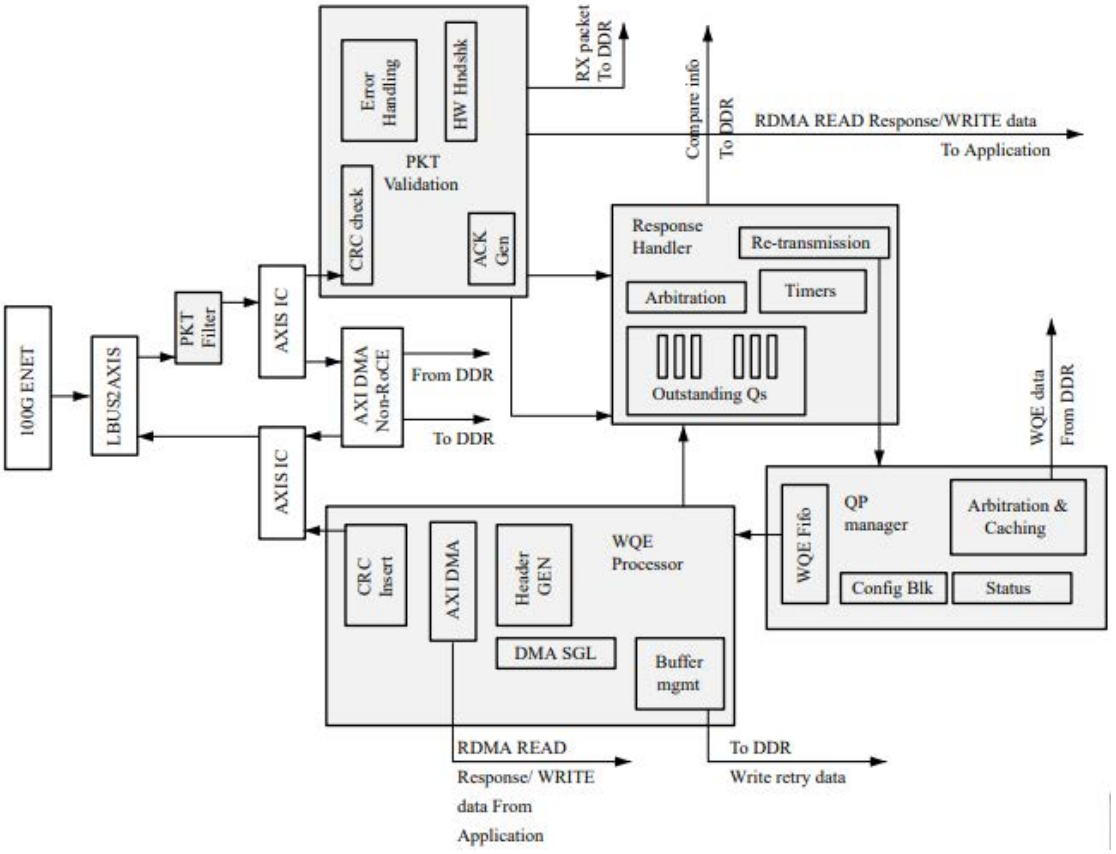

Figure 16: ERNIC IP core architecture [20]

3.5 Communication frameworks and supporting technology

Some of the use cases are likely to be distributed across multiple (CPU/GPU) nodes. An example is the traditional VLBI use case where pre-recorded data needs to be distributed from multiple storage nodes to one or more CPU/GPU nodes. Software libraries are available that leverage the technologies discussed before to optimize such data transfers. This section discusses the most widely used libraries.

3.5.1 ibverbs

The verb layer is designed by the Open Fabrics Alliance and is also called Open Fabrics Enterprise Distribution (OFED) verbs API. OFED is open-source software for RDMA and kernel bypass applications developed for Linux systems. The verb layer is often referred to as *ibverbs* library or as Verbs API. The verbs layer offers a wide support for hardware targets, Operating Systems and driver combinations. A subset of the features is portable across all devices and each hardware target comes with it's own supported set of additional features. The ibverbs library or Verbs API can be used directly (low level) by an application or through additional abstraction layers. The ibverbs methods allow for sufficient control of an RDMA session to set-up a connection to a non

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101093934

standard device, such as an FPGA.

3.5.2 Direct memory access to GPU memory

The *ib_core* kernel module allows 3rd party memory to be registered and used for RDMA operations. Combined with direct access to GPU memory this allows to form a direct data path between the network card and the GPU memory. Hence, the data does not need to be moved through the main memory or CPU, improving the latency and avoiding bottlenecks caused by main memory I/O operations.

Though several open, more generic implementations are in development, currently the only well supported option is to use Nvidia *PeerDirect* with an Nvidia GPU and Nvidia (Mellanox) NIC. The interaction between the Nvidia and RDMA interfaces and modules is displayed in Figure 17. The necessary permissions and other requirements to bind GPU memory to the *ib_core* module are facilitated by the ivberbs layer in combination with the Nivida PeerDirect kernel module, a standard component of the recent Nvidia drivers. However, it is important to mention that this kernel module is not supported by all combinations of Linux kernel and OFED driver versions.

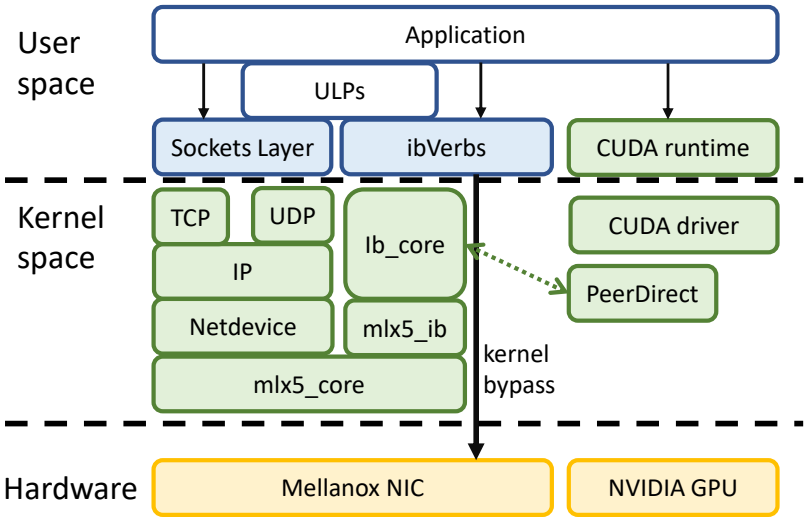

Figure 17: Nvidia PeerDirect software stack architecture

3.5.3 UCX

A relatively new entry in the software stack is UCX, which is advertised as “a communication framework for modern, high-bandwidth and low-latency networks”. UCX provides a set of higher-level APIs that integrate RDMA and GPU technologies. The framework is split into a low-level transports API (UCT) and a high-level protocols API (UCP). For data transport it supports RoCE, InfiniBand, Cray Gemini/Aries as well as plain TCP sockets and various shared memory implementations and provides automatic selection of the best transport mechanism. On the GPU side it offers seamless handling of GPU memory access for both NVIDIA’s CUDA and AMD’s ROCm. In particular it provides APIs that allow direct transfer of data from the network into GPU memory.

RADIO BLOCKS

This project has received funding from the European Union’s Horizon Europe research and innovation programme under grant agreement No 101093934

3.5.4 MPI

The Message Passing Interface (MPI) is an even higher-level API that has been the staple of distributed HPC applications for the last three decades. Multiple implementations exist, with Open MPI and MPICH being the most popular implementations. The MPI standard defines language bindings for both Fortran and C, but bindings are available for other languages as well. MPI implementations provide tools that can launch applications in many environments, including typical cluster scheduling setups. These tools take care of setting up basic communication infrastructure between MPI processes running on multiple nodes. This makes writing scalable multi-node applications much easier than by using a lower-level API like UCX or ibverbs.

Recent implementations of both Open MPI and MPICH are implemented on top of UCX. These implementation offer MPI extensions that provide support for CUDA and/or ROCm. These extensions allow the use of device virtual addresses in most of the standard MPI API calls. This makes it easy to implement data transfers from normal (CPU) memory on one cluster node directly into GPU memory on another node. Or even data transfers from GPU memory on one node to GPU memory on another node. Since MPI also provides APIs for asynchronous data transfers, it is possible to overlap GPU compute with those data transfers. Many of the MPI APIs can be implemented as a very thin layer on top of the UCX APIs. Therefore the expectation is that MPI adds very little overhead on top of UCX.

3.5.5 Holoscan

Nvidia Holoscan is a technology designed to accelerate data transport and optimize communication within data center environments as well as edge devices. Holoscan is an abstraction of data transport and compute resources and can use DPDK and UCX to improve data transport performance. The Holoscan Software Development Kit is open, but currently mostly supports Nvidia hardware. Holoscan is being evaluated for application in a radio telescope system at the Allen Telescope Array [21] where data is received from an FPGA over Ethernet with UDP.

3.5.6 SPEAD

The Streaming Protocol for Exchange of Astronomical Data (SPEAD) [22] has been designed specially for data transport in radio telescope systems. The protocol is based on multicast UDP and is specifically designed to transmit multi-dimensional arrays or data with associated metadata.

The *spead2* library is a high performance CPU implementation of the SPEAD protocol, building on top of ibverbs. With the spead2 implementation the NIC directly writes UDP packets in to the systems external RAM, where the data is assembled to a configured format. The RAM assigned to spead2 might be GPU memory that is mapped in to the systems address space. Thus allowing to directly and efficiently move data to the GPU without the need for an additional copy in RAM. The spead2 library can recognize missing data but does not correct or re-transmit data. Upon transmitting data from GPU to another node the NIC directly accesses the GPU memory without a need for a copy to CPU RAM. However, current implementations of spead2 are limited to a performance of approximately 120 Gbps because the implementation does not have support for multi-core execution [2]. In addition, the SPEAD format is quite specific and allows for little flexibility, consequently it might not be a good fit for every system.

3.5.7 Other

The authors are aware of other implementations. However, these implementations are less suitable for the use cases or are less mature than the technology already described in this chapter.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101093934

4 Technology down select per use case

In this chapter we describe per use case, why a certain technology is chosen for further evaluation.

4.1 Distributed Radio Telescope Systems

4.1.1 Front-end to back-end communication

In this section, we discuss the requirements for a distributed radio telescope system. We will compare DPDK and RoCEv2 and for RDMA will elaborate on the optimal transport service and RDMA operation.

Summarizing considerations already discussed in the previous chapter:

InfiniBand versus Ethernet: Compared to Ethernet InfiniBand fabrics are more expensive and make use of proprietary headers and fabrics. This makes it difficult, or even impossible, to implement on FPGAs compared to the open-source available Ethernet protocols. Moreover, Infiniband is only suitable for short range connections (several meters), making it impossible for front-end to back-end communication in a radio telescope. Therefore, we will target Ethernet based solutions.

SmartNIC FPGA versus NIC: Compared to a NIC an FPGA based SmartNIC offers flexibility, but introduces additional complexity and cost. We aim for solutions with NIC's that support RDMA.

Software solutions: Currently there are several frameworks that abstract lower level software (and hardware) solutions for usage in data centers. However these frameworks (currently) do not fit to the network topology and streaming data of a radio telescope system and do not integrate well with an FPGA as a sender. DPDK provides a mature and high performant software solution to high bandwidth data transport.

RDMA protocol: In Section 3.4.1 we already discussed the different RDMA over Ethernet implementations. We select RoCEv2 over RoCEv1 and iWarp. There are several benefits already mentioned on RoCEv2. The most important factors to choose this protocol for this use case is that RoCEv2 is the best fit for a streaming data network, due to its support for unreliable data transport (UDP). Moreover, RoCEv2 is less complex to implement on FPGA than iWarp.

The use cases is characterized by the following requirements:

High bandwidth data transport: With data transport from front-end to back-end being the most dominant bottleneck in today's radio telescope systems we shall be able to achieve near line rate at the current available fastest technology. This means that the technology selected shall support at least 100 GbE, 200 GbE and 400 GbE and shall be scalable to future data rates. Both RDMA and DPDK theoretically allow currently communication speeds of up to 400 GbE. RDMA might be more scalable than DPDK as DPDK loads the CPU significantly more than RDMA.

Streaming data transport: Data is streamed from the FPGA front-ends to CPU/GPU back-ends. Re-transmission of data from sender to receiver is not possible due to large data rates and limited buffer capabilities at the sender. The sender shall be able to identify if data is dropped or if data has arrived out of order. Both DPDK and RDMA support configurations with unreliable connections and flagging of missing data and out of order packets.

Many senders to many receivers: The topology between front-end and back-end can differ from many senders to several receivers or many senders to many receivers. Both the sender

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101093934

and receiver will have to maintain multiple connections. Both DPDK and RDMA support this topology. For DPDK the amount of connections might impact the system load. For RDMA the amount of connections will impact the hardware resource requirements.

Receiver system load: The load on the receiving system (front-end) is currently the bottleneck in receiving UDP based data. System load shall be reduced significantly compared to UDP in order to be able to receive data at rates above 40 GbE. Moreover, reducing load on the system will allow to design a system with smaller CPU, reducing system material cost, and will reduce energy consumption during operation. This also makes the technology interesting for radio telescope systems with data rates below 100 GbE. RDMA will reduce the system load significantly as the bulk of the workload is offloaded to the NIC. DPDK allows to receive at high data rates, but at a high CPU load.

Direct access to GPU memory: Writing data directly to GPU memory reduces the load on the CPU but also adds complexity. This option should be explored, as it might be beneficial to some system configurations. It is likely sufficient if data arrives in CPU memory directly and is copied to the GPU with a ping pong buffer, while the GPU is computing on the previous data buffer. This way the memory copy does cause system load and latency, but does not hamper throughput. The main consideration here is the GPU memory size. Especially at very high data rates, such as 400 Gbps, the GPU memory will fill up quickly.

Implementation complexity: Abstraction of implementations (such as data transport) allows to reduce the system complexity. Network sockets with standard UDP are at a relatively high abstraction level. However, abstraction and performance are often contradicting. Improving the data transport will add complexity to the system design. DPDK adds significant complexity to the receiving software. RDMA will add significant complexity to the sending firmware and some complexity to the receiving software. In the scope of this project we will first work with a low level implementation of the technology. However, when the implementation has matured, we aim to add abstraction for easy integration with other systems.

Hardware versus Software Both DPDK and RDMA are promising for distributed radio telescope systems. Both technologies have different pros and cons and this might differ also per system configuration. We will evaluate both technologies in this project and make a comparison.

RoCEv2 service and operation configuration:

RoCEv2 has a large configuration space in combinations of transport service and operations, here we describe the configuration that fits best to the distributed radio telescope system use case. The **transport service** defines which operations are possible. The first requirement to be highlighted is that re-transmission of data is impossible due to each use case's real-time and high throughput constraints. Consequently, the two remaining transport services are Unreliable Connection and Unreliable Datagram. The UD service can connect a single QP to multiple remote QPs, improving scalability since fewer resources would be needed. Other differences between the two services are related to the message size and supported operations. The message size for UD is limited to 4KiB, impacting the CPU performance, and it only supports SEND operations. Whereas UC supports SEND and WRITE operations with message sizes up to 2GB. Besides, the UD requires additional state-keeping, which also increases the complexity of the implementation on FPGAs. We opted for the UC transport service because the UD has higher complexity and limited message size.

The Unreliable Connection supports **SEND** and **WRITE** operations, which can be extended with Immediate data. For the use case, it is acceptable if data is dropped (at low occurrence), however it must be clear which data has been received correctly. Adding immediate data to the WRITE operation does allow insight into received and missing data as the required information can be encoded into the immediate data. However, the sender (thus the FPGA) must provide the

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101093934

memory location per message, which is a disadvantage because this requires additional state keeping. When using the SEND operation (with immediate data) the receiving side (CPU/NIC) determines where the data from the incoming message will be placed. Eliminating the need for the sending side (FPGA) to store and send along the memory location. However the SEND operation is not a true RDMA operation, meaning that it will result in a higher CPU utilization.

Therefore, for this use case the most suitable RDMA implementation is the Unreliable Connection with the WRITE operation.

RoCEv2 on FPGA:

As described in Section 3.4.4, there are several implementations available for RoCEv2 on FPGA. We rated these implementations to our requirements:

- The implementation of the protocol shall **meet RoCEv2 specifications**, in order to be used with a standard implementation of the protocol on a NIC.
- The implementation shall support the **UC WRITE** configuration. As this has been selected as most favorable configuration for our use case.
- The implementations shall meet the required **network topology**, or shall be scalable to the required topology (many-to-many).
- For a demonstrator of the technology we are targeting an Intel Agilex 7 development kit together with UBx. Therefore, the implementation shall at least **support Intel FPGAs**.
- As far as possible, we shall use open and **public available** technology.

Requirement	NAA	RASHPA	StaR	SKA	Coyote	Grovf	ERNIC
Meets RoCEv2 specifications	+	-	-	+	+	++	++
UC WRITE support	-	-	-	-	-	+	++
Network topology	+	?	?	+	+	+	+
Support for Intel FPGA	++	-	?	+	-	+	–
Public available	-	+	+	-	+	–	–

Table 3: Evaluation of FPGA IP cores

As summarized in Table 3, none of the implementations meets every requirement. Out of the public available implementations, only the Coyote framework meets the RoCEv2 specifications. Coyote is an actively maintained framework and a continuing development of the Limago and StRom network stack implementations. It was selected as the best starting point for an (adapted) implementation, though this framework is targeted towards AMD Alveo FPGAs and does not support the UC WRITE configuration. The Coyote framework was evaluated in prior work, described in [3] and summarized in Section 5.2 of this report.

4.1.2 Inter back-end communication

The inter back-end communication use case aims for efficient data transport from several CPU-GPU nodes to several CPU-GPU nodes, with low implementation complexity.

High bandwidth data transport: We shall be able to transport at near line rate on at least 100 GbE connections. Later scaling to 200 and 400 GbE.

Direct access to GPU memory: We would like to support direct data transfer from and to GPU in order to reduce the load on the CPU.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101093934

Reliable data transport: The application shall not lose any data in the transfer between nodes.

Implementation complexity: The details of data transport shall be abstracted away for the user in order to reduce the system complexity. Current available frameworks for high bandwidth data transport in data centers should be a good fit to this use case.

Table 4 shows the score for this use case for available technology. Both MPI and Holoscan are selected for evaluation. MPI is well established technology in the data center with a wide rang of support, Holoscan is promising new technology.

Requirement	UDP Sock.	TCP Sock.	ibverbs	UCX	MPI	Holoscan
High bandwidth data transport	++	-	++	+	+	+
Direct access to GPU memory	-	-	-	++	++	++
Reliable data transport	-	++	++	++	++	++
Implementation complexity	+	+	–	-	++	++

Table 4: Technology mapping on inter back-end requirements

4.2 Network multicast

The multi-cast use case is an extension of the distributed radio telescope systems use case. In this specific case it needs to be supported to receive the data from the front-end on multiple back-ends without duplication of the data (at the front-end).

DPDK can be combined with multicast as it is based on UDP data transport.

RoCEv2 configured for the Unreliable Connection with the WRITE operation, does not support multicast. However, the Unreliable Datagram transport service would allow to send the front-end data to multiple back-ends. As noted before, the UD transport service is more complex to implement and loads the CPU more than the UC transport service.

This makes DPDK the most suitable implementation for multi-cast, but RoCEv2 UD might be evaluated in the project as alternative.

4.3 VLBI correlator

The VLBI use case is very broad, covering bandwidth from 1 Gbit/s to 300 Gbit/s per station and a number of stations between 4 and 32. We evaluate the various transport APIs based on the following requirements:

High bandwidth data transport: we should ideally be able to saturate the line speed on the receiving nodes while simultaneously sending data from up to 32 data storage nodes with fair sharing between the senders.

Direct access to GPU memory: we would like to transfer data from data nodes into GPU memory without significant CPU load to keep power consumption low and/or have the ability to use the CPUs in the systems for tasks that can't be efficiently done on the GPU.

Reliable data transport: the application should not lose any data in the transfer between data storage nodes and correlation nodes.

Ease of programming: we want to spend as much time as possible on optimizing the code that implements the VLBI correlation algorithms. We definitely do not want to optimize the data transport code for a specific correlator configuration. Technologies that provide support for distributed applications are preferred.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101093934

Vendor neutrality: we prefer a technology stack that can be deployed on a wide variety of hardware without vendor lock-in.

Requirement	UDP Sock.	TCP Sock.	ibverbs	UCX	MPI	Holoscan
High bandwidth data transport	++	-	++	+	+	+
Direct access to GPU memory	-	-	-	++	++	++
Reliable data transport	-	++	++	++	++	++
Ease of programming	+	+	–	-	++	++
Vendor netrality	++	++	–	-	++	-

Table 5: Technology mapping on VLBI requirements

Based on the scoring given in Table 5, MPI clearly is the most promising technology, but Holoscan may be worth considering as well if it provides better scheduling of data transfers.

4.4 Reliable large data transfer in an image inspection system

In this section the technologies described in section 3 are evaluated for the use case "Reliable data transfer in an image inspection system" as described in chapter 2.4. To do so, the key requirements are identified and mapped on the available technologies. From a functional perspective, all technologies described in Sections 3.3 and 3.4 can be used for transferring data from the processing pipeline to the GPU node. However, when requirements such as system scalability, system cost, and development cost are taken into account, some of the technologies gain preference. Since the system will make use of different types of implementation for the image pipeline and the GPU nodes, the selection criteria can differ for each type of edge node. For example, the pipeline node will make use of a smart NIC, where the GPU node is preferably a standard off-the-shelf solution.

Requirement	Infiniband	UDP	RoCEv1	RoCEv2	iWARP	DPDK
High bandwidth data transport	++	++	++	++	++	-
Reliable data transport	++	-	++	++	++	+
Processor load	++	-	++	++	+	-
Scalability	+	++	-	++	++	++
Total Cost of Ownership	-	+	+	+	-	+
Availability	-	++	-	+	-	+

Table 6: Technology mapping on requirements

High bandwidth data transport or data throughput should be sufficient and will be around 100 Gbps for the targeted system. This bandwidth will grow in future systems, therefor system scalability is also an important technology driver. All technologies can support high-bandwidth applications. Standard UDP traffic would have a preference due to the maturity and large-scale adaptation in standard network technology. Lots of COTS solutions are available from a large number of vendors.

Reliabilty is an important requirement for the Image processing system. All protocols, except UDP, do have an option for error detection and re-transmission. The iWARP technology is TCP/IP based and therefore, re-transmission on error detection is part of the technology. PFC is a standard network protocol for congestion control, available in the RoCEv2 technology by

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101093934

default. PFC can also be part of a UDP implementation, however, this is not as integrated in the technology as for RoCEv2.

Processor load, since the receiving edge node will process the data on a GPU, smooth integration of the data transfers into a GPU platform is required to allow optimal performance usage. Protocols supporting direct writes to application memory are preferred. UDP implemented in SW is not an option, since it will take a heavy load on a CPU, as described in 3.4.1 UDP.

Scalability, besides the already described bandwidth scalability, the system should be able to scale in the number of processing pipelines and GPU Nodes. The end nodes shall not be impacted for different system sizes. The impact should only be limited to the number of endpoints and the network setup. In principle all technologies can support this requirement, however, RoCEv1 will have limitations in the network scalability since it only supports a layer 2 network. This limits the number of nodes on a network.

Total Cost of Ownership, For cost all cost drivers are evaluated, development cost, system components cost. For the Infiniband technology, only one vendor is supporting this option, which might lead to a vendor lock-in with a negative impact on the cost. The UDP technology is widely adopted for all kinds of network applications and lots of vendors, free cores and designs are available. Concerning system cost this option will be the lowest, however due to adding all options to UDP needed for the system, the development cost will be the highest. Since the system will contain a smart NIC also, the development cost for iWARP is significant since there is only a single iWARP-RDMA core available. A GPU node can make use of an off-the-shelf GPU solution and this reduces the development and system cost significantly.

Availability, Long time availability of technology is important since imaging systems will be available and evolve over time and the network technology should remain available over time or evolve gracefully. Since Infiniband is a single-vendor solution, RoCEv1/2 are introduced to move Infiniband to standard Ethernet. Standard UDP has proven to be a technology that is mature and widely adopted. This technology is not likely to disappear in the near future. The introduction of RoCEv2 as the successor of RoCEv1 proves a graceful evolution of the ROCE protocol. It is likely that RoCEv2 will take over all RoCEv1 implementations.

Considering all these points, the targeted technology for Image processing will be RoCEv2.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101093934

5 Initial Evaluation of Technology

The DAS6 cluster [23] is a research cluster at ASTRON which allows well for experimental system configurations. Both DPDK, RoCEv2 and MPI are tested in a small set-up in this system.

5.1 DPDK

We already implemented a rudimentary correlator that demonstrates the use of DPDK in a GPU correlator. This correlator is by no means complete, but it demonstrates the high-speed receive path through DPDK, and the packet payload handling by the GPU. The demonstrator is capable of receiving network packets at line speed on a 200 GbE network interface without packet loss, and to stream the packet payload into the GPU memory of an NVIDIA A100 GPU for further processing (the filter is still a dummy filter, but the use of the GPU tensor-core correlator library already works). The demonstrator shows that the DPDK correlator concept works, but is quite difficult to implement, and requires careful tuning of many parameters to avoid packet loss.

Somewhat similar, we also demonstrated this concept on an NVIDIA Jetson AGX Orin. This is a small (11x11 cm^2 System-on-Module for edge computing (i.e., in the field instead of a data center). The beauty of this device is that the CPU and GPU are tightly integrated, and share the same memory. We can receive data up to 100 Gb/s, which is a very good result for such a low-power device. The whole system draws only 61 Watt to run the DPDK demo correlator, of which 20 Watt is used by the 100 GbE network interface. As on this device, there is no distinction between CPU memory and GPU memory, the packet receipt work differently as described above (there is no need to split packet headers and packet payloads). Again, it turned out to be very difficult to tune both some system parameters and the application software to achieve this result — DPDK is efficient, but not simple to use.

The ultimate goal is to demonstrate the DPDK correlator on the NVIDIA Grace Hopper Superchip. This is a highly innovative System-on-Module for data centers. The module combines a powerful 72-core CPU (ARM Neoverse v2) with the new Hopper H200 GPU, the most powerful GPU built to date. The innovation is (like the Orin described above) the tight connection between CPU and GPU, which provides 14 times more bandwidth than previous-generation GPUs like the A100. This eliminates the common PCIe bottleneck that limits GPU performance for basically all our radio-astronomical applications.models. Grace Hopper systems typically have three PCIe gen5 slots available for 400 GbE network interfaces, for a total of 1.2 Tb/s of network I/O bandwidth. We just received two of such systems, and plan to experiment if we can push such systems to filter and correlate 1.2 Tb/s of input data in practice (or determine the bottleneck that would prevent us from reaching such a high bandwidth).

5.2 RoCEv2

In prior work [3] we explored different configurations for the RoCEv2 protocol. Within the limitations of the DAS6 system we simulated system configurations for different communication scenarios: one-to-one, many-to-one and one-to-many. These three different scenarios are shown in Figure 18. Note that these tests are done from CPU to CPU and CPU to GPU, in a real system the sender will be FPGA based. In the last part of this section we also describe a test with an FPGA as sender.

In order to test the different system configurations a test application and RDMA API and profiling tool were developed. The implementation facilitates the registration of GPU memory to RoCE so that the NIC can read from and write into GPU memory without the involvement of

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101093934

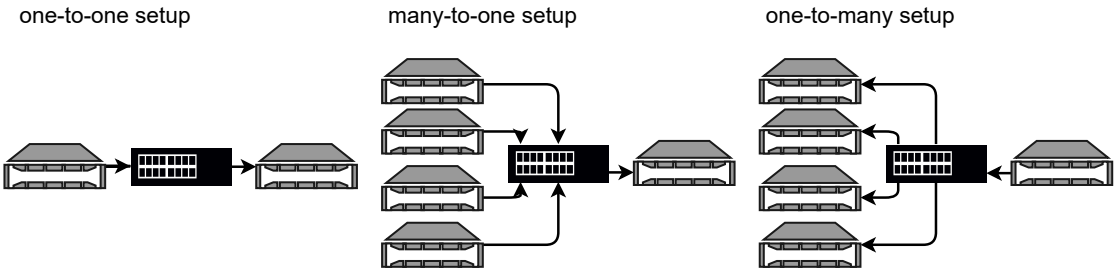

Figure 18: System configurations for different communication scenarios

the main memory or CPU. The RDMA API uses ibVerbs to query, create and modify all relevant RoCE attributes. The API also provides transport methods to the test application, which posts RoCE work requests in a separate thread to the NIC and checks the resulting work completions. The measurement method of, for example, the CPU utilisation and goodput is implemented in the RDMA API and not in the profiling tools. This is because the profiling tools include more generally usable measurement methods to, for example, read out NIC counters and measure and track function calls. A more elaborate description of these components can be found in [3].

The **one-to-one** setup, focuses on identifying the effect of different parameters between two NIC using a single connection. Among others, we explore settings such as message size, page sizes and reducing notifications in the receiver by using solicited events.

We observed that CPU utilisation decreases as the message size increases. From this, we concluded that a 16kiB message size was already capable of achieving a goodput of 98Gbps with a CPU utilisation of 75%. For larger message sizes, for example 1MB, we observe up to 200Mbps higher goodput and a CPU utilisation down to 5%. Testing with different memory page types showed no effect on the settings used. However, we note that this might affect the performance when the memory blocks become very large or when the memory is used in real-time application, requiring reconsidering the use of huge pages.

Furthermore, we assessed the use of linked work requests and solicited events. Linked work requests have more impact on requesters than responder QPs, and are necessary (in our implementation) for small message sizes (<16kiB) so that the queue is quickly refilled to avoid goodput degradation. Solicited events, decrease the CPU utilisation of the responder. In the case of 4kiB and an interval of 100 messages per solicited event, the reduction in CPU utilisation was 52% compared to no solicited events, an interval of 25 messages per solicited event and 16kiB message size achieved an even higher reduction of 75%.

We concluded the one-to-one setup by examining the increase of QPs, which resulted in a considerable dissimilarity between using one thread per QP and a single SRQ. The multi-threading method achieved a goodput of about 98Gbps for both 4kiB and 16kiB with 80% and 22% utilisation, respectively, when using 500 QPs. In contrast, the SRQ with 4kiB achieved only 60Gbps. At the same time, the 16kiB with SRQ gained a reduction in CPU utilisation of 50% and 30% for 100 and 500 QPs, respectively, compared to the equivalent multi-threaded configuration.

The **many-to-one** topology is used to study the scalability of RoCE. The aim is to simulate radio telescope systems distributed and scalable aspects using the available cluster. This test builds further on on the parameters determined in the one-to-one set-up.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101093934

A test with this set-up, has proven that a configuration using an SRQ with up to a total of 2000 QPs in the responder and 16kiB messages can transport a goodput of 90Gbps from four requesters to one responder.

The **one-to-many** topology is used to analyse the scalability of a sender.

This set-up demonstrated that RoCE is capable of sending data at 98Gbps from 1 requester to four responders using 400 QPs. At 2000 QPs, we observed limitations which are most likely related to our implementation (the amount of threads and workload) and not to a fundamental limitation in the NIC.

In conclusion with the three different test set-ups we showed that with the RoCE protocol with Unreliable Connection service and the WRITE operation, we are able to reach good performance, both in goodput as well as CPU utilization. However, the performance achieved is sensitive to the chosen parameters. Though the test set-up was limited in scale, we argue that RoCE should be able to satisfy this application area and will evaluate this technology further at larger data rates.

DMA to GPU With the one-to-one and many-to-one set-ups we tested receiving the data on GPU memory directly next to the set-up with CPU to CPU memory transport. We confirmed that we receive the data in GPU memory with similar performance as the CPU memory, during a short interval. However in the scope of the tests, we were not able to take processing of the data on GPU in to account, which might affect the performance.

FPGA as a sender In addition to the CPU to CPU set-ups we evaluated a set-up with an FPGA as sender. As introduced in Section 4.1.1 the Coyote framework was selected and adapted to evaluate data transport from an FPGA to NIC over the RoCE protocol. Coyote is a framework that offers operating system abstractions on FPGAs, including memory and network services such as TCP/IP and RoCE. The framework has its own driver and C++ API so that, among other things, it has direct memory access to the server's main memory. We use two different set-ups:

FPGA to FPGA communication: We used a set-up with an Alveo FPGA in the HACC system from ETH Zürich where the FPGA sends RoCEv2 packets over a 100 GbE network to a second FPGA. At the time of testing there was no CPU node with RDMA capable NIC connected to the same 100 GbE network. We achieved a goodput of 20.4Gbps between the two FPGAs. This is significantly lower than the line rate. Due to limitations in the network configuration it was not possible to use Ethernet frames larger than 1500B. This configuration could not be adapted. Consequently the largest possible RoCE message size was 1024B, resulting in large overhead and low goodput.

FPGA to NIC communication: A follow-up test was done in a set-up in DAS6 with the Coyote framework on an Alveo FPGA transmitting data over a 100 GbE network to a CPU node with RDMA capable NIC. In this test we were not able to receive data on the system because at the time the RoCEv2 implementation on the FPGA did not follow the RoCEv2 specifications strictly enough and the NIC rejected the RoCEv2 packets. However, we could identify that UDP data was transported from the FPGA through the switch to the NIC at near 100 GbE line rates. Note, that since we last worked with the Coyote framework both the framework and stand alone network stack have been improved upon.

Despite complications at the time of testing, this evaluation has provided sufficient insight in how to create an implementation of RoCEv2 on FPGA. At the same time, we did not find an open implementation of RoCEv2 that is both applicable to our use case and is free and open to use and adapt. Within the scope of this project we have set-out to develop our own implementation optimized for UC WRITE with immediate data, targeting the Intel Agilex 7 FPGA.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101093934

5.3 MPI

Several MPI implementations are provided on the DAS6 cluster. We looked at the Open MPI implementations that were provided. Some of those were built with GPU (CUDA) support and some were built with UCX. But unfortunately none were built with both GPU and UCX support. The version of Open MPI that did include GPU support was built with ibverbs support. Unfortunately, with this library it was only possible to use a subset of the network interfaces in the cluster due to limitations in the pre-UCX code that made it impossible to use RoCE with both mlx_4 and mlx_5 interfaces. Some preliminary benchmarking was done with a toy MPI application that simulated a VLBI correlation with multiple data sources. These benchmarks indicated that this solution may not be as scalable as desired. With one or two senders the aggregate data rate on the receiving node was close to the bandwidth of the network interface. But when the number of senders was increased the aggregate bandwidth was reduced significantly. By adding synchronization between senders and receivers we were able to mitigate this somewhat, but the aggregate bandwidth still went down when the number of receivers was increased. We are planning to connect the existing VLBI storage nodes at JIVE to a new RADIOBLOCKS cluster that is being built. This will allow a more thorough investigation of the scalability of the different solutions.

RADIO BLOCKS

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101093934

6 RADIOBLOCKS implementations and demonstrators

This chapter describes the high bandwidth data transport technology that we aim to implement and demonstrate within the RADIOBLOCKS project. The technology evaluation for RADIOBLOCKS (and the design of the RADIOBLOCKS cluster) as described in Deliverable 4.1 is taking the required supporting technology in to account. The software and firmware components developed for the technology evaluation will be generalized as much as possible, seeking harmonization with other system configurations. We strive to make implementations available to others under an open source license.

6.1 Distributed Radio Telescope Systems

We aim to evaluate and implement DPDK as well as RDMA based high bandwidth data transport over Ethernet. Both might lead to a feasible solution for up to (and scaling beyond) 400 GbE data transport. However, both technologies have different advantages and disadvantages. While aiming to demonstrate the technology at 400 GbE line rate, we note that this depends on the availability of hardware, especially the network equipment (NIC and switches) is currently hard to obtain. Alternatively we might combine multiple 200 GbE lines to achieve a combined 400 Gbps data rate, as this technology is more widely available. The load on the back-end is deemed to be similarly challenging for 2 x 200 GbE as with 1 x 400 GbE.

Because DPDK does not require a special front-end (FPGA) implementation, the sender might be replaced by another node with DPDK to generate data. This makes it easier to build a larger scale demonstrator than with RDMA, as well as to integrate with current existing systems. With the DPDK technology we aim to demonstrate the signal path from the digitizer in the front-end up to and including correlation of the received signals in the back-end for a wideband system (several very high bandwidth front-ends to one or several back-ends) in two separate demonstrators: 1) we generate data for many front-ends and send the data to a single node in the back-end, demonstrating that we can receive and process an aggregated data rate of approximately 400 Gbps at the back-end; 2) we aim to integrate with a single front-end for a wideband system, in development at UBx, as well as a back-end application, representative for a wideband system, to demonstrate the signal path from digitizer to correlator.

The development and demonstration of RoCEv2 (RDMA) requires more specialized equipment (limited available FPGA development kits). We therefore aim for a smaller set-up where data is transmitted from 1 or 2 front-ends (FPGA) to 1 back-end at 400 GbE (either single line 400 GbE or multi line 200 GbE). Optionally, at a later point in time integrating with the previously described full signal path demonstrator, replacing the DPDK components by RoCEv2.

The two technologies will be evaluated on the achieved performance in terms of throughput, CPU load, energy consumption and implementation complexity.

The distributed radio telescope system inter back-end use case will tail along with the VLBI correlator use case.

6.2 Network multi-cast

We aim to evaluate the DPDK implementation for its capabilities to support the multi-cast use case. This might be achieved by extending the above described DPDK demonstrator with a switch and multiple nodes that subscribe to the same data stream.

RADIO BLOCKS

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101093934

6.3 VLBI correlator

We intend to develop the VLBI correlator as an MPI application as this is the most widely available distributed computing framework available on typical VLBI correlator systems. The intention is to also use asynchronous tag-based MPI messaging APIs for bulk data transfers, as this will allow us to transparently transfer data directly into GPU memory using RDMA technologies like RoCEv2. This is important as we are developing GPU compute kernels to accelerate the most compute-intensive parts of the VLBI correlation algorithm. However, we keep our options open to use a lower-level API for the bulk data transfer if it turns out the asynchronous tag-based MPI messaging APIs don't give us the required data transfer rates. In that case we will use MPI just for communication to set up the bulk transfer between the different nodes that run the application.

For the VLBI correlator demonstrator we intend to connect the existing VLBI storage nodes at JIVE to the RADIOBLOCKS cluster with one or two 100 Gbit/s network links and correlate a typical EVN observation and establish to what extent data transport affects the correlation speed (provided we do not saturate the available network bandwidth on the links between the storage nodes and the RADIOBLOCKS cluster). For this purpose we will compare correlation speed with and without RDMA (RoCEv2) and direct GPU memory access.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101093934

References

[1] Chris Broekema. "Commodity compute and data-transport system design in modern large-scale distributed radio telescopes". In: *PhD thesis, Vrije Universiteit Amsterdam* (2020).

[2] Bruce Merry. "Efficient channelization on a Graphics Processing Unit". In: (2023).

[3] Willem de Laat. *The Application of RDMA over Converged Ethernet Data Transport for Radio-Astronomy Systems*. Dec. 2022. URL: http://resolver.tudelft.nl/uuid:94d16d94-e4c7-4a54-a26a-af114db02dd7.

[4] John Alexander Forencich. "System-Level Considerations for Optical Switching in Data Center Networks". PhD thesis. UC San Diego, 2020. URL: https://escholarship.org/uc/item/3mc9070t.

[5] AMD. *Open NIC source repository*. URL: https://github.com/Xilinx/open-nic.

[6] SKA. *SKA low-cbf repository*. URL: https://gitlab.com/ska-telescope/low-cbf/ska-low-cbf-proc.

[7] Damien Gratadour et al. "Prototyping AO RTC using emerging high performance computing technologies with the Green Flash project". In: July 2018, p. 45. DOI: 10.1117/12.2312686.

[8] Niklas Schelten et al. "A High-Throughput, Resource-Efficient Implementation of the RoCEv2 Remote DMA Protocol for Network-Attached Hardware Accelerators". In: *2020 International Conference on Field-Programmable Technology (ICFPT)*. 2020, pp. 241–249. DOI: 10.1109/ICFPT51103.2020.00042.

[9] Fritjof Steinert et al. "Hardware and Software Components towards the Integration of Network-Attached Accelerators into Data Centers". In: *2020 23rd Euromicro Conference on Digital System Design (DSD)*. 2020, pp. 149–153. DOI: 10.1109/DSD51259.2020.00033.

[10] Wassim Mansour, Nicolas Janvier, and Pablo Fajardo. "FPGA Implementation of RDMA-Based Data Acquisition System Over 100-Gb Ethernet". In: *IEEE Transactions on Nuclear Science* 66.7 (2019), pp. 1138–1143. DOI: 10.1109/TNS.2019.2904118.

[11] Xizheng Wang et al. "StaR: Breaking the Scalability Limit for RDMA". In: *2021 IEEE 29th International Conference on Network Protocols (ICNP)*. 2021, pp. 1–11. DOI: 10.1109/ICNP52444.2021.9651935.

[12] Andrew Ansor. *SKA RDMA receive repository*. URL: https://gitlab.com/ska-telescope/rdma-data-transport.

[13] Dario Korolija, Timothy Roscoe, and Gustavo Alonso. "Do OS Abstractions Make Sense on FPGAs?" In: *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation*. OSDI'20. USA: USENIX Association, 2020. ISBN: 978-1-939133-19-9.

[14] ETH Zürich. *Scalable Network Stack for FPGAs (TCP/IP, RoCEv2) repository*. URL: https://github.com/fpgasystems/fpga-network-stack.

[15] ETH Zürich. *Coyote framework repository*. URL: https://github.com/fpgasystems/Coyote.

[16] Mario Ruiz et al. "Limago: an FPGA-based Open-source 100 GbE TCP/IP Stack". In: *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*. Nov. 2019, pp. 286–292. DOI: 10.1109/FPL.2019.00053.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101093934

[17] David Sidler et al. “StRoM: Smart Remote Memory”. In: *Proceedings of the Fifteenth European Conference on Computer Systems*. EuroSys ’20. New York, NY, USA: Association for Computing Machinery, 2020. ISBN: 9781450368827. DOI: 10.1145/3342195.3387519. URL: https://doi-org.tudelft.idm.oclc.org/10.1145/3342195.3387519.

[18] Tianli Song. *RoCE based 100Gbps RDMA network stack on FPGA hardware*. Nov. 2021. URL: https://repository.tudelft.nl/islandora/object/uuid%3Abdc8259e-43e7-4e81-b573-c2f8c1442892.

[19] Grovf Inc. *RNIC use-cases with FPGA based Smart NIC*. URL: https://grovf.com/products/grovf-rdma.

[20] AMD Inc. *Resource Utilization for ERNIC v4.0*. URL: https://download.amd.com/docnav/documents/ip_attachments/ernic.html.

[21] Wael Farah. *GNU Radio and the Allen Telescope Array*. Nov. 2023. URL: https://casper.astro.berkeley.edu/workshop2023/agenda/presentations/day1/3_WF.pdf.

[22] J. Manley et al. *SPEAD: Stream protocol for exchanging astronomical data*. Jan. 2010. URL: https://casper.astro.berkeley.edu/wiki/SPEAD.

[23] Henri Bal et al. “A Medium-Scale Distributed System for Computer Science Research: Infrastructure for the Long Term”. In: *Computer* 49.5 (2016), pp. 54–63. DOI: 10.1109/MC.2016.127.

This project has received funding from the European Union’s Horizon Europe research and innovation programme under grant agreement No 101093934