The Application of RDMA over Converged Ethernet Data Transport for Radio-Astronomy Systems

by

W.M. de Laat

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Tuesday December 20, 2022 at 13:30 PM.

Student number:	4587960	
Project duration:	February 15, 2022 – December 20, 2022	
Thesis committee:	Dr. Ir. Z. Al-Ars,	TU Delft, supervisor
	Dr. Ir. R. Venkatesha Prasad,	TU Delft
	Ir. S. van der Vlugt,	Astron
	Dr. Ir. J.J. Hoozemans,	Voltron Data

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Abstract

The need to receive and process higher data rates in computer clusters is an ever-increasing trend. This also applies to radio-astronomic systems, which have become more distributed over the past decades, increasing data traffic between antennas and processing facilities. At the antenna, Field Programmable Gate Arrays (FPGAs) digitise the radio signals and often perform the first stage of signal processing at the antenna. Hereafter, the data is sent from the FPGAs to computer clusters, where further processing is accomplished on CPUs and GPUs. Currently used protocols for data transport between FPGAs and CPUs, such as UDP, are insufficiently scalable for higher data rates since these heavily load the receiving CPU.

Remote Direct Memory Access (RDMA) over Converged Ethernet (RoCE) was developed to overcome the disadvantages of the standard UDP and TCP protocols by using the CPU to orchestrate data transfers and not engaging the CPU in the data path. With this, the data bypasses the CPU, lowering the CPU workload and enabling throughput and latency improvements. However, the data transport methods for applications in radio-telescopes must meet specific characteristics of this application area, such as public Ethernet routing, high sustained data rates, real-time processing and implementable on FPGAs.

This work examines whether RoCE can reduce system load and increase throughput in sending and receiving antenna data. For this purpose, we characterise the data transport, derive the best RoCE configuration for the intended application and asses whether RoCE can achieve the required performance. The protocol analysis concluded that an unreliable connection transport service with RDMA WRITE with immediate data is best suited for the application in radio telescope systems.

This thesis defines a methodology to examine the impact of RoCE settings and RoCEs scalability on a representative cluster setup with CPU-CPU and CPU-GPU data transport. To conduct these tests, an application is developed with abstractions such that it can be easily reused. Our RoCE application is available on GitLab [29]. First, standard tooling demonstrated that RoCE achieves ~2x higher goodput and 3x lower CPU utilisation compared to UDP, indicating the possible scale of performance improvement of RoCE. Further performance studies are accomplished through the custom-developed application to explore various settings and network topologies (1-1, 1-N and N-1). For example, we found that using a shared receive queue can reduce CPU utilisation by 50%, and the use of solicited events can yield reductions of up to 70%, with no negative impact on resources and goodput. Direct memory access from the RoCE-enabled NIC to GPU memory is also evaluated, for which comparable performance was achieved to standard main memory in an N-1 setup. We found that RoCE can transport data from multiple transmitters over a total of 2000 QPs with a 16kiB message size to a single receiver at 90Gbps and a CPU load of 40% for one core in the receiver.

The feasibility and performance of transporting data between FPGA and RNIC are also investigated. The implementation could not transport the data from the FPGA into the CPU memory because of an incorrect checksum implementation. Nevertheless, we were able to confirm that it is possible to implement RoCE on an FPGA for use in radio astronomical data transport.

Preface

I would like to take this opportunity to express my sincere gratitude to those who have helped me along my journey. First and foremost, I thank Steven van der Vlugt for his supervision during this thesis. I greatly appreciated his kindness, passion for technique and willingness to help me with all my questions. Furthermore, I am deeply grateful to Zaid Al-Ars for his openness, positivity and yet critical outlook through which he supervised my project.

I would also like to thank ASTRON for this great opportunity. My sincere thanks to all ASTRON employees who have shared their knowledge and expertise with me. In particular, I would like to thank John Romein for his knowledge of radio telescope systems and high-performance computing systems. I would also like to thank Chris Broekema (ASTRON) and Joost Hoozemans (Voltron Data) for their perspectives, suggestions and critical questions during the project.

For the support with the FPGA implementation, I would like to express my profound gratitude to Dario Korolija and Javier Moya from ETH Zurich Systems Group. Their technical expertise and help in troubleshooting the various challenges have been of great value.

Besides the people who helped with my graduation project and thesis, I also want to thank others who made my studies a wonderful experience. The Electrotechnische Vereeniging (ETV) has been with me from the very first day. The ETV brought me friends, study buddies, parties and career opportunities. A community of amazing people who greatly enriched my college years. It has been an honour to be part of the 147th board. for this, I will be forever grateful to the ETV and everyone who was a part of it. Furthermore, a shout-out to the fellas of the squadpack. Sharing our ups and downs for over a decade now, with so many great and marvellous moments, and many more to come!

Last but certainly not least, I want to thank my family for their unconditional love and support. Your words of encouragement and unwavering belief in me have kept me going. I want to thank my parents for all the opportunities I have had and their commitment. They encouraged me in pursuing my passion for engineering, and their pride in their ambitious and inquisitive children contributed to making this succeed. A solid foundation has been built for a future of even more opportunities.

W.M. de Laat
Delft, December 14, 2022

Contents

List of Figures

List of Tables

Acronyms

COTS Commercially of-the-shelf.

CPU Central Processing Unit.

CQ completion queue.

CQE completion queue element.

ECN Explicit Congestion Notification.

EMI Electromagnetic Interference.

ETS Enhanced Transmission Selection.

FPGA Field Programmable Gate Array.

GPU Graphics Processing Unit.

NIC Network Interface Card.

NUMA Non-uniform memory access.

PFC Priority Flow Control.

QP queue pair.

RDMA Remote Direct Memory Access.

RNIC RoCE enabled NIC.

RoCE RDMA over Converged Ethernet.

SRQ shared receive queue.

TC Traffic Class.

UDP User Datagram Protocol.

WAN Wide-Area Network.

WQE qork queue element.

WR work request.

1 Introduction

In the last century, many discoveries have been made in the universe through astronomy. Together with optical telescopes, radio telescopes are an essential tool for astronomers. A radio telescope observes (radio) waves outside of the visible spectrum. However, for even more groundbreaking discoveries, we need better and better equipment to see more details of the universe and be able to look further into the universe than ever before. To achieve high sensitivity and high resolution, one would need a very large (100 square meters to several square kilometres) collecting area. These kinds of systems are very costly and impractical to build. An alternative is to build phased array and inferometry systems consisting of many small antennae spread out over a large area. The signals of these antennae are computationally combined into one large antenna.

Such a system is cost-effective and flexible but requires a powerful computational back-end. Data has to be streamed in real-time from the antennae to a central location where the data is combined through filtering, correlation and beamforming. Today's computers are well suited to implement the computations for large radio telescopes [9], the main bottleneck is currently in the (efficient) transport of data from the antennae to a central location. The increasing data traffic is not only a desire and a problem in radio astronomy but also in many other sectors such as particle physics [10] (Nikhef, CERN), other distributed sensor networks, applications in edge computing [21], but also within data centres [11, 47] (e.g. moving workload between Field Programmable Gate Arrays (FPGAs) [17] and Graphics Processing Units (GPUs) [18]).

Radio telescopes produce a lot of data since the radio signals originated from space are measured with a large number of sensors (20-10000+), with sensors operating at very high sample rates (20 MHz up to 70 GHz) or a combination of both. Due to physical constraints and data arrangement, the processing cannot occur at the telescopes themselves and requires a central facility where a supercomputer or computational cluster performs the calculations. Some essential data transport characteristics in this kind of application are the single direction in which the data is transported, from the antenna to the computation facility, and the static network and routing topology. In addition, this data transport involves a long-term stable volume of data since an astronomical survey can take days or weeks.

Current transport implementations are based on UDP / IP transmission over (partially public) Ethernet. On Central Processing Unit (CPU) nodes, this is implemented in a software stack that covers; a driver, a Linux kernel and user space. The communication data is copied multiple times between these layers to achieve separation and protection, consequently imposing a significant load on the CPU. For large Ethernet data rates, this leads to significant inefficiency in CPU utilisation or might even become a bottleneck in the application.

Remote Direct Memory Access (RDMA) technology has been developed to overcome this problem by obtaining data from another host's memory directly without unnecessary overheads due to data movement. With this technology, data movement is handled either at a low level in the operating system (e.g. Linux kernel) or directly in hardware in the network interconnect, thus drastically reducing the load on the CPU.

RDMA has already been used for many years in many supercomputers and other high-performance computing facilities, either over Infiniband or Ethernet. However, this is mainly done over internal data centre networks and not over public Ethernet. On top of that, in many radio telescope systems, FPGAs perform digitisation of the radio signals and often the first stage of signal processing at the antenna and send the data to the central processing facility. Therefore, a suitable data transport protocol must also be implementable on an FPGA.

In this work, we investigate and test if RDMA over Converged Ethernet (RoCE) is able to reduce system load and increase throughput in sending and receiving antenna data for applications in radio telescopes. Part of this work includes a representative test between CPUs and GPUs and between FPGAs and CPUs.

1.1. Context

ASTRON is the Netherlands Institute for Radio Astronomy and was founded in 1949. Their goal is to make discoveries in radio astronomy happen. They design, manufacture and operate various radio telescopes and systems needed to acquire astronomical data. In addition to their in-house activities, they also have close contact with other institutions and companies involved in astronomy at (inter)national levels.

ASTRON contributes to many scientific goals, such as observation and the life course of galaxies (clusters) and black holes. Furthermore, they also monitor the solar weather and the earth's ionosphere.

Astronomy allows us to look far into space to see, for example, how other planets move and develop. This allows researchers to estimate better what the life course of the Earth has been and will be.

In order to remain capable of groundbreaking research, they are constantly improving their instruments. New instruments and technologies must meet several tight constraints. The physical location, for example, provides limitations in size and energy budgets. At the same time, the environment imposes conditions on the required robustness due to the impact of weather and wildlife. In addition, these are often one-off large system designs expected to last for decades, making easy to carry out maintenance and reliability even more important. These aspects must be carried out within tight budgets for development, production and maintenance.

One of the most urgent challenges in radio telescope systems is to both increase the efficiency of data movement as well as to enable higher bandwidth between the (FPGA) receivers and (GPU) back-ends. This research into the operation of RoCE in radio astronomy was carried out in cooperation with ASTRON.

1.2. Problem statement

The main research question that this thesis aims to answer is as follows:

- *Can we use RoCE to reduce system load and increase throughput in sending and receiving antenna data for applications in radio-telescopes?*

To answer this question, we will study the RoCE protocol and examine its deployment in radio telescope systems. We do so by addressing the following sub-research questions:

1. What are the characteristics and requirements of applications for radio-telescopes?
2. Which configuration of RoCE is best for the intended application?
3. Can RoCE deliver the required performance and reliability at the scale of large distributed telescope systems?

Research contributions

To answer the above-stated research questions, we implemented and tested several solutions. We identify the data transport requirements for antennae data transport in radio telescope systems, as provided in Section 2.1. A test system was developed to simulate the environment and requirements between server nodes. The application is designed to scale to multiple server nodes in various topologies (one-to-one, many-to-one, one-to-many). Python-based scripts are designed to leverage a cluster job manager, *Slurm*, to automate design space testing and analysis. More discussion on this contribution can be found in Chapter 4. Via these contributions, we analysed the impact of various features offered by RoCE. Lastly, we tested and evaluated the RoCE protocol between an FPGA and a network card to study the feasibility of the protocol on FPGAs. This contribution is covered in Chapter 6.

1.3. Thesis outline

The remainder of the thesis is divided into five chapters:

Chapter 2 provides an introduction to the building blocks of a radio telescope processing pipeline and three radio telescope systems that describe the need for an improved data transport method. The second part

briefly introduces several RDMA technologies and a detailed discussion of the RoCE protocol. The chapter concludes with a discussion of related studies on the performance and scalability of RoCE.

Chapter 3 first describes the hardware and software components available and used. Then different measurement methods are discussed and chosen to measure the protocol adequately. Lastly, it details the distribution of the design space over different setups and configurations to reduce the number of tests without compromising the necessary insights.

Chapter 4 contains a complete account of the design and implementation of the RoCE test system and application. It starts with a global system overview, followed by a hierarchical approach to relevant application layers in subsequent sections.

In **Chapter 5**, the measurements of all configurations are analysed. Starting with a short analysis between UDP and RoCE followed by successive measurements using the following transmission topologies; one-to-one connection, many-to-one and one-to-many. In which the knowledge of previous measurements is used for more target-oriented subsequent measurements.

In **Chapter 6**, FPGA implementations of RDMA and RoCE protocols are discussed, after which the chosen implementation is briefly explained. In addition, this chapter discusses the methodology and results of this implementation, aiming to assess the RoCE transport protocol between an FPGA and a network card.

In **Chapter 7**, conclusions are drawn from the previous chapters. Furthermore, we present recommendations for future research regarding the scalability of RoCE, the application of RoCE in radio astronomy applications and the implementation of RoCE on an FPGA.

2

Background

This chapter gives background information on all relevant aspects and technologies related to the research in this thesis. Firstly, we introduce a common radio telescope processing pipeline, and three different radio telescope use cases relevant to our work in Section 2.1. We then address the data transport requirements for these use cases in Section 2.2. Hereafter, we discuss remote direct memory access technology in Section 2.3. In succeeding Section 2.4 provides a detailed explanation of the RDMA over Converged Ethernet (RoCE) protocol. In Section 2.5, we discuss which features of RoCE match the requirements for application in radio astronomy systems. Finally, this chapter concludes with a section presenting related work on evaluating RoCE features and scalability.

2.1. Use cases

First, we start with a discussion of the processing pipeline used in the different use cases, followed by three specific use cases that use this pipeline.

Radio telescope processing pipeline

To look better and further into space, we need increasingly sophisticated equipment. A classic way of receiving radio signals from space is through a dish antenna. To perform better measurements, the antenna must have a higher sensitivity. This can be achieved by increasing the size of the dish. However, there are physical and mechanical limits to the size of a dish. Another method to retrieve more detailed information from the universe is to use the signals of multiple antennae together. Combining the information from antennae is often very similar, but the implementation (techniques used) might differ per system. Therefore, only the primary components will be explained to capture EM signals and transform them into figures of the universe.

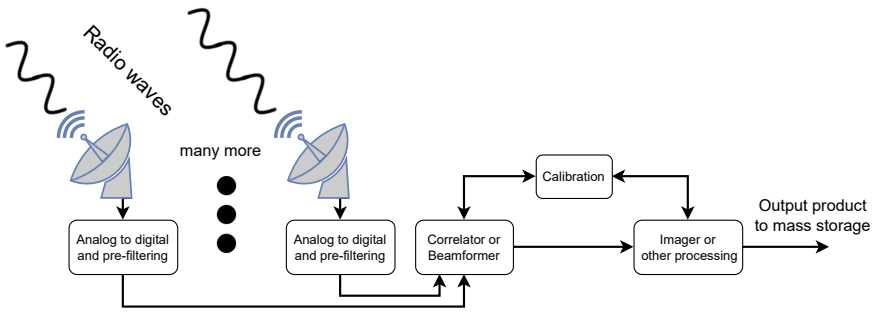


Figure 2.1: Radio telescope basic signal pipeline

Figure 2.1 shows a basic and simplified overview of a radio telescope processing pipeline. First, the EM signals emitted by sources in the universe must be captured by antennae, e.g. with dish antennae or dipole antennae. These antennae convert the EM signals (from space) into an analogue electrical signal containing both the signal of interest as well as Electromagnetic Interference (EMI) that primarily originates from earth.

The analogue signal is converted into the digital domain, where most of the filtering and processing is done. The conversion to the digital domain is done close to the antennae because the received signals of interest are very weak and thus very sensitive to interference from outside sources. The first digital stage is the filtering of the signal. This is not only to remove unnecessary signals but often also to divide the signal into different frequency bands for further processing.

After that, the data can be processed using either correlation or beam forming, and sometimes a combination of both. Beamforming creates one or more higher quality, thus better signal-to-noise ratio, signals. This is achieved by compensating the antenna signals for the phase shift between antennae where after they can be combined. This allows astronomers to look further into the universe at the cost of spatial resolution. The correlation method (also known as interferometry) applies cross-correlation to the antennae data. The result is a higher spatial resolution at the expense of sensitivity compared to beamforming.

Both processes deliver different products, but they have in common that the operations are performed on aggregated real-time streamed data. This real-time data originates from antennae in the field where the data should be as raw (un-processed) as possible. As a result, this process needs a lot of resources and high I/O bandwidth since it receives vast amounts of data from the antennae. Another critical stage in radio telescope processing is calibration. The environment around the antennae continuously changes, and so does the signal path of the EM waves from the universe. Hence the system needs to be tuned to gain optimal results.

The last stage is the most different for each survey. Here, much research-specific processing takes place; consequently, the end products will also be very different. Nevertheless, most output products are either an image or spectral information of the universe.

To further characterise radio telescopes, we will briefly describe three use cases at the extremes of radio telescopes here and in the following three subsections in more detail. The first use case is LOFAR, a small-band telescope system with many antennae sampled at a relatively low rate. The system contains many antennae grouped in stations. The throughput of individual antennae is relatively low, but the combined throughput of all antennae would be huge. Therefore the antennae data is already partially beamformed and filtered at station granularity, yet the resulting throughput of the system is still large. These antennae and field stations are distributed over a large area in Europe; therefore, some data traffic is transported over long distances over public Ethernet.

The second use case is AARTFAAC; this project uses a sub-set of LOFAR (576 antennae) and works directly on the raw antenna data instead of station data. The combined data rate is very large, and the radio telescope uses a private network.

The third and last use case is ALMA, which has a small number of wideband antennae (72). Each antenna generates a tremendous data rate from the receivers to the processing facility, which is transported over a private network. So, in this case, a small number of nodes will provide huge data throughput.

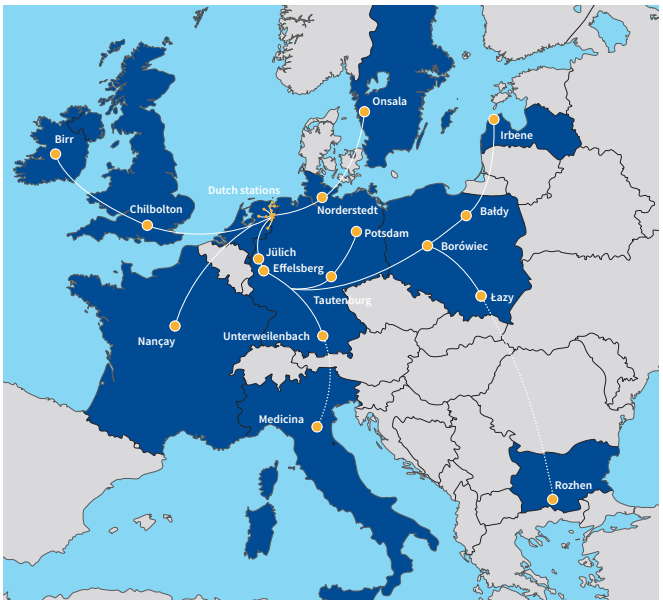


Figure 2.2: Map of LOFAR stations across Europe

2.1.1. Use case 1: LOFAR

ASTRON's LOFAR[1] (Low-Frequency Array) is currently the largest low-frequency radio telescope worldwide. LOFAR is a multipurpose sensor network with an innovative computer and network infrastructure that can handle large data volumes. LOFAR consists of many receiver stations spread out over the Netherlands and Europe, as shown in Figure 2.2. A top-level view of the LOFAR system is shown in Figure 2.3. These stations stream high volumes of data over private networks and public Wide-Area Networks (WANs) to a central data processor node (COBALT) in the Netherlands [9]. The LOFAR radio telescope enables astronomers to research, for example, the creation of the first stars and galaxies and cosmic magnetic structures [15].

Each station cabinet receives data from many antennae, which is filtered and pre-processed to the requirements set for the survey. In addition, the data is grouped by sub-frequency bands inside the station cabinets. After which, each frequency band is sent to the central correlator. The data is also pre-processed (filtered and beamformed) in the station cabinets due to the constrained bandwidth to the central compute nodes. Because the data is beamformed on a station level, only a small portion of the sky is observed. The digitisation of the antennae data and processing at the station level are both implemented in FPGAs. These FPGAs are configured once per survey, and a single survey can run for a couple of hours up to several days. The final step for the FPGAs in the station cabinet is to send the data over the Ethernet to the processor nodes using User Datagram Protocol (UDP) packets.

In the central correlator each processor node converts and stores the received Ethernet data through a software network stack. Because of this, the data gets copied between kernel space and user space memory several times before being processed on CPUs and GPUs.

It is essential to highlight that several characteristics make this system so unique. First of all, the data rate towards the central processor. In total, there are 126 FPGAs in station cabinets spread across western Europe that can each transmit 3Gb/s divided over a maximum of 488 sub-bands. This results in an aggregated bandwidth at the central processing node of almost 400Gb/s. The data is grouped and spread out over 24 CPU-GPU nodes at a subband granularity (each node processes several subbands). Besides, processing a single subband requires the subband data of all antennae, resulting in numerous connections between end nodes. Secondly, the data is partially transmitted over the public internet. As a result, the optimisations possible in a private, isolated network can not be done in this system.

Lastly, the data generated by the antennae and filtered by the FPGAs is so large that it is impossible to store and re-process the raw data. Therefore, it is impossible to retransmit data if something goes wrong during transmission. Even so, small amounts of missing data do not directly impact an observation's result. However, it is required to identify missing data so it can be accounted for in the processing pipeline. In addition, the incoming data on the central processor can only be buffered for a short time window. Consequently, the data has to be passed on to the GPUs for further processing as soon as possible.

To summarise, one of the main requirements for such a use case is the many-to-many single-direction data transmission. An FPGA must be able to transmit to various nodes (distribution of subbands over all nodes) at lower data rates. At the same time, a Network Interface Card (NIC) must be able to receive data from many FPGAs (126+, needs the data from all antennae per sample in a subband) that all transmit at low rates but combined, leading to a high data rate. Another requirement is that the data transport over public Ethernet must be possible, and it must be clear whether and which data has been lost during transport. A currently ongoing upgrade of LOFAR will increase aggregated data rates by 2-4x.

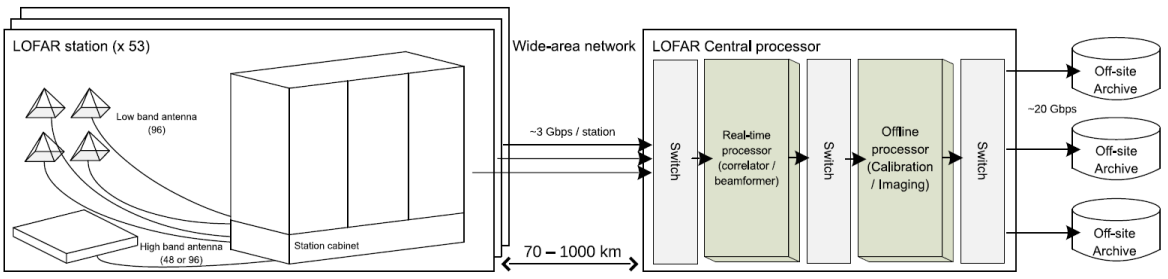


Figure 2.3: Top-level overview of the LOFAR system[9]

[1] https://www.astron.nl/telescopes/lofar

2.1.2. Use case 2: AARTFAAC

The AARTFAAC (Amsterdam-ASTRON Radio Transients Facility And Analysis Center) system is a real-time transient detector. It can quickly create images of the visible sky with a spatial resolution of 10 arcseconds. Such an image is created every second to discover transients in the universe.

Normally astronomers point their (radio) telescopes in the direction they want to observe. However, focusing on a single place is not convenient if one is looking for short events or transient objects. It offers a huge advantage to look everywhere simultaneously. The AARTFAAC radio system can scan the visible universe every second, looking for anomalies. As soon as something abnormal is detected, other radio astronomical instruments can take follow-up measurements in that direction with higher precision. In addition, this system can make a (relatively low resolution) image of the visible universe every second. These transient events provide insights into various astrophysical objects, such as emission mechanisms of jet sources and properties of intervening medium and pulsars [13].

AARTFAAC uses the raw (non-beamformed) data from 567 low-band antennae located at the core of LOFAR. These antennae are omnidirectional, meaning they can survey the sky without physically moving the antennae. Both LOFAR and AARTFAAC work in parallel and independently since the data is replicated after the digitiser in the station cabinets and sent to the AARTFAAC correlator. The system uses only a few subbands since the aggregated raw data rate of each subband of a low-band antenna would be too large to transport. The raw data is gathered in a central field station and sent over Ethernet to the processing facility located in Amsterdam. The current system is limited to an aggregated data rate of 100 Gbit/s out of a potential data volume of 1.5 Tbit/s. The future LOFAR system will expose all 2304 antennae of the LOFAR core and generate, when used at its full potential, 6 Tbit/s of aggregated raw data.

In summary, the AARTFAAC system leverages many antennae located at the core of LOFAR. The system uses a few subbands of the low-band antennae due to throughput limitations caused by raw antennae data transmission. Using RoCE technology to improve the throughput at the receiver side might enable to use more frequency bands and or more antennae in a future update of the system.

2.1.3. Use case 3: ALMA

The Atacama Large Millimeter/submillimeter Array (ALMA) radio telescope is an array of up to 72 antennae. The telescope is located in the Atacama Desert in Chili at 5000 meters above sea level. It creates the perfect conditions for observations in the GHz domain since it suffers from less pollution due to water vapour (dry air), low radio interference and scant clouds. Due to incredible engineering astronomers are now able to extensively research matters around black holes that are millions of light years away from us. In addition, the radio telescope also allows scientists to research the chemical composition of gas clouds inside various galaxies and around various planets [7].

Figure 2.4: ALMA antennae under the Milkyway, source: Sergio Otarola - ALMA [1]

The ALMA telescope array consists of a mix of twelve-meter diameter dish antennae and seven-meter diameter dishes spread across 15 square kilometres. Each antenna is currently equipped with eight receiver bands which will be increased to 10 in the coming years. When the system is fully operational, the antennae can detect signals from 950 GHz down to 35 GHz, corresponding to wavelengths of 0.32 mm to 8.6 mm, respectively. The system can be used in different configurations and will at a maximum use 72 dishes in a future

upgrade. The ALMA telescope has a much wider bandwidth and operates at much higher sampling frequencies than LOFAR. The larger frequency bandwidth and higher frequency require higher sampling rates, which drastically increases the data rate per antenna. In the next-generation ALMA system, a single dish is able to generate a data stream of up to 1.2 Tbit/s per receiver, with two receivers per dish.[2]

The analogue signal of each receiver (called the front end) is converted into a digital signal in the so-called back end located in each antenna. The digital signal is sent over optic fibres towards the correlator relatively close by. The correlator combines these signals and generates astronomical data sent to the operations support facility. The operations support facility houses several compute clusters that run multiple processing pipelines to compute relevant astronomical products.

2.1.4. Key differences and characteristics

First, one of the key differences between these systems is the network topology. LOFAR has many field stations transmitting at low rates (in the future up to 10Gb/s) towards several compute nodes. However, this is entirely different for ALMA, which transmits data at a very high data rate (in the future up to 2 times 1.2Tbit/s) from relatively a few antennae to a few nodes. Thus, a generic implementation must be capable of high transmission rates and receiving and sending with a large number of connections. This is different for AARTFAAC; this system's data rates and connections depend heavily on the technical capacities. In AARTFAAC, the raw antennae data needs to be transported, which increases the data rate per connection. On the other hand, AARTFAAC uses fewer subbands and antennae, reducing the overall data volume.

Generally speaking, the data rate per antenna in AARTFAAC is lower than ALMA but higher than LOFAR. In contrast to the number of connections which is higher than ALMA and lower than LOFAR. In conclusion, ALMA and LOFAR are the extremes regarding the throughput and the number of connections, respectively.

Besides, the network medium over which the data transport takes place differs. Both AARTFAAC and ALMA have the advantage that the transport takes place over private networks. This is not the case for a large part of the LOFAR system since it leverages public Ethernet.

Lastly, These systems must operate in real-time with a constant high data rate. As a result, it is impossible to store unprocessed data for an extended period; even for several seconds might be infeasible.

2.2. System requirements

We have derived the following common requirements from the above-listed use cases. These requirements will be listed below, as well as several non-functional requirements discussed during multiple interviews with engineers and researchers who work on enhancements of radio astronomy systems.

Functional requirements

These requirements describe the system's functioning (e.g., what the system should do). These must be measurable and ultimately can be answered with a clear yes or no.

R1 The setup must have an FPGA that sends packets and a NIC to receive packets.

R2 The system must be capable of transmission over a public Ethernet network.

R3 The protocol and devices must allow different network topologies such as 1-to-1, 1-to-N and N-to-N with at least 200 devices on either side.

R4 Must achieve a goodput of at least 90Gbit/sec, while multiple sender nodes transmit at lower rates towards a receiver node capable of 100Gbit/sec.

R5 The data must be transmitted via a connection without retransmissions because the system cannot re-transmit data.

R6 The system must be able to flag missing data such that this information can be used further down the processing pipeline.

[2] `https://www.almaobservatory.org/`

Non-functional requirements

These non-functional requirements are requirements that do not define the functional aspects of the system. For example, performance and costs-related requirements do not describe the system's functioning but impact the system's implementation. On top of that, they include requirements that can be answered subjectively and are, therefore, not measurable. Lastly, non-functional requirements might be less strict because some are not mandatory to meet to accept the solution. Our data transport solution for application in radio astronomy should meet the following non-functional requirements:

S1 Must be created with hardware and software Commercially of-the-shelf (COTS) products to reduce development complexity and costs.

S2 The solution must be future-proof, meaning that it is scalable to even higher speeds and more endpoints. Allowing upscaling of the system without large changes. (e.g. better hardware should increase performance without (large)changes to the code base)

S3 The solution should make use of mature technologies that can be expected to continue to be supported for a long time (with possibly minor adjustments).

S4 Open-source software, firmware and hardware are preferred to proprietary whenever possible.

S5 The RoCE protocol should be implemented in an abstract and reusable manner.

2.3. RDMA

In recent decades, more computing resources have been coupled together to tackle more extensive and diverse tasks. This has led to increased data sharing between compute nodes and the need for better interconnects between them. Improvements have been made in two technical fields. Firstly, new physical interconnects have been developed by various companies such as Infiniband, Slingshot, Intel Omni-path, and many other vendor-specific interconnects. The second improvement is the development of new protocols for easier and faster remote memory and storage access. One of the most known protocols to improve efficiency and throughput is the Remote Direct Memory Access protocol.

Remote Direct Memory Access allows one host to access the memory of another host or device not in the same physical system or chassis. Figure 2.5 shows the conventional way (via a TCP or UDP software stack) in which host 1 tries to retrieve data from the memory of host 2. Both host kernels are involved in the data transfer of each byte as it needs to copy the data from the user to the kernel space. The transfer flow for RDMA data transfers is shown in Figure 2.6. Both host kernels are only involved during the setup phase in which they establish a communication channel and transfer requests are communicated without context switches. As a result, the kernel involvement in the data transfer is drastically reduced and, therefore, the system load is also reduced.

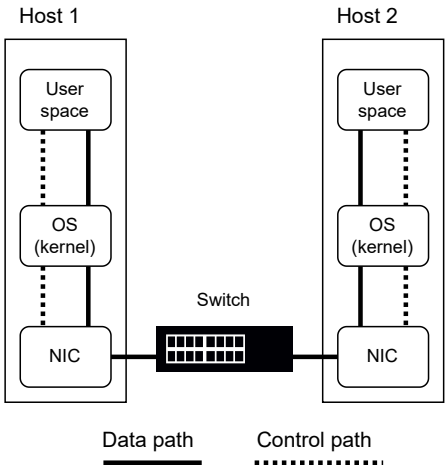


Figure 2.5: Vanilla data transport via software stacks

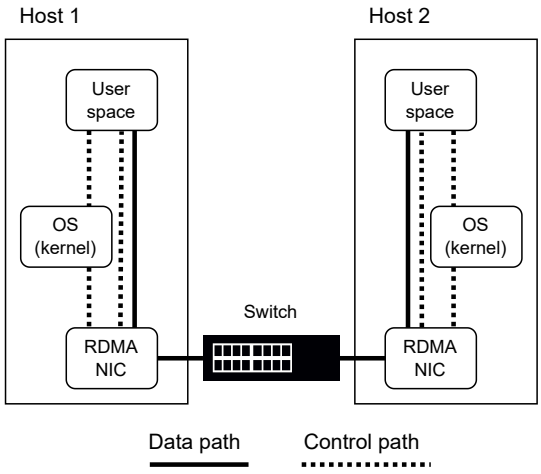


Figure 2.6: RDMA data transport with zero-copy

Infiniband, iWarp and RoCE

This section discusses the differences in the InfiniBand (IB), iWarp and RoCE RDMA protocols. Figure 2.7 gives 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 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 [3].

UDP is a widely used transport layer protocol to transport data in an unreliable connectionless manner over commodity Ethernet. Hence, the protocol does not require prior communication to set up the communication channel and does not require specialised internet hardware. The UDP network stack is shown in the outer left column in Figure 2.7. 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. Despite this, there is still much to handle in the Linux network stack. This burdens the CPU with a lot of computational work and data shuffling but also inefficiencies arise at high data rates because of many interrupts generated by the network card.

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 implementation uses InfiniBand fabric, which reduces header overheads compared to Ethernet fabric. Besides, InfiniBand fabrics and protocols are optimised for low latencies and short distances compared to Ethernet fabrics [6]. Unfortunately, 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.

RoCEv2 uses the well-known and commonly used Ethernet link layer and the IP and UDP protocols. Which differs from RoCEv1 as RoCEv1 does not use IP routing. We do not discuss RoCEv1 in greater detail due to this major disadvantage over RoCEv2. In the rest of this thesis, we refer to RoCEv2 when we mention RoCE. RoCEv2 allows network designers to transport the data traffic through commodity InfiniBand cables and switches. This is a massive advantage over InfiniBand since it does not require a particular cable type and switches. Though, 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 [31]. Besides, RoCEv1 is not routable across InfiniBand subnets since it does not use the Internet Protocol layer.

RoCE uses many mechanisms of Infiniband in the network and application layer, as shown in Figure Figure 2.7. 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. UDP, unfortunately, does not ensure a reliable transfer which might cause issues in various application domains. Luckily this can be taken care of by the higher-level IB transport protocol implementation, which will be discussed in Section 2.4. Regarding the network stack, the IB transport protocol adds headers that contain information about the RDMA operation and connection. Hence the maximum theoretical goodput is reduced compared to native UDP.

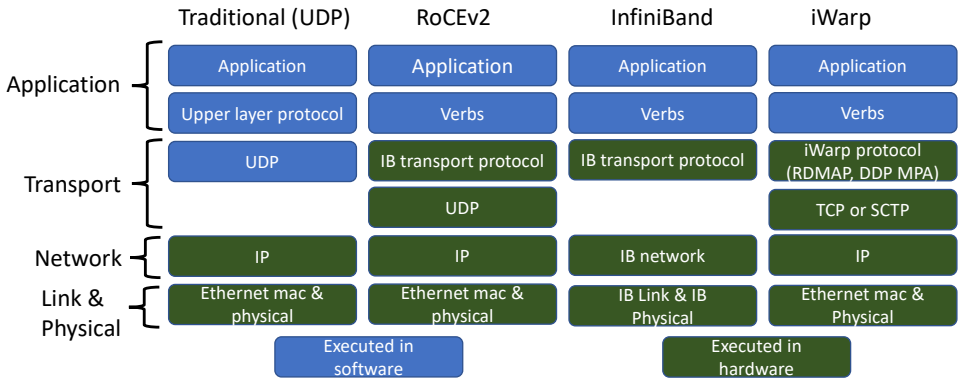


Figure 2.7: Network layer overview of different RDMA protocols

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, so this does not need to be taken care of at a higher level, as is done in RoCE. 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. This protocol is not immediately useful for RDMA since it does not define a wire protocol or operation services such as read and write operations. 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.

For this thesis, we selected RoCEv2 because the implementation should be Ethernet-based and not InfiniBand-based, as the data transport has to be possible over public Ethernet. We select RoCE instead of iWARP because iWARP will be more complex to programme and use with FPGAs because of the TCP connection and less publicly available documentation. Moreover, there are already FPGA implementations available for RoCE, which are discussed in Section 6.1.

2.4. RoCE

2.4.1. Architectural overview

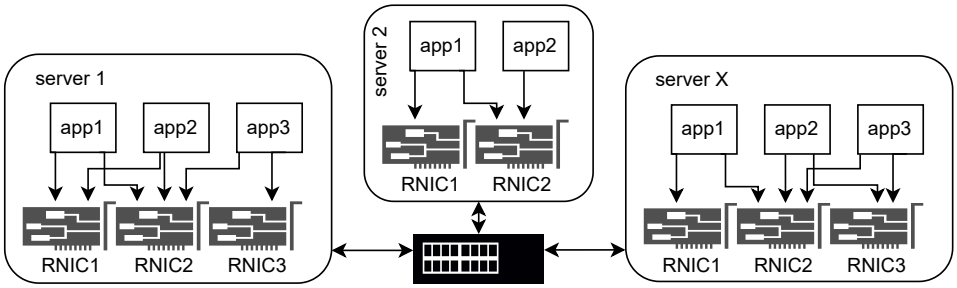


Figure 2.8: RoCE system context overview

This section briefly describes the key components related to RoCE at a high level. To send data via RoCE, one needs at least two servers equipped with RoCE support (e.g. via full software implementation, a RoCE enabled NIC (RNIC) or an FPGA with RoCE support), as depicted in Figure 2.8. The figure illustrates the scalability of the technology as multiple programs could use multiple RNICs. RoCE is built upon several conventional networking protocols to enable Ethernet packet transports. We discuss the usage of these protocols, and the functionalities they provide in Subsection 2.4.2. Subsection 2.4.3 details the notion of converged Ethernet and the networking protocols to create it. In addition, the RoCE technology requires user-space libraries and kernel drivers to communicate between CPUs, NICs and GPUs; these are described in Subsection 2.4.8.

A fundamental aspect of the RoCE protocol is the queue model, which forms the foundation for the communication method of data transport requests between the host program and RNIC (also referred to as channel adapter (CA) by IB), as depicted in Figure 2.9. The queue model defines two queues in which the host program can post Work requests (WRs), a send queue and a receive queue. These queues can only be used to communicate from the host program to the CA. Each work request placed into these queues represents a single data transfer instruction. Some RoCE operations require both servers to post work requests in the correct queue, while others require a single WR at one end node, referred to as double-sided and single-sided operations. The CA has a single queue type to communicate to the host program, called completion queue (CQ). The reusability of queues, the impact of queues on system resources and more details related to the queue model is provided in Subsection 2.4.7.

To establish a RoCE connection between two servers, one must create a queue pair (QP) in each CA. These QPs are connected via a transport service, which defines the degree of reliability and how the QP transfers data. A transport service can either be a connection-oriented or a datagram service. The datagram services allow a single QP to send and receive messages to/from multiple QPs. On the other hand, connection-oriented services can only transmit data to a single remote QP. In addition, the transport service can either be reliable or unreliable. Reliable service guarantees the delivery of each message, or else it will create an error message. These transport services are discussed in greater detail in Subsection 2.4.5.

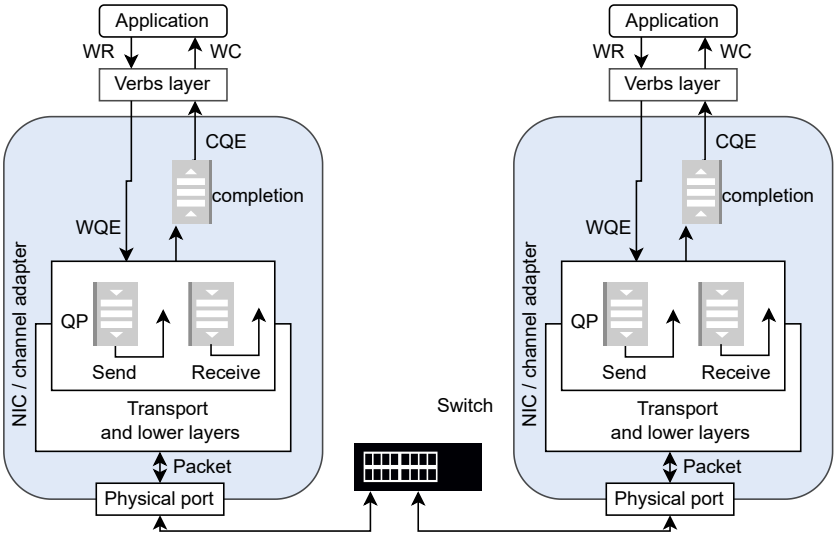


Figure 2.9: RoCE communication stack

A QP contains much more information than just the transport service. During the creation of a QP, one must also, for example, select the allowed operations, QoS and retry counters. Moreover, a QP is associated with a single protection domain (PD) that adds additional abstraction layer to allow multiple QPs to access a group of memory regions (MRs). In addition, each protection domain also has its own key which adds an extra security step. A registered memory region belongs to one or more PD and contains the virtual address of a memory block, the size of the block and the access rights such as read and write permissions. We will not cover some other internal RoCE (or ibVerbs) structures in this report as they do not contain (relevant) settings for our research.

2.4.2. RoCE networking

This section covers the networking protocols and packet formats related to RoCEv2. An overview of the network headers used by RoCE is shown in Figure 2.10. We discuss each header and its protocols, whereafter we explain how one establishes a converged Ethernet network.

Ethernet Protocol

The Ethernet protocol is the lowest protocol layer used in Ethernet traffic, located in the link layer. This protocol defines Ethernet frames and the Ethernet header. A physical Ethernet connection can route such Ethernet frames that include the Ethernet header. The header contains, amongst others, the source and destination address and indicates that the following header is the IP header.

Internet Protocol

The Internet Protocol (IP) is a network layer protocol consisting of two major versions, IPv4 and IPv6, both supported by RoCE. IPv4 is the older, more widely used protocol compared to IPv6. The most significant improvement of IPv6 is the increased address range which will become a problem for IPv4 in the future. The IP protocol enables the fragmentation of a larger message or packet into multiple Ethernet frames. Moreover, the IP header enables network routing and several services such as congestion protocols and priority classification, which will be discussed later on.

User Datagram Protocol

This protocol resides in the transport layer of the OSI model. The protocol is built upon the Internet Protocol and uses a connectionless communication model. The UDP header contains the source and destination ports to distinguish data streams originating from the same sources. Some of these destination ports are reserved for upper-level protocols. The reservation of those destination ports is regulated by IANA [3]. IANA assigned destination port 4791 to the RoCE protocol, thus each UDP packet containing destination port 4791 must contain an Infiniband Base Transport Header (IB BTH).

[3] `https://www.iana.org/`

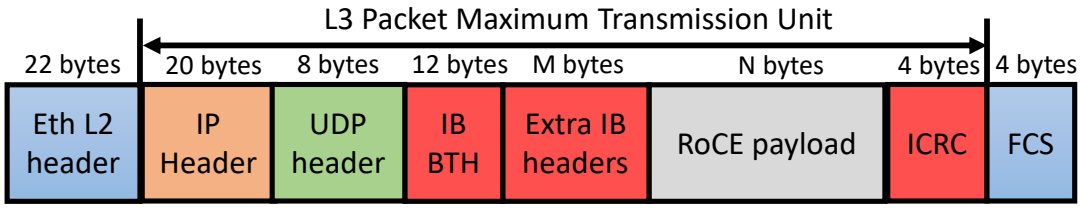


Figure 2.10: RoCEv2 Ethernet frame format

RoCE Protocol headers

The final headers in the Ethernet package are specific to RoCE. These headers are taken from the standard InfiniBand stack. The Base Transport Header (BTH) will always be appended, and depending on the operation and transport service, additional RoCE headers will follow. Then finally, the actual payload can be added and terminated with an invariant cyclic redundancy check (ICRC). This ICRC is specific to RoCE and increases the protection of bit corruption in the payload, compared to the Ethernet protocol's Frame Check Sequence (FCS). The BTH and the other RoCE headers are addressed in more detail in section Subsection 2.4.4.

2.4.3. Converged Ethernet

Converged Ethernet (CE) is a characterisation of an Ethernet network where different application networks are deployed in a single lossless network, often referred to as Data Centre Bridging (DCB). This section discusses protocols that can be used to create Converged Ethernet.

Previously, data centres deployed separate networks for database storage, high-speed interconnects and external connections, which resulted in many cables and hardware. By merging different data streams, the space for hardware and cost is drastically reduced, though it increases the interference between the streams and the probability of congestion in the network. As a result, it becomes essential to differentiate between application-specific data streams since the requirements may differ. For example, one would like to prioritise outgoing data from run-time applications to take precedence over data towards external (backup) databases. A Converged Ethernet network is one in which priority and congestion management mechanisms are applied to mitigate interference and congestion. These mechanisms are implemented via additional protocols inside the networking layers.

Priority mechanism might be achieved via the Priority Flow Control (PFC) protocol, allowing users to map their packets to a priority class. These priorities can be linked to resources such as specific ingress buffers. As a result, transmission is paused independently for each priority when buffers are about to overflow. Allocating buffers to priorities can result in lower-priority data traffic never being sent if there are too many higher-priority packets. The Enhanced Transmission Selection (ETS) standard prevents such transmission stalls. ETS can limit the egress bandwidth per traffic class, allowing lower priority packets at all times to be sent (at a lower speed).

Lastly, one needs to control congestion in networks to reduce losses. Although previous standards reduce the likelihood of congestion, they will not intervene when congestion occurs or completely pause the transmission. A protocol that proactively reduces end-to-end congestion is the Explicit Congestion Notification (ECN) protocol. ECN is an extension of the IP protocol consisting of just 2 bits in the IP header. These bits indicate if ECN is supported and if congestion has occurred in the network data path. The ECN protocol marks packets with congestion when the packet buffer utilisation exceeds the ECN marking threshold. The effectiveness of ECN also depends on the support of ECN marking in the routers and switches. A network administrator is supposed to set these thresholds themselves, which is difficult to optimise in multi-purpose high-throughput and low-latency networks.

The RoCE specification provides *RoCE congestion management (RCM)* to avoid congestion hot spots and optimise throughput. RCM implements a feedback mechanism for the ECN-marked packets received by the destination node back to the source node. This allows the source node to receive congestion notifications and decrease their injection rates on a per QP basis to decrease queueing delays and congestion effects. This feedback is sent to the source node via congestion notification packets (CNP), the composition of these packages is shown in Table 2.1. The interval and amount of CNP are not defined by the RoCE specification; thus the RCM does not need to send a CNP per received ECN marking.

The effects of congestion are getting worse with increasing network speeds in data centre networks. Exist-

Table 2.1: RoCE CNP format

Ethernet header
IPv4/IPv6 header
UDP header
BTH DestQP set to QPN for which the RoCEv2 CNP is generated Opcode set to b'10000001 PSN set to 0 SE set to 0 M set to 0 P_Key set to the same value as in the BTH of the ECN packet marked (16 bytes) - Reserved. MUST be set to 0 by sender. Ignored by receiver
ICRC
FCS

ing RDMA congestion control (CC) schemes have limitations in responsiveness and unawareness of micro-bursts due to long end-to-end control loops. Besides, it requires explicit settings in all network adapters and switches to benefit from the CC mechanism fully. These limitations drive the research to improve congestion control [19, 34, 49]. [32] discusses these limitations in greater detail and proposes a high-precision congestion control (HPCC) mechanism focusing on RDMA congestion, leveraging precise load information from in-network telemetry. Proactive and accurate congestion control (PACC) via P4 switches is proposed in [52]. PACC uses a PI controller technique to calculate the congestion based on previous and current information and signals the congestion via existing DCQCN schemes. RoCC [46] calculates the congestion and fair rate of flows inside switches and signals congestion via custom feedback messages. HPCC, PACC and RoCC require adjustment inside the switch, referred to as switch-driven solutions. Switch adjustments are not necessary for TIMELY [35], based on round trip time measurements between end nodes. The ease of implementation comes at the cost of increased latency and less precise congestion information compared to the other proposed solutions.

Last year, NVIDIA launched their Zero Touch RoCE - round trip time congestion control (ZTR-RTTCC) technology [2], enabling congestion control between supported NICs. Their technology does not require extra configuration on switches and does not require additional parameter selection or hardware support on NICs and switches. This makes it very easy to use, though it reduces the maximum throughput (due to extra protocol packets). In addition, it is difficult to implement this on an FPGA and is likely to be relatively resource-consuming as it requires a lot of state-keeping.

It must be noted that RoCE might be used over non-converged Ethernet networks. However, this increases the likelihood of buffer overflows and congestion, which are no longer detected and resolved by the Ethernet protocols. Consequently, packets will be dropped or delivered out of order causing retransmissions or even message drops, inducing degradation in goodput. In the event of congestion problems between CPUs, we will use PFC, ECN and ETS since these are commonly used protocols.

2.4.4. RoCE headers

Upper-level protocols need a method to communicate their settings and variables to remote nodes. Ethernet protocols use headers to share such settings and parameters between network cards and routers. RoCE adds extra headers to communicate the transport protocol, RDMA operation and other related information. This section describes the headers used for reliable and unreliable connection transport services (BTH, RETH, AETH).

Each ethernet packet created by the RoCE protocol must contain the *Base Transport Header (BTH)*, irrespective of the transport service or opCode. This header has 12 bytes describing, for example, the OpCode, destination QP number and packet sequence number. The packet sequence number (PSN) is used to check if packets arrive in order and to detect duplicated or missing packets. The OpCode defines, besides the (RDMA) operation type, the transmission service and the remaining headers.

A single RoCE operation or message might require multiple Ethernet packets to complete the data transport. The *RDMA Extended Transport Header (RETH)* together with the OpCode indicate the start of a new message. Receiving a RETH packet defines a new RDMA operation for the QP defined in the BTH. If the pre-

vious operation has not yet been completed, it will not be completed; however, the consequences depend on the transport service. The RETH contains the virtual address, the remote key and the memory size in bytes of the requested operation.

The key to reliable connections is the information on whether the destination received the packets correctly. RoCE sends ethernet packets containing the *Acknowledge Extended Transport Headers (AETH)* as a reply in case of reliable transmission. The header contains two fields, the syndrome and the message sequence number, represented by 1 and 3 bytes, respectively. The syndrome indicates if the packet is acknowledged or not and might contain the Limit Sequence Number (LSN) if the packet is acknowledged. RoCE also provides an end-to-end message level flow control mechanism to communicate the available resources from the receiver during run-time, which we will not discuss.

RoCE supports the addition of 4 bytes of Immediate data, which is not written into the memory location defined by the operation. The data is therefore separated from the payload and stored inside the *Immediate Extended Transport Header (ImmDt)*, added in the last Ethernet packet of the message. The receiving end-node copies this data into the consumed work request, which is transformed into a work completion event. Thus the immediate data is delivered to the user program via the work completion instead of immediately written into memory.

RoCE theoretical throughput efficiency

The headers mentioned above reduce the maximum theoretical goodput of vanilla UDP data transport. Besides, RoCE restricts the maximum data payload size per Ethernet packet via the so-called path maximum transmission units (PMTUs). Depending on hardware support, the PMTU can be set to 256B, 512B, 1024B, 2048B, and 4096B. A larger PMTU results in higher efficiency since more data is transmitted per packet. The following example considers a WRITE with Immediate operation with a message of 4096 Bytes and a RoCE PMTU of 4096 B. The standard required headers, namely, the Ethernet, IP, UDP, BTH and ICRC, require 14, 20, 8, 12 and 4 Bytes, respectively. A single Ethernet packet is necessary to fulfil the operation since the payload is equal to the PMTU. So the RETH and Immediate data header will be added in the same Ethernet frame; these contain 16 and 4 bytes, respectively. The theoretical Ethernet efficiency can be calculated according to Equation 2.1 and results in 98.1% efficiency for this example. Hence a 100Gb Ethernet link can achieve a maximum goodput of 98.1 Gigabits per second (Gbps).

$$\begin{aligned} \texttt{efficiency 4kB} &= PMTU/(PMTU + headers) * 100\% \\ &= 4096/(4096+78) * 100\% \\ &= 98.1\% \end{aligned} \tag{2.1}$$

There are (many) Ethernet packets without the Immediate data or RETH header if the message size is larger than the PMTU. This reduces the header overhead and increases the theoretical maximum efficiency to 98.6% as calculated via Equation 2.2.

$$\begin{aligned} \texttt{efficiency middle packets} &= 4096/(4096+58) * 100\% \\ &= 98.6\% \end{aligned} \tag{2.2}$$

Standard UDP traffic requires 42 Bytes of data for the Ethernet, IP and UDP headers and supports larger payloads of up to 9000B called jumbo frames. The decrease of the header size and increase of the payload per header increases the efficiency of UDP data transport to 99.5%. This demonstrates that RoCE, due to the added headers and the limited payload size, has a lower maximum theoretical efficiency (up to 1.5Gbps at 100GbE) than UDP.

2.4.5. RoCE transport services

A transport service defines the conditions and available mechanisms between two or more end nodes. An example of such a condition is the notion of connection-oriented transport, which limits the communication of the local QP to a single remote QP. The transport service also defines whether or not the reliability mechanism must be used. Lastly, the transport service specifies the available (RDMA) operations. We first discuss the available connection-oriented transport services, then briefly discuss the datagram services. It is important to note that all transport services, except Unreliable Datagram, support a maximum message size of 2GB, regardless of the operation. The maximum message size supported by the unreliable Datagram transport service is equal to the PMTU being used.

Connection oriented services

The first implementations of RoCE supported a Reliable Connection (RC) and an Unreliable Connection (UC) service. Later, two other connection-oriented services were added to improve the scalability and reusability of QPs, the so-called Extended Reliable Connection (XRC) and Dynamically Connected (DC).

Unreliable connection transport service is used in systems where message loss is acceptable or in systems where reliability is achieved in lower network layers (for example, via DCB). The service allows just a single channel between 2 QPs. In other words, when two servers run multiple processes, e.g. both three processes, exchanging data between each other, one needs to have a total of 18 QPs to achieve full process connectivity, as can be calculated from Equation 2.3. The service supports SEND and RDMA WRITE operations which might be enriched with Immediate data. All traffic between these QPs flows from the requester to the responder in a unidirectional flow. Hence, the operation requester is never notified of the transmitted packets and message status.

$$\#QPs = \#nodes * localprocesses * remoteprocesses \tag{2.3}$$

The **Reliable Connection** Transport service, on the other hand, allows the responder to inform the requester if it correctly received the packets and the complete message. Reliability is achieved via timeouts and the Go-Back-N retransmission mechanism. This service requires additional settings, such as the maximum number of retransmissions and timeout timers for each QP. The work request shall return with errors and transition the QP to an error state when a timeout or retransmission counter is exceeded. In effect, a reliable connection requires extra state-keeping and extra headers to signal transmission problems. This service can perform all operations supported by RoCE, and the number of QPs needed to achieve (full process) connectivity is equivalent to the UC service.

The **Extended Reliable Connection** service allows a local QP to communicate to all processes on a single remote host over a reliable connection. This reduces the number of local QPs by a factor of remote processes, thus reducing local resource usage. Nevertheless, the XRC is more challenging to implement, and programs only benefit when end nodes require communication to multiple processes in a single remote node. In addition, care should be taken so that the processes do not interfere with each other as the processes use the same resources.

The latest transport service is the **Dynamically Connected** service that combines features from RC and UD. The DC service decreases resource sharing and improves scalability further compared to XRC since a single QP can be connected to multiple processes across multiple nodes. The most significant disadvantage of this service is the single available transport channel for a QP. In other words, a local QP needs to send a complete message to a remote QP before it is allowed to connect and communicate with another remote QP. Besides, the dynamic behaviour drastically increases the complexity of state-keeping.[4]

Datagram services

QPs leveraging datagram services are not tied to a single remote QP. The destination is defined per work request, and an extra context, the end-to-end context EEC, specifies appropriate remote nodes. The operations supported by the Unreliable Datagram (UD) service are limited to SEND operations and thus not capable of RDMA operations or ATOMIC operations. In addition, the message size is limited to the maximum packet size supported in the path. RoCE supports packet sizes between 512B and 4KB (increments by a factor of 2), forcing high message rates when one needs high data throughput.

The Reliable Datagram service solves these issues due to the added reliability at the cost of scalability. The scalability decreases compared to UC due to extra information and resources needed to establish and maintain the reliability mechanisms. Both diagram services add extra headers to the Ethernet packet and thus reduce the theoretical maximum throughput. Besides, the diagram services increase the implementation complexity for FPGAs due to the extra EE context.

2.4.6. RoCE 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. An overview of

[4] https://www.openfabrics.org/images/2018workshop/presentations/303_ARosenbaum_DynamicallyConnectedTransport.pdf

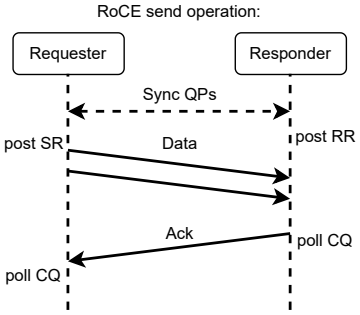


Figure 2.11: Send sequence diagram

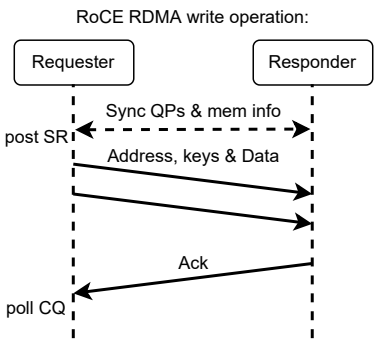


Figure 2.12: Write sequence diagram

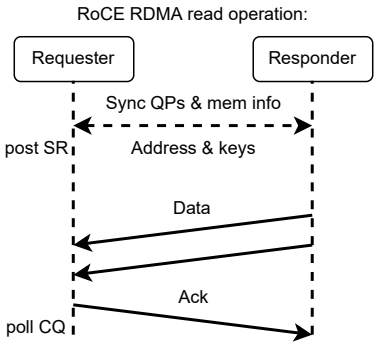


Figure 2.13: Read sequence diagram

Table 2.2: RoCE operations support for specific transport services (+ is supported, - is not supported)

RoCE operation	Transport service				
	Unreliable Connection	Reliable Connection	XRC	Reliable Datagram	Unreliable Datagram
SEND	+	+	+	+	+
RDMA WRITE	+	+	+	+	-
RDMA READ	-	+	+	+	-

the supported operations per transport service is shown in Subsection 2.4.5. We do not discuss ATOMIC and memory binding operations as they are irrelevant to our use cases.

SEND operation

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 2.11 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.

RDMA WRITE operation

Data transport without user-space involvement of the responder is achieved with 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 2.12 depicts the sequence diagram for WRITE operations (without immediate 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. The Immediate data consists of 32bits 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 RNIC. To clarify, the remote responder must post an (empty) receive work request to the receive queue to receive the immediate data.

RDMA READ operation

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 2.13. Requirement R5, given in Section 2.2, stipulates that the system cannot support retransmission which is an essential component of the reliable services. Therefore, we do not discuss this operation in more detail.

2.4.7. Queue model

Work requests must be communicated between the user program and the network card to start data transfers. This communication must be highly optimised to achieve high data rates and a low CPU workload. This section discusses the queue model and related methods available to reduce the (internal) communication load. The communication flow follows the RoCE queue model, which defines four types of queues, a send queue, receive queue, Shared receive queue (SRQ) and a completion queue. A user program can post work requests in the first three queue types. After that, the Verbs layer converts the work requests into work queue elements used by the hardware, as shown in Figure 2.14. We use the ibverbs API as the interface and implementation of the Verbs layer, as discussed in greater detail in Subsection 2.4.8.

A message rate bottleneck might occur when one tries to achieve high throughput while using a small message size due to the work request communication between the user and RNIC. Luckily, multiple mechanisms are available to cope with this issue. The easiest to use is the linked work request. It is a mechanism to link multiple work requests and send them to the Verbs layer and RNIC. Consequently, the number of calls to the Verbs layer and RNIC is significantly reduced. Other mechanisms to reduce the workload, such as solicited and signalled events, are discussed later in this section.

Figure 2.14 depicts the consumer queueing model. The user program creates work requests (send or receive requests) and posts them into their corresponding queue. Each QP has its own set of send & receive queues and might share a completion queue with other QPs. Hence, completion queues are independent of work queues. Nonetheless, each work queue must always be associated with a completion queue since a work request error is communicated via work completion events.

The **send queue** holds operations that cause data to be transferred between two QPs. Hence, the host program must place SEND (w. Imm), RDMA WRITE (w. Imm) and RDMA READ operations into the send queue to request (start) an operation. The information of an operation is placed in a send work request (SWR or SR), which is converted into a qork queue element (WQE) by the Verbs layer and placed in the send queue. The information inside the send request depends on the operation. For example, the SEND operation does not require the knowledge of the destination's memory address, while RDMA WRITE does. Once a work queue element is completed, a completion queue element (CQE) is generated and placed in the completion queue, as visualised in Figure 2.14.

The **receive queue** contains operations that specify where incoming remote data needs to be placed. A receive operation, for example, defines the memory location required when the remote QP uses the send operation. Other instructions requiring a receive queue element are instructions containing immediate data. The immediate data is not instantly written to the memory because it is treated as status information. The immediate data is instead returned as data in a CQE. To generate such CQE, the RNIC must consume a WQE which is only available when a work request has been posted to a queue. Thus, RDMA WRITE operations with immediate data require receive requests too.

An improvement can be made over the standard receive queue when an application posts numerous receive requests to multiple QPs by taking advantage of a **shared receive queue (SRQ)**. The queue is used for receiving operations which allows the same operations as previously explained for the receive queue. The strength of the SRQ is that the queue can be shared (used) concurrently by multiple QPs, even if they use different transport services. Instead of keeping track of and filling, e.g. 10, different receive queues, the program only needs to check and replenish a single receive queue. The SRQ, together with the usage of linked

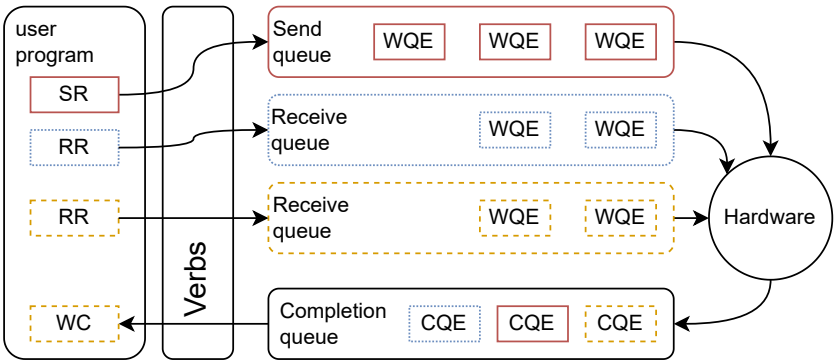


Figure 2.14: Consumer queueing model

work requests, decreases the amount of communication between the host program and RNIC. On top of that, the RNIC now has only a single receive queue to check. As a result, the utilisation of the program might decrease, and the throughput could increase if the host-to-RNIC communication is the bottleneck due to the message rate.

Once a work request is completed, a Completion Queue Element (CQE) is generated and placed in the **completion queue**. A CQE contains information about the local QP, the operation, and the remote QP number. Besides, a CQE contains information about transmission errors. As mentioned at the beginning of this section, a single completion queue might be connected to multiple work queues. Each work request shall always result in a completion queue element when finished processing, except when using signalled events in the send work requests, which is discussed later. One retrieves the data in a completion queue by invoking a *poll()* operation on the queue. If one desires low latency, the program should continuously call the *poll()* operation. Though, this causes a high workload on the CPU. Programmers can reduce the load by polling at a time interval or implementing the notification method.

Ibverbs provide a notification mechanism via a **Completion channel and completion notification**. The RoCE protocol allows one to connect multiple completion queues to a single completion channel. This channel can notify users when a new CQE is available. One must arm the channel with a request notification to get such a notification. The channel can only be armed once, generating just a single notification for the first new CQE. As a result, one must arm the channel each time after reading and acknowledging a notification. After notification, the user must still *poll()* the CQ to retrieve the CQE. Usually, each WQE results in a notification, if not already notified, except when the user uses the solicited flag or signalled flag.

Solicited completion events decrease the number of notifications at the responder and, therefore, might reduce the workload. A solicited event is created when a responder receives a message containing the solicited flag. The solicited flag is thus set in the send work request by the requester. A responder must arm a completion channel with a solicited flag to create a notification when a message is processed containing the solicited flag. Consequently, the responder RNIC might (silently) receive and process multiple unsolicited messages before notifying the user program of completion events in the completion queue. In effect, more CQEs will be polled in a single call, thus decreasing the total number of polling operations. This method is solely possible for operations that require receive work request.

Unsignaled work requests decrease the number of CQE at the requester and thus reduces the communication and workload. This mechanism allows the user to select if a CQE should be generated from a work request. One reason to not signal (send) work completions is the fact that they do not contain new information when processed without errors, unlike the receive work completions. For example, a receive work completion does contain the opcode, message size, and remote QP number. Besides, the user program is still notified if an error occurs. Thus, all previous operations must be successful if the requester processes a signalled completion. The ability to use (un)signalled send requests is an attribute of the QP and each SR.

To emphasise, the solicited flag (set by the requester) impacts the notification events at the responder, while the signalled flag (again set by the requester) impacts the creation of CQE at the requester.

2.4.8. Stack architecture

This section discusses the library, kernel and driver stack related to this research. Figure 2.15 depicts the software stack used in our implementation. User applications can interface with Upper-Level Protocols (ULP) and APIs such as MPI, UDAPL and ibVerbs. In our work, we use the ibVerbs user-space library to interface with lower-level drivers and the network card. The ibVerbs API implements management for InfiniBand (including RoCE) and iWARP RDMA protocols. Many API calls are equal between the supported protocols in ibVerbs; the difference is primarily in the attributes that need to be included. Ibverbs communicates mainly directly with the RNIC, thus bypassing the kernel. Some calls still require kernel interaction, for example, when creating, modifying or destroying RDMA resources such as Protection Domains and Queue-Pairs. The API uses Kernel bypass when interacting with work requests and completion confirmations. ibVerbs uses the *ib_core* kernel module for its RDMA resources and memory-related kernel calls during the creation and removal of RoCE structures. InfiniBand-specific requests are retrieved from the *mlx5_ib* kernel module, while all other queries and lower-level communication are handled via the *mlx5_core* driver.

Furthermore, the *ib_core* implements a peer memory API permitting the registration of device memory located in the same chassis. The Nvidia PeerDirect and the GPU driver, being the CUDA driver, leverage this peer memory API to enable RDMA access to GPU memory, this is discussed in greater detail in Section 2.6.

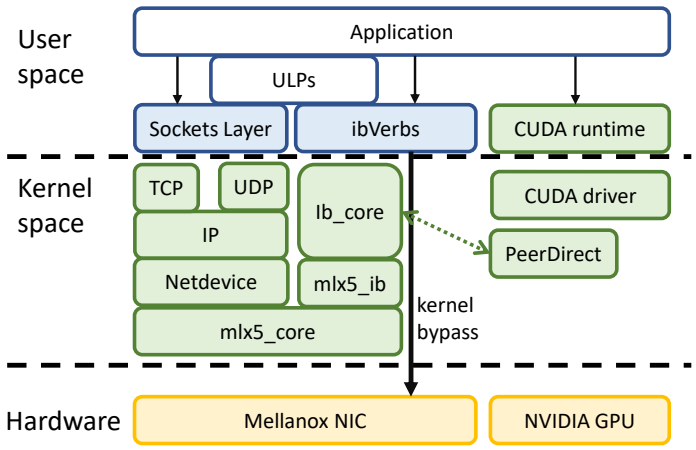


Figure 2.15: RoCE stack architecture

2.5. RoCE applied to the use case

In this section, we discuss the optimal transport service and RDMA operation for the application in radio astronomy based on the use cases and requirements discussed in Section 2.1 and Section 2.2, respectively.

A fundamental choice is the type of transport service. The 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, see requirement R5. 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, 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 due to the EE context, which also increases the implementational complexity on FPGAs. We opted for the UC transport service because the UD has higher complexity and limited message size.

UC supports SEND and WRITE operations which can be extended with Immediate data. Requirement R6 prescribes that it must be clear which data has been received correctly. This requirement, therefore, makes it impossible to use a native RDMA WRITE operation, as such an operation completely bypasses the CPU. 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 memory location per message, which is a disadvantage since storing many memory addresses requires extra resources.

The alternative is to use SEND operations (with immediate data). The receiving side (CPU/RNIC) then 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. Even so, we would still need to use immediate data to get the necessary insights into the (not) received data and its memory location. The underlying problem is that an incorrectly received message does not consume a receive request.

Consequently, one is not sure what the timestamp is of the data placed in the memory. In the further processing pipeline, it is necessary to simultaneously use all data from the different transmitters of a single time point. If it is not sure which time point belongs to the data, this must be determined after receipt and, if necessary, put in another correct place. This requires extra computational effort and is, therefore, less efficient. This problem worsens if one intends to use a shared receive queue to reduce RoCE resources and possible CPU utilisation. If using an SRQ, it is unknown which receive request will be used for the incoming message. An SRQ is used by multiple QPs and thus for multiple data transport connections making it unclear which memory region is used for a specific QP and in which order. Consequently, to which timestamp the data belongs and from which transmitter or subband the data originates is unclear. This can be partially mitigated by using multiple SRQs.

Another disadvantage is that the SEND operation will cost more CPU utilisation and resources. After all, for each receive request, a scatter/gatter element must be created that contains the relevant memory address, length of the operation and local key. This data must be kept close to the process as these are needed to

quickly refill the queues.

We decided to use the WRITE with Immediate operation since it ensures that the data of each time window is consistently placed in a standard manner. This reduces the complexity, and therefore the workload, on the receiving CPU(s).

2.6. Direct memory access to GPU memory

The *ib_core* kernel module allows 3rd party memory to be registered and used for RDMA operations. Employing this method with GPU memory results in 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. Since we only have Nvidia GPUs at our disposal, we will only discuss the method and restrictions for Nvidia GPUs. The interaction between the Nvidia and RDMA interfaces and modules is displayed in Figure 2.15. The necessary permissions and other requirements to bind GPU memory to the *ib_core* module are facilitated by Nivida via the 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 Mellanox driver versions.

We should also consider the available GPU memory for the RDMA operations. The available amount of memory for RDMA operation is determined by the GPU's Base Address Register (BAR) size. The BAR size determines how much memory is immediately and simultaneously available over PCIe. One can therefore use more memory than the BAR size. However, having more memory simultaneously registered for RDMA operations than the BAR allows is impossible. The BAR size varies by GPU type. For example, the BAR size is only 256MB for the Nvidia A4000, which is mainly purposed for display and as much as 60GB for the Nvidia A100, which is mainly purposed for compute. The BAR size of the A100 is larger than its 40GB of device memory so that the entire GPU memory can be used for RDMA operations.

Finally, the placement of the GPU and RNIC in the system must be considered. Servers today often contain multiple CPU sockets, the system is divided into more than one Non-uniform memory access (NUMA) domain. Data movement from one NUMA domain to another is limited by the bandwidth of the interface between the CPU sockets, which is typically lower than the PCIe bandwidth. To achieve maximum performance, it is therefore important to ensure that when transferring data between the CPU, RNIC and GPU, the devices are located in the same NUMA domain.

2.7. FPGA

Field programmable gate arrays (FPGAs) offer reprogrammable hardware which can be adapted to the user's needs after manufacturing. That is to say, the functionality of the FPGA can be changed, which differs from using the same functionality in a different sequence, as is the case for CPU and GPU programming. However, the clock rate of FPGAs is generally lower than that of CPUs and GPUs, limiting the performance boost in some use cases. Nonetheless, FPGAs provide enormous performance boosts in various application domains, such as medical image processing and database acceleration.

Xilinx has introduced an Alveo product line which can be used in computer and server systems because of its PCI Express connectivity. The Alveo line offers various features such as onboard High Bandwidth Memory (HBM2), DDR4 memory, Direct Memory Access (DMA), partial reconfiguration and up to 2x100GbE connections. The ease of deployment in existing server structures and the high bandwidth ethernet interfaces make them highly suitable for our development setup.

2.8. Related work

This section evaluates existing efforts in (scalability) assessments of RoCE between CPU nodes.

In [48], the authors compare TCP, UDP and RoCE over 10Gbps and 40Gbps Ethernet networks. They show extensive improvements in RoCE compared to UDP and TCP, both in latency and throughput. Though, it must be noted that the switch was over-designed as it could support 100Gbps. As a result, congestion was mainly in the network adapters and not in the switch. The authors of [22] evaluate the impact of low-level details such as PCIe transactions and NIC architecture. Besides, they concluded that cache misses in the NIC can be reduced in 2 ways for multiple configurations (different transport services and operations). First, by using huge 2 MB pages, address translation cache misses are reduced. Second, QP state cache misses can be reduced by using fewer QPs. Though, they do not examine the impact of the message size or usage of a shared receive queue.

The effects of throughput-sensitive flows (using large message sizes) and concurrent latency-sensitive flows (using small message sizes) are evaluated in [51], and [30]. Their results show the suppression of small message sizes in the presence of large messages.

[36] presents a complete software implementation for dynamically connected transport. Their work does not require RoCE hardware support, which improves usability. Regardless, they report SSD NFS storage's read and write throughput instead of main memory throughputs.

[16] discusses congestion and livelock challenges when running RoCE with more than 1000+ nodes and an aggregated bandwidth of 3Tb/s in Microsoft data centres. A key finding is using shared buffers in a switch to reduce the likelihood of PFC pause frames. Their work addresses network topology and PFC-related problems, not lower-level RoCE-related settings regarding QPs and the like. Scalability is also examined in [23] by leveraging up to 4 source nodes running multiple processes. Their results show nearly equivalent throughput for 1kB, 10kB, 100kB and 1MB message sizes. Moreover, they also show that RoCE achieves much better ratings in terms of throughput and latency than TCP implementations. Lastly, Collie is designed and implemented in [27], a tool for users to systematically uncover performance anomalies in RDMA systems. The paper provides insights into possible RDMA bottlenecks and performance measurement methods inside the host and network adapter. Their publicly available test program can reveal host or network configuration errors. However, a significant part of the measurement methods, such as PCIe bottlenecks, are unavailable because they are proprietary. Moreover, they do not define what values in goodput or latency should be achieved for particular settings.

All studies mentioned do not provide a detailed insight into the CPU utilisation and the effect of RoCE configuration settings such as a shared receive queue or solicited events. Our work aims to exploit the performance impact of all relevant RoCE settings.

3

Methodology

This chapter discusses the methodology to acquire the information to answer research questions 2 and 3 as stated in Section 1.2. The objective is to derive the best RoCE configuration for application in radio astronomical systems and to validate whether the RoCE protocol can deliver the performance needed for use cases related to radio astronomy. To do so, we need to simulate the requirements and environment of these use cases. Using real telescopes for these measurements would be impractical and impossible. On the other hand, CPU server nodes are readily available and are also easier to use, so we use these in our test setups. Section 3.1 discusses the available servers and other hardware resources, followed by the software tools in Section 3.2. In Section 3.4, we discuss our methods to measure CPU utilisation, goodput and other metrics. We used four setups to gain insights into the performance of RoCE; first of all, we briefly show the difference between UDP and RDMA transport. After that, we divided the necessary tests over three setups to simulate the environments needed and to reduce the design space exploration. With these tests, we approach a radio telescope system as closely as possible with the available test system. In Figure 3.1 we show these last three test setups: *one-to-one*, *many-to-one* and *one-to-many*.

Comparison of UDP and RoCE

The first setup shows the differences between UDP data transmission and RDMA transmission with well-established and publicly available tools such as *qperf* [14].

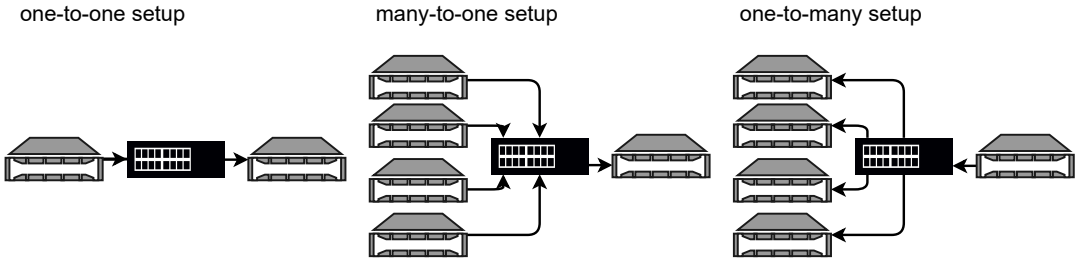


Figure 3.1: Overview of three test setups

one-to-one

The objective of the second setup is to identify the behaviour of various RoCE settings between two nodes. As the basis for our CPU/NIC implementation, we started with Andrew Answer's RDMA transport test application, which provided a single RoCE connection and implements the SEND operation over UC [5]. We adopted this application to test the effect of message size, linked work requests, memory type and the shared receive queue. In the last stage of these tests, the number of Queue Pairs is varied to characterise the scalability of Queue Pairs between two nodes. The insights from these tests are used as a starting point in the many-to-one test. This setup is discussed further in Subsection 3.5.2.

many-to-one

The many-to-one tests are designed to simulate the distributed and scalable aspect of radio telescopes. Thus the goal is to test the scalability of RoCE. We use the insights from the one-to-one test to reduce the design space exploration. This setup uses up to four nodes to send data to a single receiving node as depicted in

Figure 3.1. Furthermore, the number of QPs between each node increases to test the scalability. An in-depth explanation of the many-to-one test is provided in Subsection 3.5.3.

one-to-many

In this setup, the performance is tested in an arrangement where a single RNIC transmits data towards multiple receivers, which characterises the scalability of a sender. This test was conducted for the completeness of our study as this setup is not representative of radio astronomical applications. The relevant settings associated with this setup are presented in Subsection 3.5.4.

3.1. DAS6 cluster

We evaluate the implementation on the DAS 6 cluster at ASTRON [4]. This cluster consists of 4 regular compute nodes and one fat node. Each regular node contains two NVIDIA A4000 GPUs and a single port 100GbE Mellanox Network Interface Card, all connected via PCIe gen4 x16 interfaces. These nodes have two AMD EPYC Processor CPUs in a dual NUMA domain setup containing 135GiB RAM per NUMA domain. In all our tests, we utilise the CPU and host memory located in the same NUMA node as the NIC unless otherwise stated. This eliminates any negative impact due to inter-NUMA domain data transport. The fat node contains one NVIDIA A100 GPU, one Xilinx Alveo U280 and one single port 100GbE Mellanox NIC. This node contains two AMD EPYC 7H12 64-Core Processors in each NUMA domain with a total of 1,08 TiB RAM divided over both domains. All nodes are connected via a shallow buffered 100GbE Mellanox switch. The Mellanox NICS in these nodes have hardware support for the RoCEv2 protocol. Appendix A.1 provides a more detailed overview of the hardware.

3.2. Software

Several software packages and tools are used to compile, validate and measure our program. Compilation of the CPU program is done with *GCC 9.4.0*, and all nodes in DAS6 run Rocky Linux OS. *RDMA-Core 39.1*, *Mellanox OFED 5.4*, and *CUDA 11.6.2* allow us to use the RDMA capabilities between the user program, NIC and GPU. RDMA-Core provides *libibverbs* used in our benchmarking program. Mellanox OFED is a package containing user-space code, kernel modules, drivers and debug tools for the network cards. In addition, NVIDIA PeerDirect (part of Nvidia Linux driver 510) is installed to enable direct memory access between the NIC and GPU. *Slurm* is used to reserve node resources and run tests on the nodes. We used *python 3.9* to iterate over the design space by automating the execution of benchmarks and the creation of figures. The packages used in these scripts can be seen in the repository [29]. We used *TCPdump* (*libpcap 1.8*) and *Wireshark* to verify our application and validate RoCE settings. Appendix A.2 provides a more detailed overview of the software components.

3.3. Network topology and settings

Ethernet network devices use buffers to optimally receive and transmit data. For an Ethernet port, a distinction is therefore made between a buffer for incoming traffic (ingress traffic) and outgoing traffic (egress traffic), as illustrated in Figure 3.2. An important and widely used capability of Ethernet traffic is traffic pri-

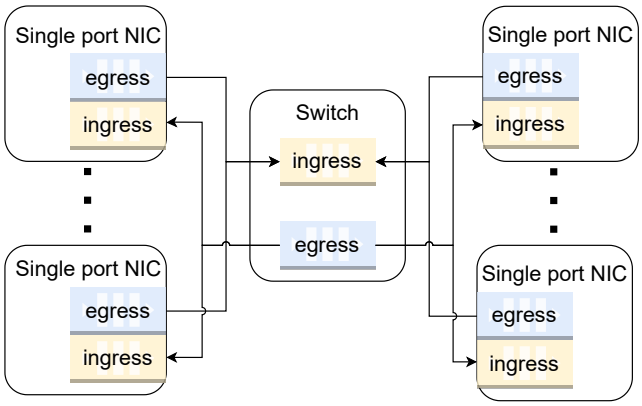


Figure 3.2: Illustration of buffers inside an Ethernet network

oritisation via Traffic Class (TC). Different buffers can be allocated for each traffic class to keep different priorities from interfering with each other. In our test network, we use lossless Ethernet at traffic class 3, for which a separate buffer is used in the network adapters with PFC enabled. In the switch, shared buffer pools are used per TC; for TC 3, the ingress pool has an alpha of 1, and the egress pool has an infinite alpha.[1] The most important property of a shared buffer pool is that the buffer memory is shared over several ports and the amount of memory used is not fixed but dynamically limited by the alpha value.

In addition, certain congestion mechanisms can also be set per priority, such as Explicit Congestion Notification (ECN). For TC 3, we enabled ECN on the ports in the switch with a minimum and maximum threshold value of 300kB and 3MB, respectively.

3.4. Measurement method

The RoCE data transport method we want to test must operate at very high data rates while achieving very low CPU utilisation. These performance characteristics make it relatively easy to affect the measurement and so extra care must be taken about the method used to measure these and other components. After all, our measurements should not influence the actual measurements. Besides, RoCE provides many features and settings which create ample design space, so besides choosing measurements properly, it is required to store and process the measurements in a standardised and automated way. In this section, we first discuss the method to measure CPU utilisation, after which we discuss the goodput measurement method in Subsection 3.4.2. At last, we discuss the application profiling in Subsection 3.4.3, which describes a method to measure function calls.

3.4.1. CPU utilisation

CPU utilisation of the receiving server is an important performance index in this study. Different techniques allow us to measure system, process (program), or thread-level utilisation. Our benchmarking is performed on nodes with large amounts of resources. The benchmarking program can use a single thread per QP or multiple QPs in a single thread. Therefore, the system's overall (resource) utilisation would not represent the program's actual load, we require a method to measure utilisation per thread. Methods to measure the utilisation on a per thread bases are, for example, *Linux perf* [25], *the proc filesystem* [26] and *clock_gettime()* [24]. The measurement needs to be done in a consistent manner while iterating over multiple design spaces. Besides, the measurement must be stored in a standardised, automated manner to enable automated plotting of data.

The *proc filesystems* provide relevant insights; however, the documentation does not provide information regarding the measurement interval or precision. The *perf tool* is embedded in the Linux kernel and provides various methods to measure many performance characteristics. Although, it does not provide an easy-to-use API to measure at a self-defined interval. Moreover, we would like to have the opportunity to measure at an interval of tens of milliseconds to identify bursting behaviour. Measuring the utilisation with *clock_gettime()* is possible on a per-thread basis and at a self-selected interval and locations inside the program. However, this method consumes clock ticks (and thus increases utilisation) inside the thread.

All things considered, we decided to use the *clock_gettime()* measurement method as it allows a fine-grained measurement method when needed. In addition, we do not expect measurements at millisecond intervals to involve significant utilisation.

3.4.2. Goodput

Goodput is defined as the rate of useful bytes that a node receives. Consequently, the actual ethernet throughput is always larger than the goodput since ethernet protocols add headers to the payload. The goodput can be derived via several methods, such as network adapter counters or the number of completed messages.

The goodput can easily be calculated in the application itself if the amount of correctly received messages and the corresponding interval is known. The message size set in the work request defines the goodput of a single RoCE message. Then one can calculate the goodput when the number of completed messages in a time interval is known. Though, the smallest achievable measurement interval and precision depends on the message size and the polling interval of the completion queue.

Another method to determine the goodput is to leverage the counters of the network card or switch. These report the total number of Ethernet packets and bytes received from all connections. These counters are

[1] `https://support.mellanox.com/s/article/understanding-the-alpha-parameter-in-the-buffer-configuration-of-mellanox-spectrum-switches`

not specific to RoCE, and the update interval is also unknown, which constitutes a significant disadvantage. Nonetheless, the network card reports other valuable metrics such as congestion packets, out-of-buffer incidents and RoCE retransmission in case of reliable service. The network counters can be accessed locally on each node, whereas the switch counter requires a connection to the switch's management interface. Besides, the update interval and time resolution of the counters are unclear. This imposes yet another disadvantage to leveraging the switch for throughput measurements.

We decided to calculate the goodput using work completions as it can be measured through a self-set frequency. However, with this method, it must be noted that the maximum measurement frequency is limited by the interval at which work completions come in. The goodput is measured on each node since messages might be dropped between the sending and receiving nodes due to unreliable transport service. In addition, we report the network card counters on a per benchmark basis to detect transmission problems such as buffer overflows and congestion.

3.4.3. Application profiling

Knowing the goodput and CPU utilisation is vital to answering our research questions. Nevertheless, they do not provide insight into time-consuming function calls. For example, without application profiling, there is no insight into the time it takes to create a QP. Furthermore, RoCE settings can also impact the speed of creating or deleting RoCE-related structures. For this reason, we have created a profiling tool using the *Chrono* high-resolution clock. To make the data easy to view, we have stored it in a *JSON* structure to use it with *Chrome tracing* [37]. *Chrome tracing* is a free-to-use web application built into the chrome web browser to visualise event tracing data. A *JSON* file is also easy to use and analyse in python. Unmistakably, trace profiling comes at the cost of keeping track of multiple timers. Moreover, the profiling tool is designed to write the profiling data during runtime in the *JSON* file. This method ensures the availability of the data even when the program crashes at the expense of increased runtime overhead. For this reason, we do not use application profiling during performance tests unless otherwise stated.

3.4.4. Other measurements

The methods discussed in the previous subsections provide information for our KPIs. However, these do not provide sufficient insight into a possible cause of any degradation in performance. For example, the goodput shall drop if the work queue is not replenished on time, so it is important to know if it becomes empty during measurement. We have therefore added a method to track and report queue utilisation during the measurement by keeping track of the number of posted and completed work requests. Another cause by which goodput can drop is the loss of messages during transportation. This might be caused by packet drops due to buffer overflows, the out-of-order arrival of messages, ICRC errors or congestion in the network. These causes can be traced through the counters of the NIC. In addition, transport and congestion insights can be obtained via the switch's management interface. The disadvantage of the switch counters is that they are updated at most once a second. This results in less insight into the (peak) values because they are often also averaged over time.

3.5. Measurements

In Section 2.4, we discussed the RoCE protocol and its features. The design space will become enormous if we use a naive design exploration method. Therefore, we have broken it down into four stages, which we will explain in their respective subsections.

3.5.1. RDMA VS UDP

The aim of this setup is to demonstrate the performance difference between UDP and RDMA transmission. We use Qperf [14] to answer this question as it supports both RDMA and UDP transfers. In addition, Qperf can report CPU utilisation and the achieved goodput. We compare the results of *rc_rdma_write_bw* and *udp_bw*. We perform these tests with node501 and node505 in DAS6 as sender and receiver, respectively. We will use a few message sizes (4kB, 32kB, 1MB) as they might impact the performance of both transport methods.

The following command-line argument is used to test the performance of RoCE:

```
qperf -v -cm1 --use_bits_per_sec -t 30 --msg_size <size>

<remote_IPaddress> rc_rdma_write_bw
```

The following command-line argument is used to test the performance of UDP:

```
qperf -v --use_bits_per_sec -t 30 --msg_size <size> <remote_IPaddress> udp_bw
```

3.5.2. One-to-one

The one-to-one connection setup and experiment aim to identify the impact and trade-offs of RoCE settings between two CPU nodes. This subsection covers this single connection experiment's goal and design space. From the use cases discussed earlier, desire emerges for a system with the smallest possible message size and a large number of connections, while at the same time, CPU utilisation must remain low and data throughput high. We first identify the parameters that impact the throughput and utilisation for small message sizes. After that, we identify the effect on the performance when the number of Queue Pairs increases between two nodes. This data allows us to determine the optimal configuration and possible trade-offs for the use cases of radio astronomy. Besides, this approach gives us a good starting point and smaller design exploration when the setup expands to multiple nodes.

The two nodes are connected via a switch over 100GbE. The network is configured into a lossless configuration via PFC. We run the traffic on traffic class 3 (TC3) with PFC enabled. Thus the RoCE traffic in the network cards is sent and received using different virtual buffers than standard UDP and TCP traffic. The data rate is set to unlimited and the vendor transmission scheduling algorithm is used. The switch is also modified to run TC3 in separate buffers and queues. The congestion notifications thresholds inside the TC3 buffers are set to 4MB and 6MB, respectively the minimum and maximum. Hence we are using converged Ethernet as explained in Subsection 2.4.2. Besides, a single NUMA node domain is used during each test to eliminate any overheads and limitations that might occur due to intra NUMA domain communication. Thus the network card, GPU, CPU memory and CPU cores are located within the same NUMA domain.

Furthermore, the send, receive, and completion queue sizes are unchanged between these experiments. The network card takes work orders from these queues, and the size of these queues should not have any effect as long as they are not emptied during the measurement. The queue size might become important when the number of QPs increases since more resources will be needed. Hence, we consider the queue size when we test scalability with multiple nodes, as discussed in the next section. The queue utilisation is measured during each run to determine if the program can post work requests at sufficient speed.

Each configuration will be tested over a period of 10 seconds and will be repeated three times to expose any anomalies between runs. The measurements at the start of the test and at the end can result in large outliers. Caused by, for example, the first-time filling of queues and not starting or closing threads simultaneously on all nodes involved in the measurement. Besides, we are interested in sustained performance because the data transport for our use cases takes place over a long period of time without fluctuating data ingress rates. Therefore we exclude the first and last second of measurement data for our analysis.

Transport service and RDMA operations We evaluate the RoCE protocol using RDMA WRITE with Immediate operations over the unreliable connection transport service. A rationale for this choice can be read in Section 2.5. However the application allows to configure alternative RDMA transport services and operations.

Message size The message size impacts the system resources (larger buffers) and the possibility of losing a message. We want to minimise the message size since a larger message requires the transmission of more packets between nodes. This increases the possibility of losing a packet during transmission. The loss of a single packet will lead to the loss of the entire message since we are using an unreliable transport service. On the other hand, we expect performance degradation at smaller message sizes due to communication limitations between the CPU and NIC and higher processing load due to increased RoCE operations per second. We iterate over different message sizes (from 4kiB to 200MiB) to identify the impact of the message sizes.

Memory regions, page size and GPU memory We used three types of memory allocation to reduce the design space: transparent huge pages, 2MB and 2GB huge pages. These three types should give us insights if one creates a bottleneck in throughput. These settings are only used on the receiving side since we are mainly interested in the receiver's behaviour. The number of memory regions and their sizes used in a single QP can also be varied. These settings depend heavily on the production application that leverages RoCE. We expect the number of memory regions used, in our case, to have no effect on throughput or utilisation but mainly on resource utilisation. We expect this because the test application reuses memory regions in work jobs when the queue is not yet fully filled. Hence, the number of memory regions is set to a constant value while the size of a single memory region is equal to the message size multiplied by the number of linked work requests.

Linking WR and solicited events Work requests can be linked and posted in a single call to the RNIC. This configuration is expected to decrease utilisation and might improve throughputs in certain cases. We, therefore, increase the number of linked requests (in several steps from 1 to 100) to see when this performance improvement stagnates. In addition, we are testing the impact of solicited events together with linked work requests. One solicited event per linked work request will be added during these tests.

Queue pairs and Shared Receive Queue The number of queue pairs is increased after the before-mentioned settings have been tested. The number of QPs is increased in four steps to a maximum of 500 connections to keep the design space small while revealing the possible degradation of performance when the amount of QPs increases. In addition, it is also more relevant to observe the impact when the data originates from multiple sources, which will be tested in the many-to-one setup. By default, our test application provides each QP with its own thread; however, this might not scale well for large amounts of QPs. Therefore, this iteration will also be extended to use shared receive queues at the receiving node. In this case, we combine all QPs in a single SRQ during this test.

Unsignalled events Both the source node and the sink node may experience bottlenecks. Whereas the previous solicited events and SRQ should reduce the CPU and memory utilisation in the sink node, they do not influence the source nodes. Besides linked work requests, one could try to use solely signalled work events to reduce the CPU utilisation in source nodes. This mechanism reduces the insight into the finished work requests, which may make this configuration unworkable for some applications. Nevertheless, in our use cases, the transmitter will be replaced by an FPGA making this option impossible. Nonetheless, we will only leverage this method if there is evidence of a source node bottleneck that reduces the throughput.

3.5.3. Scalability; many to one

In a real-world scenario, multiple FPGAs send data toward a single compute node. The compute node thus needs to handle multiple connections originating from different devices. This test aims to asses such scalability and to reveal the trade-offs and limitations.

This setup has several similarities with the one-to-one test. Each test will again use the RDMA WRITE with Immediate operation over the unreliable connection service. Secondly, the program runs in the same (single) NUMA domain to avoid side effects due to NUMA crossing. Lastly, we use the same measurement method, so each test is run for a period of ten seconds and repeated three times.

As stated before, the DAS6 cluster at ASTRON is used for each test. This cluster is limited to five compute nodes with a 100GbE RNIC; as a result, the scalability will be tested with a maximum of four sending nodes and one receiving node. We use the most powerful node (node505) as the receiving node in each configuration. The four sending nodes have a higher aggregated throughput than the receiving node is capable of receiving. Without adjustments, this would definitely lead to congestion problems. Several methods to deal with congestion are discussed in Subsection 2.4.3. During testing, the bandwidth of each transmitter will be limited to 23Gbps link speed via Enhanced Transmission Selection (ETS) to avoid major congestion problems[2]. This is representative of an actual radio telescope system since the transmitting FPGAs will also be developed and configured in such a manner to avoid a regular congestion problem in the network. Besides, the aggregated bandwidth with this setting is 92Gbps hence requirement R4 is still met.

Message size The message size impacts the system resources (larger buffers) and the possibility of losing a message. The size of these messages will be based on results from the one-to-one tests, with a focus on smaller sizes, as these are preferred.

Page size and GPU memory To reduce the design space, we use two types of memory allocation: GPU memory and transparent huge pages or 2GB huge pages, depending on the test result of the previous tests. Again we will only use these settings on the receiving side as we are mainly interested in the receiver's behaviour.

Memory regions The amount of memory used by our test application will increase considerably when the number of clients and/or QPs increases, especially in the receiving node. The receiving node in our test application needs an amount of memory available equal to the amount of memory used by all clients combined. The number of memory regions and their sizes depend heavily on the production application that leverages RoCE. Therefore, it is not possible to choose a representative value for this.

We expect the number of memory regions used, in our case, not to affect throughput or utilisation but mainly resource utilisation. We expect this because the test application reuses memory regions in work jobs when the queue is not yet fully filled. Hence, the number of memory regions is set to a constant value while

[2] A 10% deviation from the requested rate limit is considered acceptable by Mellanox.

the size of a single memory region is equal to the message size multiplied by the number of linked work requests. For this reason, we chose the node with the largest memory (node505) as the receiving node. The number of memory regions will be unchanged compared to the previous setup, provided the memory usage does not become too large in case the previous setup shows that a relatively large message size must be used to achieve our objectives. In addition, the memory region's size is (again) linked to the number of linked work requests and the message size.

Linking WR and solicited events For these settings, a single value will be chosen based on the best results of previous one-to-one measurements. The solicited events method will be used if it shows positive effects. The number of linked WR will also be determined from previous results to limit the total number of measurements.

Clients, Queue Pairs and Shared Receive Queue The number of queue pairs is a crucial parameter during the scalability test. The number of queue pairs increases in the receiving node as the number of clients increases. The maximum number of QPs to be tested is based on one of the use cases and is covered later in this section.

The benefit of the SRQ will be determined from the previous setup; if positive, we will use the SRQ for all tests to check whether a larger amount of QPs causes a problem.

Queue size Though we expect the solicited events mechanism to decrease the CPU utilisation, it also imposes a small drawback on the resource requirements. Solicited events decrease the number of notifications, causing the completion queue to be read out at a slower interval, so more events must be stored in the queue. Consequently, QPs leveraging solicited events require a larger completion queue. The SRQ also impacts the optimal queue size since all WR and WQ will be placed in a single queue rather than a queue per QP. Therefore, the receive and completion queue will be at least be a factor 2 larger than the send queue in the client nodes. If a suboptimal throughput is measured, a larger queue size might be tested to determine if it was a limitation.

Unsignalled events In case the unsignalled feature was required in the previous setup, it will be used again during these tests.

Scalability calculations

The data generated by a single FPGA needs to be transported to a single or multiple end nodes, as described in Section 2.1. Moreover, the subband data of a single antennae need to be gathered at a single compute node. This requires a transposition of the antennae data, which might be achieved by the transport method. In this section, we will derive the number of QPs required for the LOFAR use case, Figure 2.1. Relevant information on LOFAR is given in Table 3.1.

We make several assumptions to derive the number of QPs required by a single central compute node. First, we assume a connection topology that allows the data to transpose over the network. Second, we consider the usage of a single QP per subband at each sending FPGA. This implies that a single message contains the data belonging to a single subband originating from one FPGA. This is the most fine-grained manner to distribute the subbands across QPs and thus requires the most number of QPs to transport every subband. Another method is to transmit the data of multiple subbands in a single message. This results in interleaved subband data placement pattern, which the central processing application should support. Suppose the processing application cannot perform optimally with this interleaved placement. In that case, it will cost extra CPU utilisation to rearrange the data, which is precisely what we are trying to diminish. Besides, losing a single packet would result in a data loss of multiple subbands.

Using the assumptions mentioned earlier, the maximum number of QPs needed per central processing compute node can be calculated with Equation 3.1, with the amount of QPs rounded up as we do not transmit

Table 3.1: Overview of relevant LOFAR system properties

Sending FPGAs per field station	4
Maximum subbands per sending FPGA	122
Total number of subbands per field station	488
Number of field stations	80
Central compute nodes[3]	24

[3] 12 servers with 2 sockets are regarded as 24 nodes

partial subbands.

$$\begin{aligned} max\_\#QPs\_per\_computenode &= \frac{\#subbands * \#senders}{\#compute\_nodes} \\ &= \frac{488 * 80}{24} \approx 1627 \end{aligned} \tag{3.1}$$

In our study, we take extra margin by increasing the maximum number of QPs to be tested to 2000 QPs, allowing any expansion to more stations or subbands to be accommodated. This number of QPs will be tested with a single receiving compute node in DAS6 over 100GbE. As such, we expect a throughput per QP of 45Mbps, considering a RoCE Ethernet efficiency of 98.1% and an aggregated bandwidth limit of 92Gbps.

3.5.4. Scalability; one to many

With this setup, we examine whether limitations arise in the current RoCE implementation and hardware when a sender is tasked with more QPs. In addition, we demonstrate that our application is capable of handling such topology. Nonetheless, the RNIC sending node shall be replaced with FPGAs in a real-world scenario.

The sending and receiving nodes are swapped compared to the previous many-to-one setup. Other settings, such as network settings, NUMA domains, measurement method and the number of nodes, remain unchanged.

The tested configurations are a subset of the many-to-one configurations with a focus on the sending node, as this node is more stressed due to the higher number of QPs. The subset will be determined based on the results from previous tests.

4

Implementation

This chapter discusses the system context required to run our RoCEs test application and RDMA API implementation. The application enables us to quantify the performance of RoCEs when exploring its design space and is publicly available via Gitlab [29]. We discuss the system in a hierarchical top-down manner via the C4 model[1], the legend for the C4 model figures is given in Figure 4.1. First, the system context overview is given in Section 4.1. Second, we elaborate on the RoCEs application and RDMA API interaction in Section 4.2. There after, Section 4.3 provides the most fine-grained explanation of the components that compose the application and API.

4.1. Overview (or system context view)

The input to the RoCE test system is given by a user or by a python benchmark automation program as depicted in Figure 4.2. These external users provide command-line arguments as input to select various settings, such as the number of memory regions and the test duration. The repository provides a complete list of the command-line arguments [29]. The user must launch the RoCE benchmarking program on (at least) two machines equipped with network interface controllers supporting RoCE, which form the RoCE test system together. These RNICs exchange RDMA information (needed to establish a RoCE communication channel) over TCP and transmit or receive RoCE packets. The software and hardware components required for RoCE are shown in Figure 4.2 as an external system since we have to rely on TCP sockets, ibVerbs and the network card for some features.

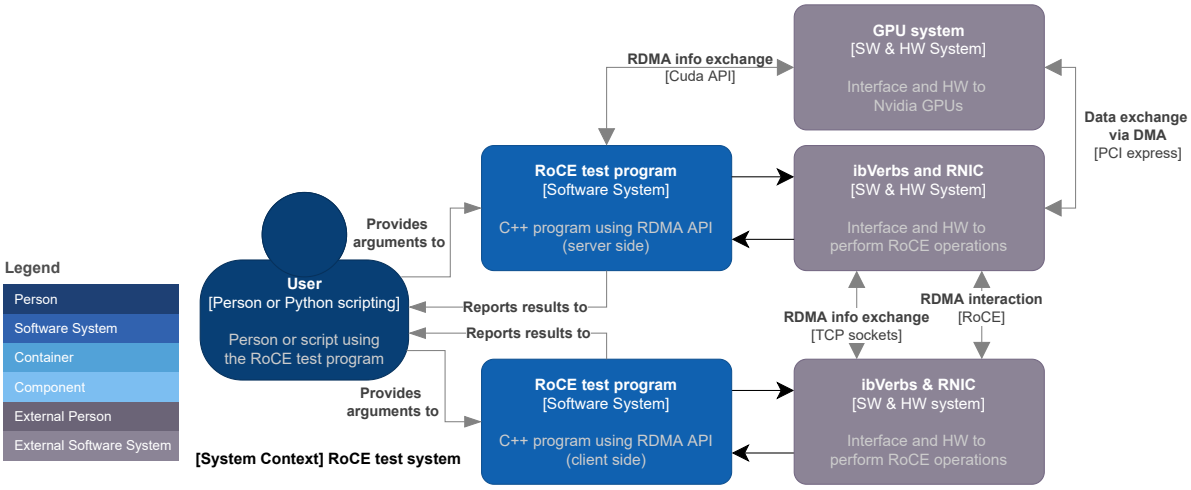


Figure 4.1: C4 model legenda

Figure 4.2: RoCE test system context overview

[1] https://www.c4model.com/

The system needs a single RoCE benchmarking program running in server mode to which multiple clients might connect. The maximum number of concurrent clients and the maximum total number of clients are set at compile time, while the user provides the actual number of expected clients as a runtime argument. The memory usage required to receive or send data to each client restricts the total number of clients.

The data transport direction between the server and client nodes depends not just on the type of operation but also on the *reverse* program option, *-R*. By default, the server node acts as an RDMA requester and thus initiates the operations, while clients act as RDMA responders. Consequently, the *reverse* program option allows us to transform a one-to-many topology into a many-to-one topology, in the case of RDMA WRITE with Immediate operations. The test program does not support many-to-many topologies natively. Nonetheless, this could be achieved when running multiple program instances on each node.

Furthermore, the user can select a NIC and the preferred network port if the host machine contains multiple RNICs that expose multiple ports. Likewise, it is possible to select a GPU if more than one is present.

4.2. Container view

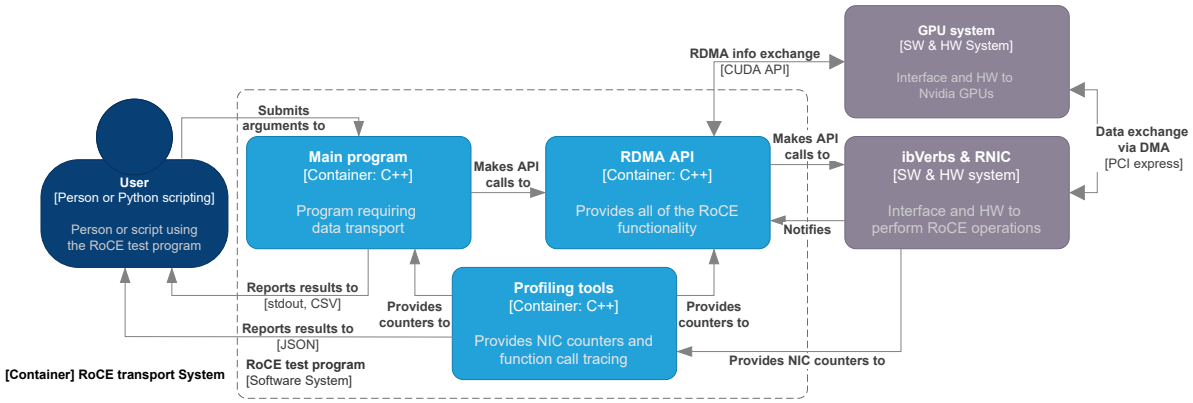


Figure 4.3: Container view of a single RoCE test program

Figure 4.3 provides a more detailed view of a single RoCE test program. The program is divided into three containers, the main program, RDMA API and profiling tools. The user (or python program) interacts with the main program. The main program can be replaced by any program that seeks to transport data via RoCE and uses the RDMA API. To put it differently, this program can be replaced and adapted to other use cases, and above all, it can be developed independently of the RDMA API. The RDMA API container provides an abstraction to the ibVerbs and GPU memory usage. The third container is profiling tools; this container provides methods to retrieve network card counters and measure time spent on functions.

4.3. Components view

4.3.1. RDMA API

The inner components of the RDMA API container are shown in Figure 4.4. The test program interacts with the RDMA API component and the RDMA memory manager. The *RDMA API component* provides several services. Firstly, the RDMA API interacts with the ibVerbs layer. The API uses ibVerbs to query NIC properties and to create and modify QPs and all RoCE attributes. Secondly, the API initialises the memory manager, which keeps track of each region's state. Thirdly, the API provides "default" transport functions to the main program, which handles the actual data transport. These transport functions post work requests and check the work completions. We have developed several functions optimised for our objectives, located in the RDMA work request component, explained later on. It is important to note that these functions depend on the use case, so they should be adjusted as needed. Lastly, the RDMA API provides methods to exchange RDMA information between the external nodes.

The *RDMA memory manager component* has a central role in the RDMA API container, providing information between the primary and child threads. The RDMA memory manager allocates memory and keeps track of two attributes: whether the region is queued and the validity of the data. In other words, the memory manager provides functions to determine whether the data transport has taken place and thus contains old

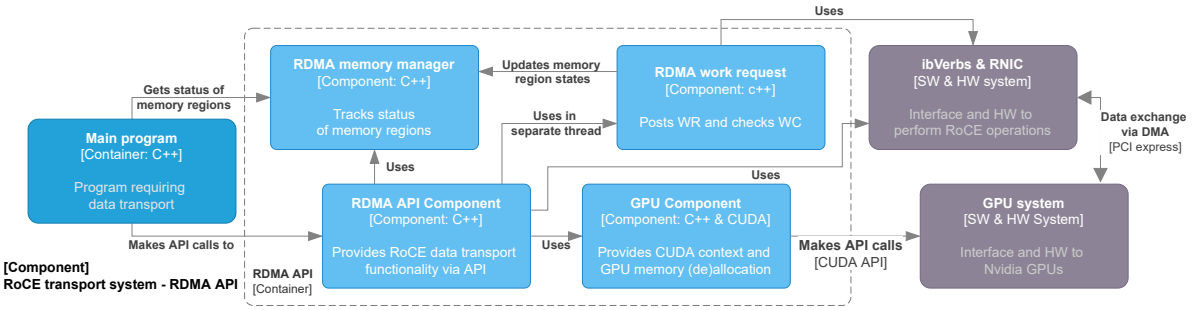


Figure 4.4: Component view of the RDMA API container

(incorrect) or new data. These states are initialised by the RDMA API, updated by the RDMA work request component and read by the main program. These components (might) run in separate threads; thus, the memory manager uses a mutex lock to update and read these states safely. The placement and handling of RDMA operations (messages) are implemented in the *RDMA work request component*. The API is designed in a modular but flexible manner, and as such, multiple transfer methods have been developed to cover all relevant capabilities of RoCE. Each transfer method is started in a separate thread so that the main programme is not blocked while the data transport takes place.

The RDMA API can use GPU memory for RDMA operations. This feature is implemented in a separate *RDMA GPU component* to improve modularity and usability. This component employs a direct memory access method between the RNIC and GPU, provided by NVIDIA PeerDirect technology, as described in Section 2.6. This technology improves the latency and PCIe utilisation since the data is not copied to the main memory.

4.3.2. Profiling tools

The profiling tools container implements two profiling tools that enable the program to read NIC counters and measure the wall time of a function call. These tools have been designed to be independent of the other containers and components, making them usable in other applications too. The structural decomposition of the profiling tool container is shown in Figure 4.5. The *NIC snapshot component* supports Mellanox adapters utilising mlx5 drivers. However, other cards should be easily supported if they report the counters in a similar manner (via the filesystem). The second tool is the function profiling tool. This tool aims to gain insight into the execution time of function calls. It uses the *Chrono* high-resolution clock and writes the data directly into a *JSON* file as soon as the function is finished. In addition, one can turn this tool on or off before compilation by editing a single line of code. This eliminates any overhead in the application after compilation when turned off. This tool was mainly used during development to gain insight into slow components and the number of times a function is invoked.

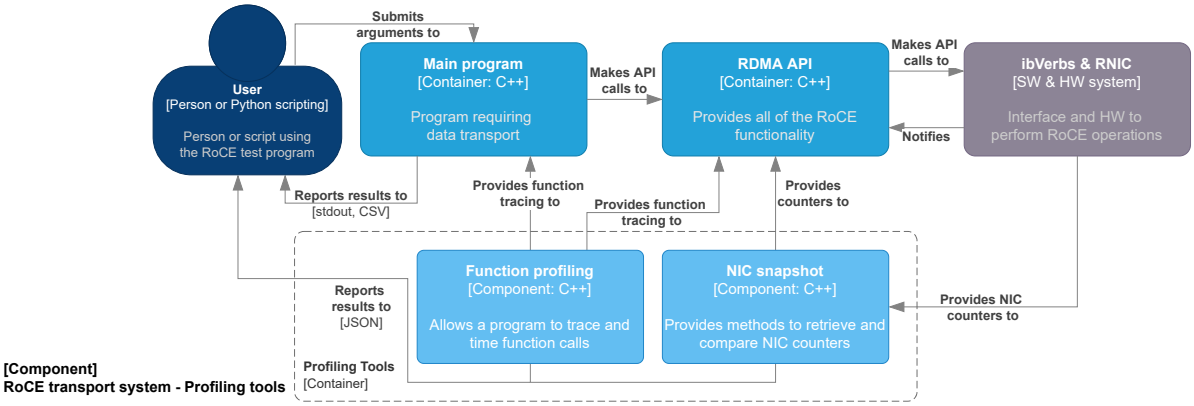


Figure 4.5: Component view of the profiling tools container

4.3.3. Main program

The main program can be regarded as an intermediate layer between the input (the terminal commands) and the actual use of RoCE by the RDMA API. It is specifically made to accept and pass many different settings to the API and to take relevant measurements for our study. So this container can be replaced by another application or processing pipeline that requires data transport.

For this container, the internal components are less relevant than the tasks required to establish and use a RoCE connection. Therefore, we discuss the sequence diagram provided in Figure 4.6 instead of a component view of the container. The sequence diagram shows the basic operations required for the usage of the RoCE protocol between two servers using RoCE-enabled NICs. First, each host must create a QP and its required attributes, such as the protection domain, completion queue and event channel.

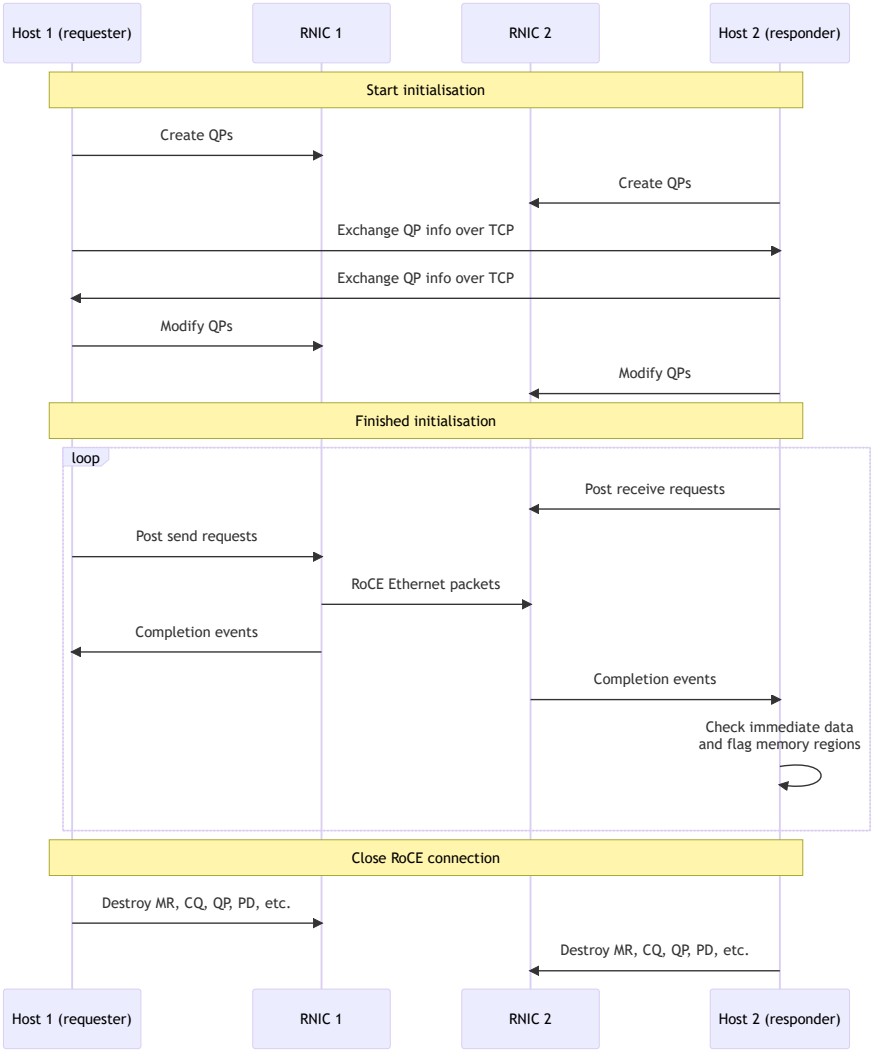


Figure 4.6: Sequence diagram of the primary interaction between hosts and RNICs in the test system

After this, the information from a QP can be shared with the remote host over a connection of your liking. We use a TCP connection because it provides guaranteed transmission of the data. One must exchange the QP number, packet sequence number, global identifier (this is a sort of IP address for RoCE), and local key for each QP. In addition, for each memory region, the memory address, format and matching key must be shared. Some of this information is then used to modify the QP's state and connect the QP to a single remote QP. Linking the remote QP information allows the local QP to send (or receive) data to (or from) this remote QP. The remaining information is required once a RoCE work request is posted to the respective queue.

The QPs are now ready for use, and the memory regions can be used until they (or other important RoCE attributes) are discarded. Because we use the WRITE with Immediate operation, the receiver of the data,

i.e. the responder, must first place receive requests in its receive queue. Then the sender, i.e. the requester, can place send requests in the send queue, which will cause the local RNIC to read the corresponding data from memory and send it over the Ethernet via RoCE packets. Because of the unreliable connection, the sending host receives a completion event of the operation as soon as the last RoCE packet for the operation has been dispatched. Thus, the completion events in the requester do not depend on a correct transmission or acknowledgement from the responder RNIC or host.

The responder host receives a completion event as soon as all packets of a single operation (also called a message) are successfully received. The completion event contains, amongst others, the immediate data and error codes if relevant. With this information, the related memory region can be flagged so that the further processing application or pipeline is aware of the existence of the received data.

Data transmission ceases as soon as the send queue is empty. In addition, data transport can be stopped at all times, on both sides, by deregistering corresponding memory regions or by adjusting the QP into the reset or error state. The proper way to close a QP channel is to first process all work requests in the queues, then deregister all memory regions, destroy the completion queues & QPs and finally delete the protection domain and close the RoCE device.

4.4. Measurement automation

The test application is designed to explore various RDMA configurations and to determine whether the RoCE protocol can deliver the required performance for distributed telescope systems. Therefore, we must test the protocol's performance over an ample design space. We have automated this through two python scripts, the first allows us to iterate over a design space and use the required cluster resources for each test via *Slurm* jobs. The second script analyses all data from all tests and compute nodes via multiprocessing to gain insight into the performance achieved or bottlenecks encountered.

5 Results

This chapter presents the results of the different setups and configurations. First, in Section 5.1, we discuss and compare data transmission results via UDP and RoCE. The results of various RoCE configurations within a one-to-one setup are discussed in Section 5.2. The chapter then discusses the results of the many-to-one and one-to-many configurations in Section 5.3 and Section 5.4, respectively.

This chapter contains box plots to show the goodput and CPU utilisation. This reveals the distribution of data points and reflects the readings' stability. The box extends from the Q1 to Q3 quartile values of the data, with a line at the median (Q2). The whiskers run from the box's edges and extend to a maximum of $1.5 * (Q3 - Q1)$. We do not plot the outliers, as, in many cases, they negatively impact the readability of the plot. In addition, CPU utilisation is displayed as a percentage relative to the full load of one CPU core. Per configuration, the number of threads and cores used can differ. To compare them, we report the aggregated CPU utilisation of the threads used. So a CPU utilisation of 100% means that the total load is comparable to the load of an entire core, and 200% implies that the load is comparable to two fully utilised cores.

In our study, we are interested in the performance of the RoCE protocol over a long time and not in the effects during the start-up and shutdown of the data transport. Firstly, the network measurements during start-up can show various artefacts that can be highly dependent on the hardware and network settings used. So the start-up period would not so much give a picture of the RoCE protocol but mainly of the hardware and network settings used. Secondly, the aim is to see if the RoCE protocol is suitable for radio astronomy applications. RoCE is used continuously over an extended period (order of hours and days), where a steady data rate must be maintained in this application area. A short start-up and shutdown period is, therefore, acceptable. From this, it was chosen to test each configuration three times, with 10 seconds of data transmission per test. After that, only the middle 8 seconds are used for the goodput and CPU utilisation calculations.

All goodput and CPU utilisation measurements were taken per QP thread at an interval of 10ms unless otherwise stated. The interval of the measurements may differ due to the measurement method as it depends on the goodput. For this reason, and because of multi-QP measurements (i.e. multithreading), the data is resampled afterwards at an interval of 10ms to interpret the measurements better.

5.1. Comparison of UDP and RoCE

Table 5.1: UDP and RoCE performance comparison

Message size		9kB	32kB	64kB	1MB
UDP	Throughput	25 Gbps	43 Gbps	50 Gbps	-
	CPU utilisation	122%	145%	146%	-
RoCE	Throughput	92 Gbps	98 Gbps	97 Gbps	92 Gbps
	CPU utilisation	95.2%	77.6 %	43 %	7%

This section briefly compares the performance difference between UDP and RoCE. The protocols are evaluated between two nodes connected over a 100GbE network, more information regarding the (hardware) setup and methodology can be found in Section 3.1 and Subsection 3.5.1, respectively. The results shown in Table 5.1 have been acquired with Qperf. UDP achieved a maximum throughput of 50Gbps at a message

size of 64kB. Larger message sizes were not possible with UDP and resulted in NaN values. For any message format, RoCE outperforms UDP. In addition, it can be seen that when using UDP, CPU utilisation increases as message size and throughput increase. This is not the case for RoCE, where CPU utilisation decreases as the message size increases. The most significant performance difference is visible for the 64kB messages. Here the throughput of RoCE is almost twice as high, with only a third of the CPU utilisation required compared to UDP.

This brief comparison demonstrates the performance of RoCE over UDP. However, it is important to understand that both UDP and RoCE methods can be improved by optimisations such as offloading, multiple concurrent processes/streams and interrupt scheduling.

5.2. One-to-one setup

This section presents the results acquired in the one-to-one topology setup. In the following subsections, we discuss different configuration results. We use previous experiments' knowledge in subsequent subsections to test more purposeful and optimised configurations. The selection of configurations is based on providing an understanding of the impact of the RoCE features and arriving at a configuration that allows line rate transport for the smallest possible message size along with low CPU utilisation. Where applicable, the most relevant settings for each experiment will be stated; the remaining settings per experiment are available in Appendix B.1. In each subsection, we first discuss noteworthy outliers and then discuss the impact of the tested parameter.

5.2.1. Message size and memory page size

The first iteration of tests provides insights into the effects of the message size and memory type. The change in the settings per configuration is listed in Table 5.2 of which the results are shown in Figure 5.1 and Figure 5.2, the data source (requster) and sink (responder) node respectively.

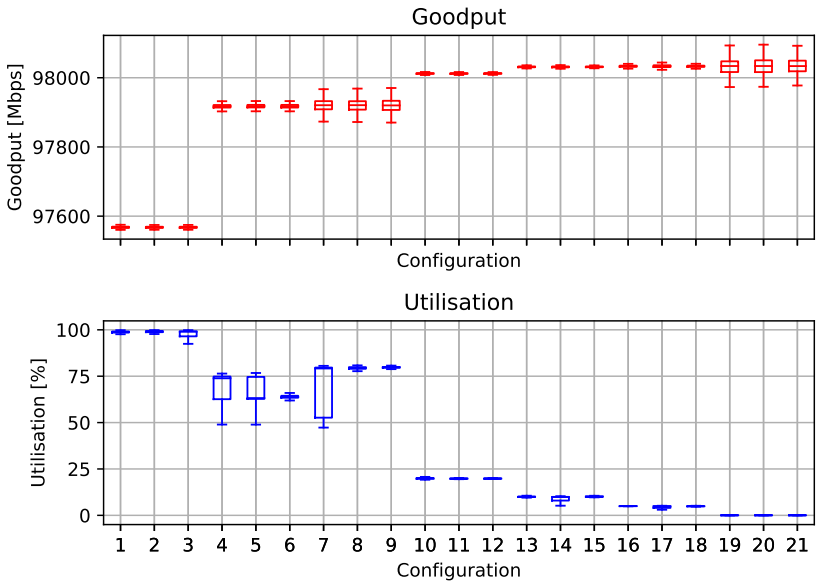


Figure 5.1: Results of message size and page type test for the requester node (configurations provided in Table 5.2)

Table 5.2: Configurations during message size and page type tests (used in Figures 5.1 and 5.2)

Configuration	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Page type	T	2M	1G	T	2M	1G	T	2M	1G	T	2M	1G	T	2M	1G	T	2M	1G	T	2M	1G
Size	4kiB			16kiB			50kiB			250kiB			500kiB			1MiB			200Mib		

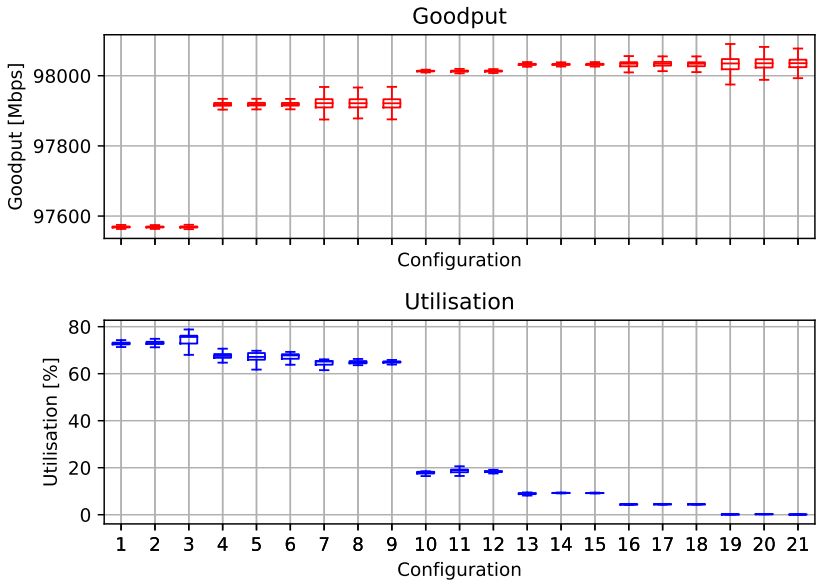


Figure 5.2: Results of one-to-one message size and page type test at responder node (configurations provided in Table 5.2)

First, both graphs show a relatively small standard deviation for each measurement. Though, the application reported an empty send and receive queue on several occasions when using a message size of 4kiB or 16kiB, regardless of the memory configuration. A more thorough investigation of the results during these measurements revealed that the responder recorded missed packets (less than 500kB of data) when the receiver queue was empty. More importantly, throughput dropped to around 65Gbps simultaneously and took about 350 milliseconds to recover to 97Gbps. This drop in throughput was also seen in the requester and thus appears to be an effect of a congestion management method which reduces throughput. However, it cannot be stated with certainty whether this was caused by a "hiccup" in the switch or by the receiver. The requester also reported multiple empty transmit queues during these configurations. However, these occurred at different times and did not seem to be related to the observations in the responder node. However, a short throughput degradation of 10 to 20Gbps is visible for a single measurement interval when an empty queue is registered.

Secondly, the goodput rate was higher than 98Gbps for all configurations with a message size between 50kiB and 200MiB. Besides, these configurations did not report any message loss or empty queues.

Message size

From the figures, it can be seen that the message size impacts both the throughput and CPU utilisation. The CPU utilisation decreases as the message size increases in both the requester and responder nodes. This is likely a result of a decrease in message rate since the goodput is nearly constant as the message size grows. A lower message rate reduces the CPU's workload since the rate of WR decreases, thus reducing the interaction with the RNIC and the amount of memory state tracking. The results also show that the CPU utilisation of the requester is higher than the receiver for each configuration. One viable reason is that the corresponding memory regions must be looked up and passed along to the WR for each message.

Memory page type

The second aim of these tests is to identify the impact of the memory page type. Comparing the results of equivalent message sizes provides insight into the effect of page types. This shows neither clear nor significant differences in goodput or CPU utilisation in Figures 5.1 and 5.2. Though, it is important to note that the transported data is not used during the measurements. In other words, the test application does not perform calculations on the memory used. An application using these memory blocks can benefit from a particular page type as it reduces cache misses. From these tests, it can be concluded that RoCE can use 2MB and 1GB pages without performance degradation. Moreover, on this (small) scale, there is no advantage or disadvantage to using the memory type.

5.2.2. Linked work requests and solicited events

The previous design space exploration showed that the RNIC could saturate a 100GbE connection with a message size starting from 16kiB. At the same time, the CPU utilisation decreased as the message size increased. This suggests room for improvement regarding CPU utilisation when using small-sized messages. In addition, it makes less sense to further optimise low utilisation and high throughput configurations. For this reason, we used 4kiB and 16kiB message sizes in the results discussed in this subsection.

In these tests, the interval at which solicited events are sent is linked to the number of linked WR. If the solicited events mechanism is used, the program sends a single solicited message per linked work request. Table 5.3 lists each configuration's (varying) settings.

The goodput and CPU utilisation corresponding to the responder node is provided as boxplots in Figure 5.3. Note the goodput deterioration in configurations 1 and 6. A few congestion notifications (<300) were sent, and a negligible amount of message loss was reported for these tests. Besides, the CPU utilisation of the requester node was continuously close to 100% during these tests. This is also the maximum possible because a single QP and, thus, a single thread were used. In addition, the send queue utilisation in the requester was often exceptionally low (below 10%), and the queue often became empty. These readings suggest that a limitation in the requester node caused the reduction in goodput. More specifically, the limitation occurs due to the CPU overhead introduced by posting a single WR at a time to the RNIC since the goodput rate increased when the number of linked WR increased, while all other settings remained unchanged.

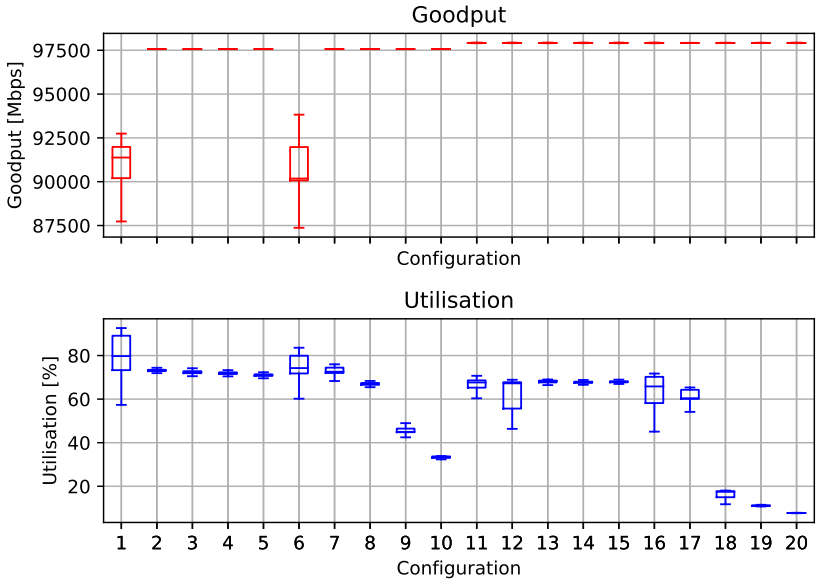


Figure 5.3: Results of one-to-one linked WR and SE test at responder node (configurations provided in Table 5.3)

Table 5.3: Configurations during linked WR and solicited events tests (used in Figures 5.3 and 5.5)

Configuration	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
#Linked WR	1	5	25	50	100	1	5	25	50	100	1	5	25	50	100	1	5	25	50	100
Solicited events	no					yes					no					yes				
Size	4kiB										16kiB									

The time series plot of a single run using configuration 1 is shown in Figure 5.4. The goodput decreases three times to 55Gbps, whereafter it recovers to 92Gbps. An empty receive queue and a small amount of message loss were recorded at the beginning of these reductions. Such a reduction also occurs in some other configurations, albeit less frequently. A recurring phenomenon in these reductions is the registration of an empty receive queue. So it appears that congestion mechanisms, such as pause frames, are sent when a RoCE operation requires receive requests which are not present. This was not as expected because RoCE's documentation dictates that messages will be dropped if no receive requests are present. Moreover, it was expected that backpressure from the RNIC would be treated differently from the lack of work requests. This is because the RNIC can receive and process the messages but does not have a work request for this and should

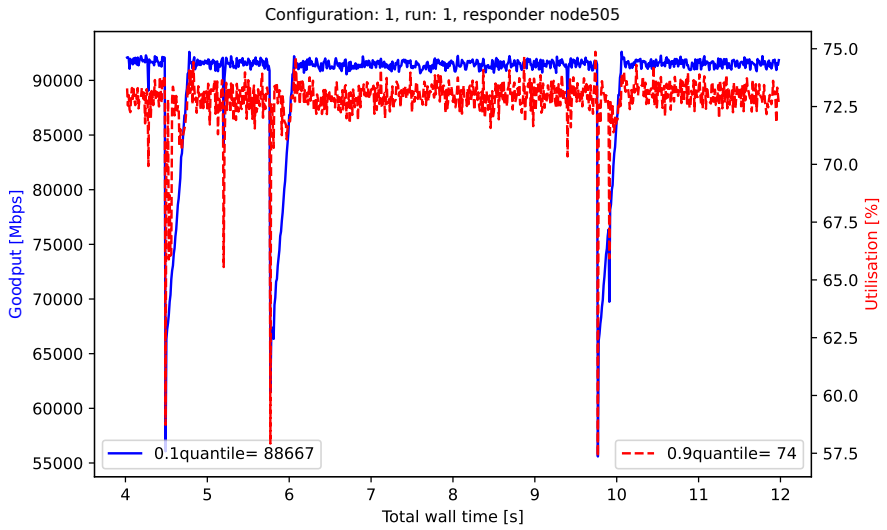


Figure 5.4: Plot of the throughput and CPU utilisation, configuration 1 of the linked WR tests during one-to-one setup

therefore drop the message. Yet, it appears that PFC intervenes and causes the transmitter to send data at a lower rate.

Impact of linked WR

The effect of linked WR on CPU utilisation (without the usage of solicited events) is not significant in the responder during both the 4kiB and 16kB tests, as the average CPU utilisation differed at most 5%. Especially considering a common deviation in CPU utilisation of 2% between equivalent repetitions. Yet, from the goodput results, it is clear that a message rate limit is reached if no linked WR is used, along with a message size of 4kB.

The effect of the linked WR on utilisation is different for the requester, as shown in Figure 5.5. The CPU utilisation decreases as the number of linked work requests increase for the 4kiB and are equal when using solicited events, as expected. The 16kiB configurations show a similar trend in CPU utilisation when the number of linked WR increases. Though, the utilisation is higher when using solicited events. This is expected to be a deviation because we measure on a non-real-time system, so background processes may also have an influence. In addition, the CPU utilisation always exhibits a relatively large standard deviation in the requester.

Impact of solicited events

More significant improvements are achieved when leveraging solicited events for 4kiB and 16kiB configurations. The decrease in utilisation originates from the reduced number of notifications and, thus, fewer interrupts. The reduction in the number of notifications also means that more work completions are retrieved through a single API call. This further reduces the required interaction between the application and the verbs API and RNIC.

The improvement becomes substantial in 4kiB configurations when the interval of solicited events increases to 50 and 100. Whereas the improvement already becomes significant with an interval of 25 for runs

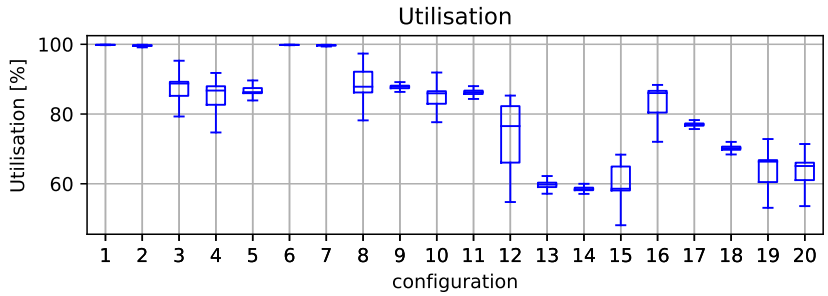


Figure 5.5: CPU utilisation for different linked WR values in the requester (configurations provided in Table 5.3)

with a message size of 16kiB. The CPU utilisation is reduced by almost 75% when using solicited events at an interval of 25 messages with 16kiB messages, compared to its equivalent configuration without solicited events. A similar comparison for the tests with a solicited interval of 100 messages when using messages of 4kiB shows a reduction of 52% in CPU utilisation.

Notice in Figure 5.3 stagnation of the advantage of solicited events for 16kiB configuration with an interval of 50 and 100 solicited events, respectively configuration 19 and 20. The lower notification interval reduces the workload related to receiving notifications and retrieving work completions. However, the number of messages and immediate data to be processed does not change through the use of solicited events. So at some point, the basic workload to process the number of messages will become apparent. And therefore, the benefit of larger solicited event intervals will stagnate.

From the preceding discussion of the results, it can be concluded that posting linked WR may mainly impact the requester and barely impact the responder. In addition, linked work requests can reduce or prevent a bottleneck in the communication between the CPU and RNIC whenever there is a high message rate. Lastly, the usage of solicited events positively affects the CPU utilisation of the responder. Though the benefit stagnates at higher intervals, the interval where after stagnation occurs depends on the message size.

5.2.3. QP scaling and shared receive queue

An important characteristic of data transport in the application of radio astronomy is the support for multiple concurrent connections. We refer to the support of multiple concurrent connections of a protocol as the protocol's scalability. In this subsection, we take the first step towards testing the scalability of RoCE between 2 nodes by increasing the number of QPs. In general, using multiple QPs in a single node can be beneficial when using multiple parallel applications, processes or threads that need to communicate with a remote node. In this section, the results and effects of the responder node are discussed. For completeness, equivalent plots of the requester are provided in Appendix B.2. In short, the requester has a higher utilisation compared to the responder. This could be reduced by signalling the completion events of fewer WR instead of every WR, as discussed in Subsection 2.4.7.

The previous experiment revealed that solicited events positively affect the responder's CPU utilisation. In addition, using 50 and 100 linked WR gave a significant additional benefit when using 4kiB message sizes. During the following measurements, the number of QPs increases, which will put more stress on the CPU because it has to keep track of more connections and memory regions. It was decided to use solicited events with 100 linked WR to reduce CPU utilisation without compromising insight into processed messages. The other settings that are not varied during the following measurements are provided in Appendix B.1.

Scalability using separate threads

During these tests, the program processes each QP in a separate thread and each QP has a private send and receive queue and a dedicated completion queue. The message size and the number of QPs per configuration are listed in Table 5.4. Figure 5.6 shows the achieved performance when the number of concurrent QPs increases between two end nodes. Note that the goodput is less than 1Gbps below the theoretical maximum of 98.1 and 98.5 Gbps, respectively, for 4kiB and 16kiB. Moreover, the goodput was relatively stable, and a few reductions in throughput were encountered during a few runs. More importantly, no significant or constant congestion was observed during any run. The figure reveals a clear difference between the goodput of each 4kiB and 16kiB test, with the 16kiB configurations being 330Mbps higher.

For CPU utilisation, the 16kiB configuration consistently outperforms the 4kiB configuration, with the difference increasing as the number of QPs increases. In addition, the CPU utilisation for the 4kiB configuration increases drastically once the number of QPs increases to 100 and 500. The first part of the cause is the increased context switching as the number of threads increases beyond the number of cores. Second, the 4kiB configuration has four times more notification events to handle, which increases the number of context switches compared to 16kiB configurations. Thirdly, the threads of the 4kiB configurations have a higher workload as they need to process four times as many messages, each requiring immediate data checks and placement of work requests. The third reason already causes an increase in CPU utilisation at lower QPs, whose impact was also covered in previous subsections. Yet, it plays a magnifying effect as the number of QPs increases since the threads have lower waiting (or idle) times, causing larger context-switching overhead.

The increase of CPU utilisation during 16kiB configurations is more gradual and thus encounters a minor penalty as the number of QPs increases. Therefore, this result is more in line with our expectations, as the total workload should not increase significantly as the aggregated message rate remains equal. So we should only see a (small) multithreading overhead.

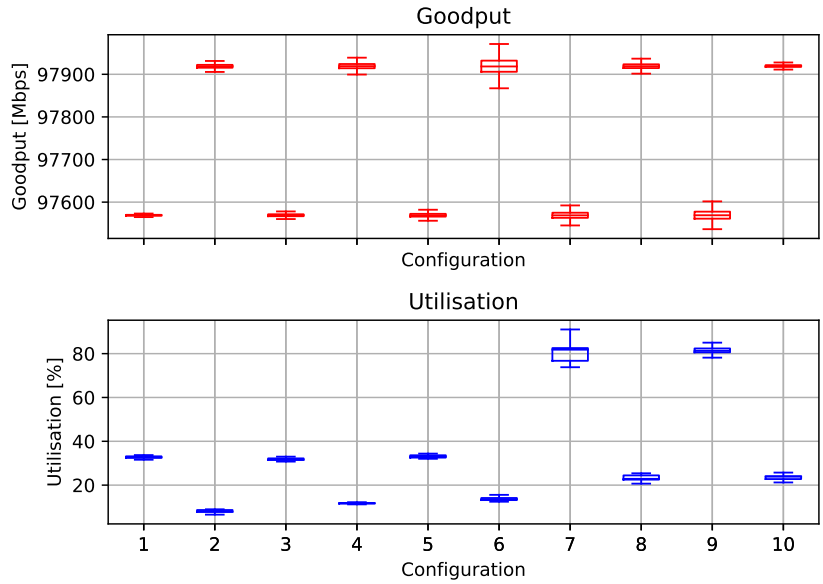


Figure 5.6: Results of one-to-one QP scalability test at responder node without SRQ (configurations provided in Table 5.4)

Table 5.4: Configurations during queue pair scaling test using one-to-one setup (used in Figure 5.6)

Configuration	1	2	3	4	5	6	7	8	9	10
Size [kiB]	4	16	4	16	4	16	4	16	4	16
QPs	1		5		10		100		500	

Scalability using a shared receive queue

Examining the use of the Shared Receive Queue is worthwhile as the number of QPs increases. The performance of the previous QP scalability test was satisfactory. Besides, we are interested in even higher scalability at a later stage. Therefore, we use 100, 500 and 750 QPs in this test with 4kiB, 8kiB, 16kiB message sizes, as assigned to the configurations in Table 5.5. Note that the 8kiB results were done at a later stage because the 4kiB configurations performed less than expected. All QPs are processed using the same SRQ and completion queue in a single thread in the responder. The queue size for both the receive queue and completion queue has been increased by a factor of 2 compared to previous tests, as it is expected to take more time to find the data of each QP in an array. Increasing the queue size can help bridge this extra time to prevent a queue from becoming empty. All other configuration values remained unchanged compared to the previous scalability tests, such as 100 linked work requests and the use of solicited events.

The responder results for the SRQ configurations are shown in Figure 5.7. The goodput of all 4kiB configurations is exceptionally low, with an average below 60Gbps. In addition, the NIC counters also indicate that congestion was severe. The requester did not cause this degradation since identical settings were used in previous tests when testing the scalability without an SRQ. In addition, a high amount of network congestion was observed during the 4kiB configurations. Further investigation showed that this congestion was caused by backpressure in the responder. The responder cannot process the data stream fast enough, causing it to send congestion notifications to the requester and pause frames to the network switch. The requester then reduces the data rate to reduce the congestion.

One reason for the degradation might be that the message rate is too high to handle in a single SRQ. A NIC often uses parallel processing elements. In these tests, a single SRQ is used to process all messages which belong to various QP resources. As a result, there is only one SRQ for all QPs, which possibly constrains the parallelism, and that, together with a high message rate, can lead to a possible bottleneck in the RNIC. We have not been able to validate this due to the many modifications required to support multiple SRQ threads in our test application. Instead, 8kiB configurations are measured because one of the aims is to achieve a close-to-line rate goodput with the lowest possible message size and low CPU utilisation. For the 8kiB, an immediate increase in goodput and slightly lower CPU utilisation can be seen, compared to the 4kiB. Also note that the goodput deviation does increase for the 8kiB configuration as the number of QPs increases. No empty queues were recorded for all measurements, however, it is visible from the detailed results that the

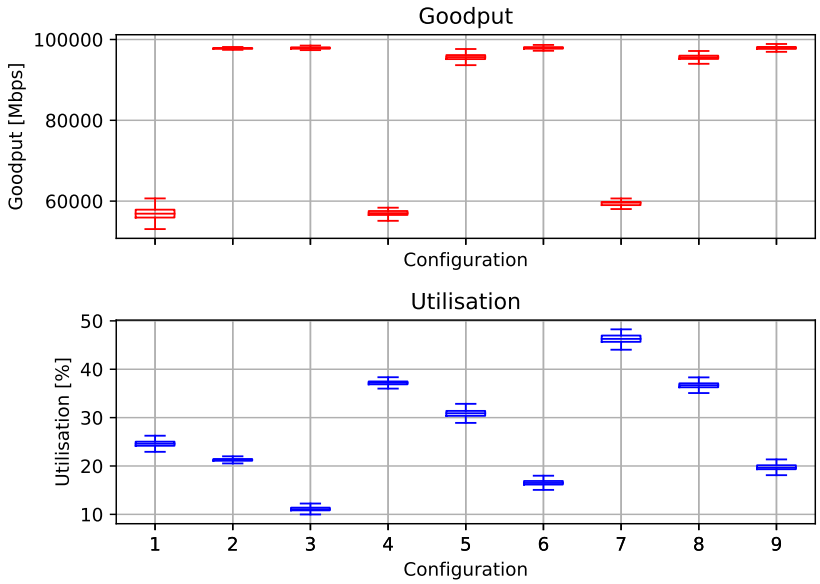


Figure 5.7: Results of one-to-one shared receive queue test at responder node (configurations provided in Table 5.5

Table 5.5: Configurations used queue pair scaling with SRQ test using one-to-one setup (used in Figure 5.7)

configuration	1	2	3	4	5	6	7	8	9
Size [kiB]	4	8	16	4	8	16	4	8	16
QPs	100			500			750		

queues for the 4kiB and 8kiB configurations regularly fall below 50% utilisation while for the 16kiB configuration this remains, with a few exceptions, above 65%. Compared to previous measurements, more fluctuation is observable in queue utilisation. The fluctuation is now more noticeable partly because there are far fewer queue elements. To illustrate, without the SRQ, 2 queues were used per QP (a receive queue and a completion queue). So for 500 QPs, a total of 1000 queues were used with 1000 elements each. For the same number of QPs, using an SRQ, only 2 queues are used, each of which had 2000 elements. In the many-to-one setup, the queue size will be increased. First, because without increasing the queue size, the number of QPs are equal to available elements in the receive queue. Secondly, to reduce the possible negative effect of too small SRQ with the knowledge that still many resources are saved by using the SRQ. Notice in Figure 5.7 that the CPU utilisation increases incrementally with the number of QPs, while the goodput remains relatively constant. For each WC received, the responder has to retrieve the corresponding QP data in a QP information array to check the immediate data and flag the corresponding memory block. This causes an increase in CPU utilisation since the computational overhead increases as the QP information array (in which to search for the correct QP) becomes larger.

A comparison of the 16kiB configurations with and without SRQ is given in Table 5.6. This signifies a positive effect of the SRQ on CPU utilisation; however, the advantage reduces as the number of QPs increases. This can be improved by running multiple SRQs or improving the arrays in the application, reducing the CPU overhead due to searching and retrieving the QP data. After all, one of the ancillary benefits of distributing QPs over more threads is the increase in cache memory, making the QP-related data more quickly available. The results of 4kiB configurations with and without SRQ are not compared since the goodput is lower when using the SRQ. It is expected that CPU utilisation always increases when the message rate rises, so it is impossible to compare them properly.

Table 5.6: 10% quantile results of the (aggregated) CPU utilisation, one-to-one QP scaling

Number of QPs	100	500	750
Seperate threads	23%	24%	-
Shared receive queue	11%	17%	20%

5.2.4. GPU memory via peerDirect and SRQ

The RNIC can directly access part of the GPU memory through the PCIe interface by using the GPU peerDirect configuration described in Section 2.6. Consequently, the data traverses the PCIe bus once, reducing the load on the PCIe bus. Using the GPU memory in this manner requires the Mellanox and Nvidia drivers to be installed appropriately, as detailed in Subsection 2.4.8. Though, some Linux kernels are also not (yet) supported. Because of these restrictions and the use of a research compute cluster, which others also use, it has only been possible to use the GPU memory via DAS6 node505. Besides, note that the CPU still handles the control of RoCE and thus, the GPU only receives or sends data.

Two message sizes were used to investigate the performance of RoCE with GPU memory; the results are shown in Figure 5.8 and the corresponding configurations Table 5.7. The goodput decreases significantly as the number of QPs increases. The number of QPs, whereafter the goodput decreases, is lower for the 8kiB configuration than for the 16kiB configurations. It is also noticeable that the goodput fluctuates significantly when using one QP. The fluctuation is substantially less when 10 or 100 QPs are used along with 16kiB message sizes, configuration four and six, respectively.

The trend of CPU utilisation with increasing QPs is consistent with previous measurements. However, the increase is lower than during the SRQ test using main memory, likely due to the decreased data rate. As a result, only configuration 6 can be compared with configuration 2 of the previous SRQ test, which is listed in Table 5.5 and shown in Figure 5.7. Both configurations achieve a CPU utilisation of around 11%. The 10% quantiles are 94.5Gbps and 97.5Gbps for configuration 6 and SRQ main memory test configuration 2, respectively. Therefore, this difference is only 3Gbps, and the averages are closer. The main difference is a systematic decrease in goodput when using GPU memory, which is not present in previous measurements.

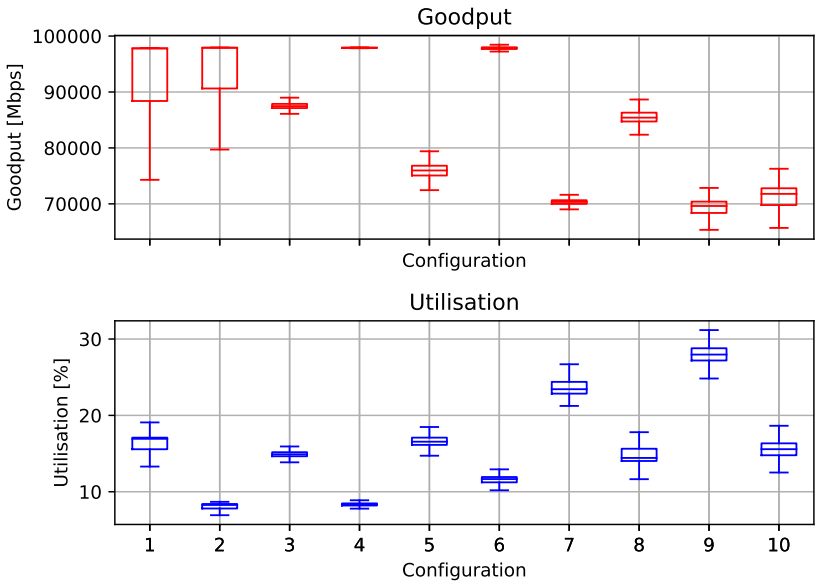
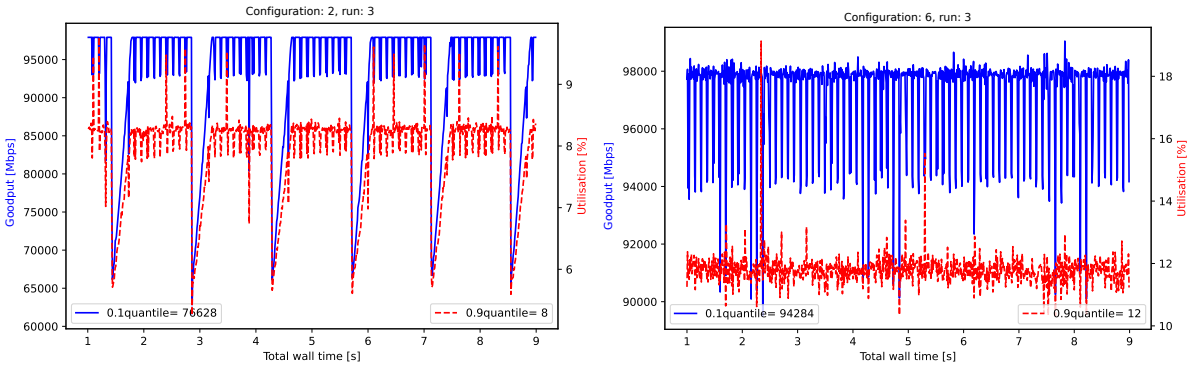


Figure 5.8: Results of one-to-one GPU tests at responder node (configurations provided in Table 5.7)

Table 5.7: Configurations used during GPU test in one-to-one setup (used in Figure 5.8)

Configuration	1	2	3	4	5	6	7	8	9	10
Size [kiB]	8	16	8	16	8	16	8	16	8	16
QPs	1		10		100		500		750	

Figures 5.9a and 5.9b show the trajectory of the (aggregated) goodput and utilisation over time for configurations 1 and 6, respectively. These figures are provided in larger dimensions in Appendix B.2 Figure B.5. Notice the large drops in goodput down to 65Gbps and the more frequent drops to 88Gbps at intervals of 1.2 and 0.1 seconds in Figure 5.9a. The interval of these reductions is constant over time, and the amount of reduction is also remarkably consistent. We do not observe this phenomenon in configurations that do not utilise GPU memory. In addition, this phenomenon is only observed during configurations 1, 2, 4 and 6, as



(a) GPU throughput and utilisation using 1QP and message size of 16kiB

(b) GPU throughput and utilisation using 100QP and message size of 16kiB.

Figure 5.9: Time series plots of throughput and utilisation in one-to-one setup using GPU memory

listed in Table 5.7. To point out, the configurations in which a throughput of around 97.9Gbps is achieved suffer from these consistent short-lived decreases in goodput. From the number of congestion notifications during these measurements, it is clear that these anomalies result from backpressure due to GPU memory usage. However, it is unclear what caused the backpressure; using fewer memory regions, a larger receive queue, or a larger message size did not improve the goodput results. A possible cause might be reduced bandwidth to the memory due to TLB misses. This might be overcome by allocating the full BAR size once via the CUDA kernel API and ensuring that the separated memory regions are connected continuously in the GPU memory.

GPU utilisation and consumption were measured in a separate test with the same configurations by querying the parameters at an interval of ten milliseconds using *nvidia-smi*. The idle consumption of the A100 was measured to be around 36 watts, which increases to 45 watts during memory allocation and deallocation. The power consumption during the data transport is about 2 to 4 watts higher than idle consumption. Both GPU and memory utilisation are reported at 0%. The low power consumption and utilisation increase are partly due to the absence of a compute kernel that uses the received data.

Moreover, the lack of a CUDA kernel meant it was impossible to profile the transport via another tool, such as *Nvidia Nsight systems*. It is unclear whether such direct memory access would be reported as it does not occur via the CPU. Adding a CUDA kernel and measuring performance on the GPU remains as future work.

5.2.5. Conclusion

In summary, the following conclusions were derived from the experiments in this section:

- Increasing the message size reduces the CPU utilisation in both the requester and responder as the notification frequency is reduced. Moreover, fewer immediate data and work completions must be checked, reducing CPU utilisation.
- Tests have shown that up to 200MiB message sizes can be transferred. However, the data of an entire message is lost if a single Ethernet frame is not received. We have shown that with 16kiB we already achieve a good performance for our use case (goodput of 98Gbps and CPU utilisation of 75% using one QP without optimisations).
- The use of transparent huge pages, 2MB or 1GB huge pages, has shown no significant advantages or disadvantages when using a single QP.
- Linked work requests have more impact on requesters than responder QPs. We identified that posting a single work request at a time (for small message sizes) does not reach the full performance and that linked work request should be used instead.
- Solicited events reduce the CPU utilisation of the responder and have no negative effect on the goodput. Though, if the interval becomes too large, the reduction in CPU utilisation will stagnate. This is because the amount of messages and immediate data does not change per time interval, so there will always be a minimum load to process them.

- It is possible to use up to 500 QPs with a one-to-one connection with only 23% CPU utilisation and goodput of 98Gbps by using threads per QP, 16kiB message size, solicited events and link work requests of 100.
- Using 4kiB message sizes compared to 16kiB causes four times higher CPU utilisation at a larger number of QPs. This makes sense as there are four times as many work requests (and thus immediate data) to process. This also provides four times more fine-grained insight into which memory parts contain data. However, this factor of 4 was previously not reflected in the CPU utilisation, so with a lower number of QPs, the penalty of a smaller message size is less.
- An SRQ reduces CPU utilisation by more than 50% for 100QPs. However, this benefit decreases as the number of QPs increases. The main reason for this is the increasing inefficiency of searching a QP's data in an array that scales with the number of QPs. In addition, configurations with a message size of 4kiB achieve low goodput. This is probably due to the high message rate without sufficient parallelism in the RNIC because of the single SRQ. In the future, these issues can be addressed by using multiple SRQs or by improving the search algorithm and data locality of the QP data.
- Direct memory access to GPU memory is possible from the RNIC and has no effect on CPU utilisation compared to the use of main memory. However, the goodput does degrade when using more QPs. In addition, systematic short-lived reductions in goodput are observable when a goodput rate of 98Gbps is obtained, for which the cause is unknown.

From the results, we can evaluate some of the requirements established for transport methods for the use cases considered in this thesis. It is evident that RoCE packets can be routed over networks as standard UDP Ethernet packets, and thus the RoCE protocol complies with R2. Furthermore, the measurements have shown that the program provides insight into (incorrectly) received messages and data, as stipulated in requirement R6, when using the WRITE with immediate data operation and the unreliable connection transport service. Lastly, this operation and transport method satisfies requirement R5 because these settings do not demand retransmission if something goes wrong during data transfer. In the subsequent sections of this chapter, we will conduct tests to validate requirements R3 and R4, which address network topologies and the minimum goodput.

5.3. Many-to-one setup

This section presents our results acquired in the many-to-one topology setup. We use four similar nodes as requesters of the RDMA WRITE operation, which write concurrently into a single server node. While developing the scalability capability in the application, it was noticed that high congestion occurs when we connect 4x100GbE to 1x100GbE. This caused significant fluctuations in the goodput due to the PFC and RoCE congestion management mechanisms. In our research, we want to test the RoCE protocol and its application in radio astronomy systems. A key characteristic of this application area is a stable data rate from data-sending FPGAs over a network where the number of end nodes does not change during measurement. As a result, it is known in advance which data rate needs to be processed and from how many transmitters it originates. Because of these properties, we decided to limit the bandwidth of the (maximum) four transmitters to 23Gbps and thus an aggregated bandwidth of 92Gbps.

In addition, the ECN thresholds in the switch have been adjusted. This is because the buffers in the switch are more heavily loaded in a many-to-one setup than in a one-to-one because multiple data streams have to be merged. The ECN minimum threshold is set to 4MB and the maximum threshold to 6MB.

In this section, we first discuss the results of the QPs in separate threads, then the use of the SRQ in this setup and lastly, the use of GPU memory. The section concludes with a comparison of these configurations.

5.3.1. Scalability using separate threads

Processing each QP in a separate thread is similar to having a different process per connection, which can be desirable if one has a separate processing pipeline for (part of) the connections. The scalability is tested up to 2000 QPs in the responder node, thus up to 500 QPs per client. More information regarding the methodology is provided in Subsection 3.5.3.

Figure 5.10 depicts the performance of the responder node. The system does not achieve its maximum theoretical performance as the number of QPs and client nodes increases. When using 40 QPs or more, a decrease in goodput and an increase in congestion notifications are noticed. During the one-to-one setup, it

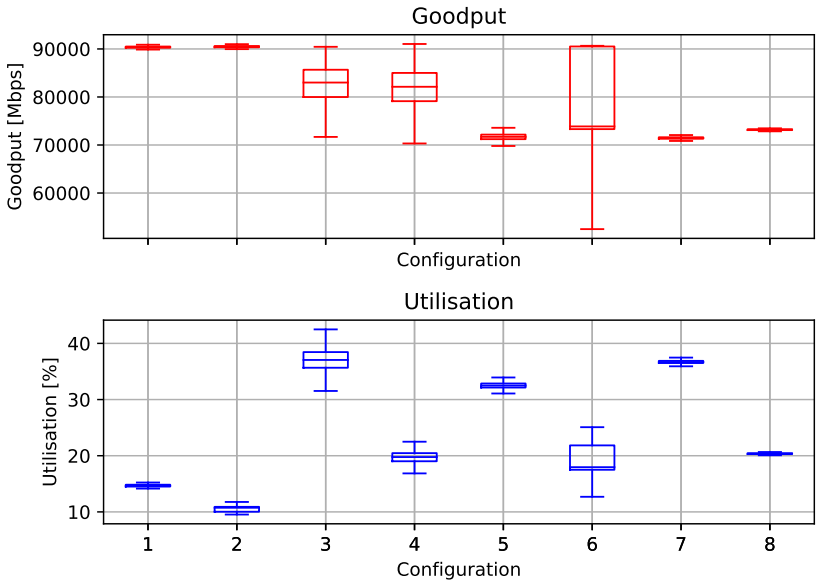


Figure 5.10: Results of many-to-one without SRQ at responder node (configurations provided in Table 5.8)

Table 5.8: Configurations during queue pair scaling test using many-to-one setup (used in Figures 5.10 and 5.12)

configuration	1	2	3	4	5	6	7	8
Size [kiB]	8	16	8	16	8	16	8	16
QPs	4		40		400		2000	

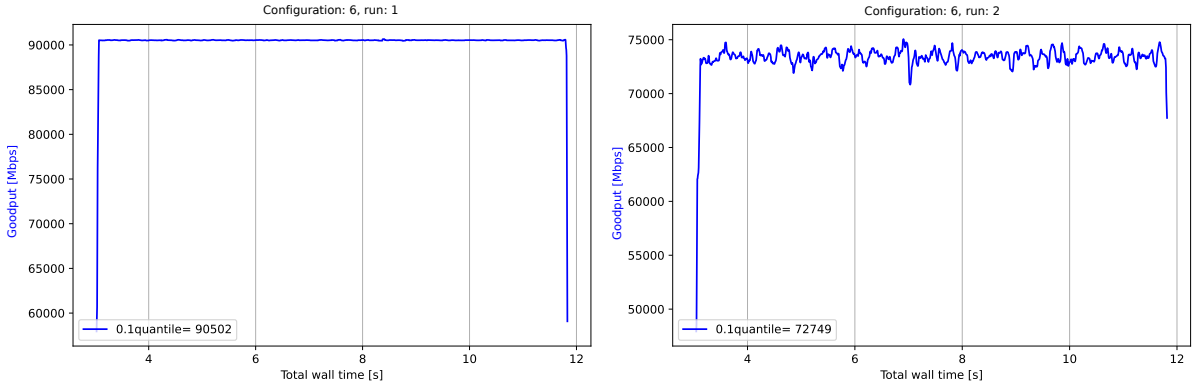
was confirmed that the test application could process 500 QPs simultaneously from a single requester with a goodput of 97.9Gbps. The one-to-one setup using 500 QPs is superior to the results obtained using 400 QPs in the many-to-one setup, which resulted in an average goodput of around 72Gbps for both message sizes.

Configurations 3, 4 and 6 report a large distribution of the goodput. For configurations 3 and 4, the results showed that the goodput varied greatly during the measurement for each of the three iterations. For better insight, a time series plot of the goodput for configuration 3 is provided in Appendix B.4 Figure B.7. This discrepancy is unexpected because these configurations use as few as 40QPs, and previously good results were obtained for this order of magnitude of QPs. The number of memory regions, queue formats and measurement intervals remained the same compared to the one-to-one setup. We, therefore, expect no significant increase in RNIC resources or negative impact of conducting a measurement. In addition, queue utilisation was always high during each measurement and never reported as empty. This discrepancy was also noticed for 40 QP configurations using SRQ, while a higher number of QPs using SRQ achieved good results. Hence, we cannot identify a clear justification for these two configurations' large degradation in goodput.

The measurements of configuration 6 report a distribution of the goodput between 90Gbps and 55Gbps. A better analysis shows that a stable goodput of 90Gbps was achieved during run 1 of the three iterations of this configuration, as depicted in Figure 5.11a. While a subsequent test (run 2) with exactly the same settings achieved a goodput of around 72Gbps, Figure 5.11b. For run 2, many congestion notification messages were sent, while almost no congestion notifications were sent for run 1. No differences could be noticed in the queue utilisation between the runs, and it was also verified that no one else was using the network at the time of the measurements. We expect this performance difference to result from PFC intervening too hard caused by start-up glitches. It is noticed that at start-ups, the ETS rate limit is sometimes exceeded very briefly. In addition, the first activated QP threads have a high data rate at start-up for a very short time, whereas this peak decreases or even disappears for subsequently launched threads. We, therefore, expect that this behaviour can be improved by implementing a slow start mechanism, for example, by filling the send queues in the requester threads more slowly at start-up. In our setup, we cannot observe the PFC behaviour in detail, so a definite cause cannot be ascertained.

The goodput deviation appears small for the 2000 QPs configuration; however, this reflects the aggregated goodput. An examination of the goodput per QP reveals a variation of up to 20% from the average throughput

of a QP in the requester. A more significant deviation in the goodput per QP was also measured for the other configurations of this test compared to the one-to-one test setup. Yet, this variation is not as visible in the responder node, which could be attributable to the buffer space in the switch.



(a) Stable throughput during run 1

(b) Instable and lower throughput during run 2

Figure 5.11: Time series plots of the throughput for two runs using configuration 6 in Table 5.8

5.3.2. Shared receive queue thread

The SRQ and associated completion queue are increased to 20.000 elements to ensure that each QP can consume multiple receive requests before replenishing the queues. The queue size of requesters is not modified since the number of QPs per client does not increase, and also, no issues were signalled in the requester nodes in the previous measurement.

Note that the performance of the SRQ configuration, as shown in Figure 5.12, is better than the previous test with separate threads. The goodput has a 10% quantile of at least 89.9Gbps for all configurations except configurations 3 and 4, which use 40 QPs. Also note that unlike the results of the one-on-one setup using the SRQ, the configurations with 8kiB have no (additional) goodput degradation compared to the 16kiB configuration. This is most likely a result of the Ethernet rate limit being lower than the rate at which the bottleneck occurred in the previous setup.

Up to 6 times more congestion notifications were sent in the tests of configurations 3 and 4 compared to the other configurations. In addition, the goodput varied considerably for each of the three measurements

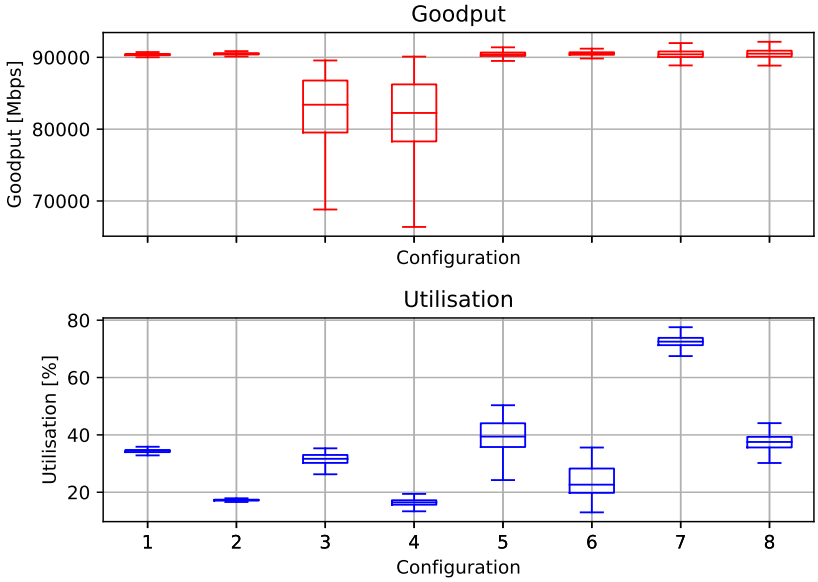


Figure 5.12: Results of many-to-one SRQ at responder node (configurations provided in Table 5.8)

per configuration; in other words, each of the three measurements achieved equal distribution in goodput. The deviation in the goodput corresponds thus to the result for the equivalent configuration in the previous test with threads per QP. The anomaly is unique to the use of 40 QPs in the responder and was not examined in more detail because the configurations with more QPs did perform well.

For the other configurations, the variation of the goodput is smaller. Yet, we noticed that the variation in goodput is more prominent in the responder than in the requester. Which is contrary to what was previously observed in the many-to-one measurement using threads per QP. The QPs in the transmitters now have less than 2% deviation from their average goodput, while in the responder, there are many outliers of 5% duration of a single data point. However, these variations are so small and less than the 10% variation prescribed by the ETS protocol that no further research has been devoted to this.

The utilisation for the smaller message size increases more strongly as the number of QPs increases. And note that CPU utilisation for 16kiB configurations is between 30% and 50% lower than for 8kiB configurations. The underlying reasons for this are expected to be similar to those mentioned earlier for the QP scaling using the one-to-one setup.

5.3.3. GPU memory and shared receive queue

This subsection discusses the results obtained while using GPU memory in the responder node; all other settings are the same as those discussed in the previous subsection. Configurations without the SRQ are not tested since configurations with separate threads per QP, as discussed in Subsection 5.3.1, resulted in poor performance.

Interesting to note from Figure 5.13 is that the goodput when using 2000 QPs with GPU memory is, on average, above 90Gbps. In addition, the goodput and CPU utilisation averages of the other configurations, except configuration 7, are equivalent to the SRQ test with main memory. Besides the lower goodput for configuration 7, four more findings are worth noting. First, the goodput variation is slightly larger than reported in the SRQ test with main memory tests. Secondly, the goodput is improved compared to the one-to-one setup with GPU memory. Compared to the one-to-one setup, there are fewer congestion notifications (except for configuration 7). In addition, we do not observe a systematic short-lived reduction of the goodput for any measurement, as was present in the one-to-one GPU measurement discussed in Subsection 5.2.4. Firstly, this could result from the limited aggregated throughput of 92Gbps. Secondly, the ECN threshold beyond which the switch sends notifications has been increased, allowing better handling of short-term congestion. Another difference from the one-to-one setup is the larger receive queue in the responder; however, we do not expect this to have had any (significant) effect. After all, the (smaller) receive queue did not seem to have been a problem in the one-to-one measurement because it was constantly populated for more than 60%. In short, we expect the rate limit and higher ECN thresholds to have positively affected the goodput.

Thirdly, it can be observed that the configurations using 40 QPs suffer from the same goodput degradation as the equivalent configuration during the SRQ test with main memory. Hence, this degradation is not expected to be caused by GPU memory. Lastly, the analysis of the individual measurements shows that all 16kiB configurations have significantly lower congestion and minor fluctuation in goodput than the 8kiB configurations. One reason for this might be that this larger message size requires fewer memory translations per unit of time because the message rate is lower. As a result, the RNIC suffers less from cache misses.

As noted earlier, the goodput for configuration 7, which uses 2000 QPs and a message size of 8kiB, is lower than the configured rate limit of 92Gbps. The queue utilisation in the requesters and responder was high and thus did not pose a problem. Though a lot of congestion notifications were sent indicating backpressure from the responder. The same configuration without GPU memory achieved a goodput of over 90Gbps, as presented in Figure 5.12, which proves that the number of QPs and the message rate can be processed in a single SRQ thread. So the problem seems to occur due to writing data to GPU memory, yet, it resolves when using a message size of 16kiB. As the number of QPs increases, the number of memory blocks used increases proportionally. And so, there are five times more memory locations used by the received operations when using 2000 QPs compared to 400 QPs. On top of that, the memory locality of consecutive received operations in the responder decreases as the number of QPs increases. Consequently, the number of TLB misses is expected to increase when the number of QPs increases due to poorer memory coalescing. A TLB miss increases the time before data can be written, increasing the pressure on the ingress buffer.

Yet configurations using larger message sizes or main memory do not suffer from the increase in TLB misses. The page type was set to transparent huge pages when using main memory. So it is feasible that the OS used a page size of 2MB or 1GB, decreasing TLB misses. However, this was not validated by enforcing a 4kB or 2MB page size and thus remains for future work. In addition, a larger message size (and its resulting

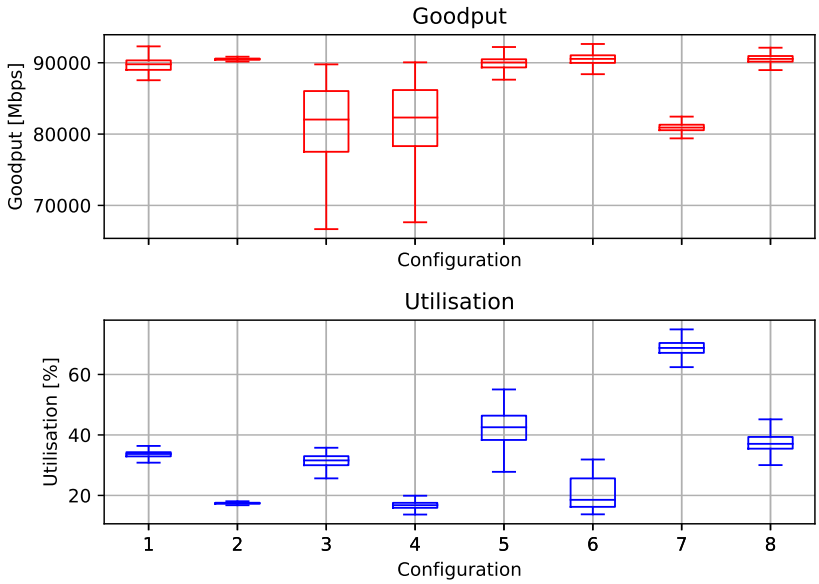


Figure 5.13: Results of many-to-one GPU SRQ at responder node (configurations provided in Table 5.9)

Table 5.9: Configurations during queue pair scaling test using many-to-one setup and GPU memory (used in Figure 5.13)

configuration	1	2	3	4	5	6	7	8
Size [kiB]	8	16	8	16	8	16	8	16
QPs	4		40		400		2000	

lower message rate) also reduces the number of TLB misses per unit of time. The lower message rate reduces the number of (scattered) memory blocks needed in a time unit. In addition, RoCE Ethernet packets are required for a single message whose memory addresses are near each other, reducing the probability of a TLB miss. And so, we estimate that the impact of TLB misses is up to 50% less for the 16kiB GPU configuration compared to the 8kiB GPU configuration. For GPU memory, it is impossible to adjust the page size; however, one can force the memory addresses of all memory blocks to align to improve data locality, as discussed in Subsection 5.2.4. Unfortunately, we could not gain insight into the alternation of RDMA operations originating from various QPs at 90Gbps because the NICs using TCPdump cannot store Ethernet traffic at this data rate.

5.3.4. Conclusion

In this section the number of QPs denotes the total aggregated number of QPs in the measurement, furthermore, four requesters (transmitters) were used and one responder (receiver). If for instance 40 QPs is stated, it implies that there were 40 QPs in the responder and 10 QPs in each requester.

- When using threads per QP the goodput becomes very poor once 40 QPs or more are used. For four QPs, a stable goodput of 90Gbps is measured, which is the theoretical maximum due to the used rate limit. In addition, the goodput for 40 QPs is unstable between 90 and 70Gbps and for 400 and 2000 QPs the goodput is relatively equal and stable at 72Gbps.

- Utilising a single SRQ thread, up to 2000 QPs can be processed at 90Gbps with CPU utilisation of up to 80% and 43% for message sizes of 8kiB and 16kiB, respectively. In addition, an increase in the goodput deviation was identified as the number of QPs increased.

- Unlike the earlier GPU measurements in the one-to-one setup, the GPU configurations in the many-to-one setup do perform equivalent to the SRQ configurations with main memory. The 8kiB with 2000 QPs is the only deviating configuration with a goodput of 80Gbps. We expect this to be caused by the increase in random writes to GPU memory (reduced memory coalescing) at a high message rate which also increases TLB and/or cache misses. For 16kiB, we expect this effect to be less significant (and not

observable) because the message rate is twice as low, and thus the alternation between memory blocks is twice as low.

The many-to-one setup shows that the SRQ configurations, with and without GPU memory, can achieve a throughput of 90Gbps for 2000 QPs with 16kiB message size. The goodput for these configurations was limited by the rate limit as no congestion or backpressure was noticed. With these results, we confirm that RoCE satisfies requirement R3 and R4 for a many-to-one setup.

5.4. One-to-many setup

In this setup, the most powerful node (node505) is used as a requester and four other nodes as responders. The most powerful node is used as the requester because, in previous measurements, it was found that the requester is more heavily loaded than the responder. Furthermore, SRQs are used in each receiver to keep utilisation and resource requirements low. Due to system limitations, GPU memory is not used during these tests because direct access to GPU memory is only possible on node505. This means we would read the data from the GPU and place it in standard DRAM main memory in the receiving nodes. This has no relevance to the use cases and was therefore not tested. All other settings (except the topology) remained unchanged from the previous one-to-many tests. The settings per configuration are similar to previous SRQ and GPU with SRQ tests and are shown in Table 5.8.

5.4.1. Shared receive queue

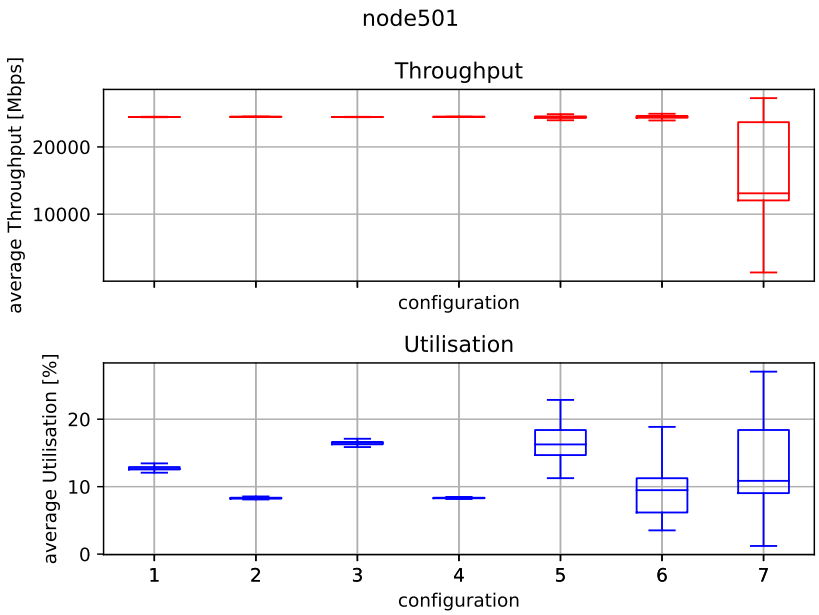


Figure 5.14: Results of one-to-many SRQ at responder node (configurations provided in Table 5.10

Table 5.10: Configurations during queue pair scaling test using one-to-many setup (used in Figures 5.14 and 5.15)

configuration	1	2	3	4	5	6	7	8
Size [kiB]	8	16	8	16	8	16	8	16
QPs	4		40		400		2000	

The result of one of the four responder nodes is shown in Figure 5.14. The goodput is constant at 24.4Gbps for configurations up to and including 400 QPs. There is no difference in goodput for these configurations, so 8kiB and 16kiB perform equally well. On the other hand, notice the increase in CPU utilisation distribution as the number of QPs increases. In addition, as previous results showed, CPU utilisation is higher for 8kiB configurations compared to 16kiB configurations.

Note that in Figure 5.14, configuration 8 is missing. This is because there are problems with these measurements and with the measurements of configuration 7. For these two configurations, timeout events were recorded for different responder nodes. This implies that for 10 seconds no data was received after the SRQ thread was started. No network congestion was observed during any of the measurements. This indicates that the responder nodes had no problems processing the amount of data because any issues are noticeable by message drops or backpressure in the network that (eventually) require congestion management. The main reason for these timeouts seems to be a design flaw. Firstly, the test application uses a delay mechanism for requester nodes to ensure responders are ready to receive data. Though, in this setup, the requester takes longer to initialise and modify QPs and to fill the queues as it has to handle up to 4 times as many QPs as the responders. This may have caused too much delay for some clients, triggering the timeout. Secondly, the number of individual threads and compute load could be a problem. The troublesome configurations have five times more QPs than have been tested in a requester. Moreover, in the previous measurements, it had been noticed that the CPU load in the requester was always higher than in the responder. The higher workload per thread and the huge number of threads can cause scheduling problems and increase cache misses.

This can be improved by handling multiple QPs in a single thread. For example, running 50 threads that each handle 40 QPs reduces the number of context switches. This makes it possible for multiple QPs to use the same completion queue, reducing the number of resources and notification channels. Besides, there was no focus on the number of cache misses and data locality during development. Providing only the strictly necessary information to a thread and storing it more efficiently in smaller structures can reduce the penalty for context switching (and cache misses). But to reduce CPU utilisation even more drastically in the requester, signalled events should be used. This reduces the number of work completions, thus decreasing CPU utilisation because not every send request is checked for correct transmission. However, if something goes wrong when processing a work request, it will always result in a work completion with associated error codes. So signalled events lower the insight into time points when send work requests are completed but retain insight into errors.

Also note from Figures 5.14 and 5.15 that the goodput in the responder varies more at 400 QPs than at 4 QPs and 40 QPs while this increase in variation is not visible in the requester. Hence, the requester can keep the aggregated throughput relatively stable, but the stability of the throughput distribution per QP decreases as the number of QPs increases.

5.4.2. Conclusion

This setup proved that it is possible to achieve a goodput of 97.6Gbps when using up to 400 QPs in a one-to-four topology. With the achieved data rate, requirement R4 (in which a minimum goodput of 90Gbps is specified) is amply fulfilled. In addition, this latest setup proved that the protocol also supports a 1-to-N topology. So it is affirmed that the protocol meets requirement R3, which requires the data transport method to support different network topologies such as 1-to-1, 1-to-N and N-to-1. Nevertheless, performance was not as satisfactory when using 2000 QPs. Three implementation modifications were identified to improve performance, firstly implementing signalled events to reduce the processing load per QP. Secondly, reducing the number of threads by processing more QPs in a single thread and optimising the data structures used to reduce the context switch and cache miss penalty.

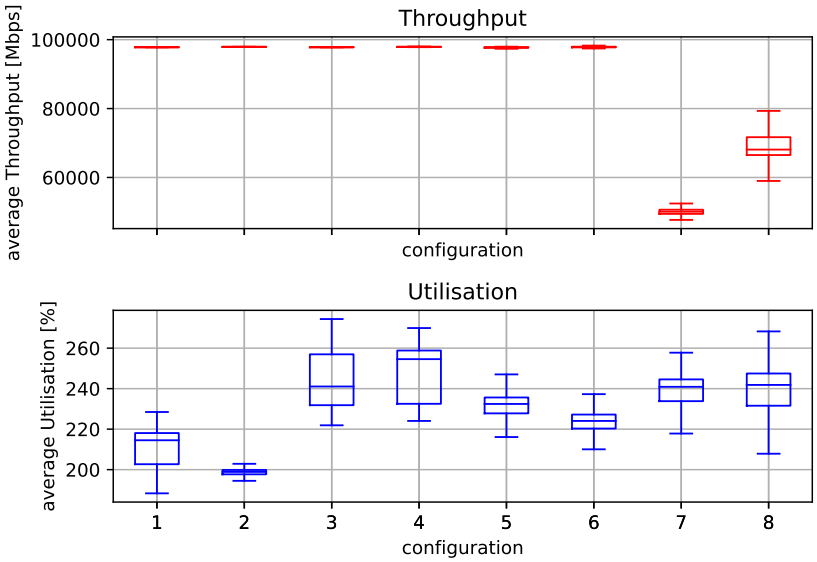


Figure 5.15: Results of one-to-many SRQ at requester node (configurations provided in Table 5.10

6

FPGA implementation and results

Most radio telescope systems require data transport from FPGAs close to antennae to a central processing location. In this thesis, we investigate the feasibility of RoCE for such systems; therefore, we also examine its implementation on FPGAs and its possible limitations. This chapter discusses our methodology and results to create a proof-of-concept implementation of RoCE on FPGA. The RoCE protocol requires a UDP/IP network stack and a DMA implementation augmented with RoCE. Fully implementing such a complex system is infeasible in the time frame of our project. Hence, we discuss available RoCE FPGA implementations in Section 6.1, whereafter we discuss the selected implementation in greater detail in Section 6.2. Section 6.3 covers the experimental setup and methodology. Lastly, we discuss the results in Section 6.4.

6.1. Related FPGA implementations

This section evaluates the existing FPGA RDMA solutions in order to compare their opportunities and shortcomings. Schelten et al. present a network-attached accelerator (NAA) framework with partial support for RoCEv2 [39, 43]. 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. [33] 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 [50] 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.

Companies such as Xilinx and BittWare also offer RDMA FPGA implementations, respectively ERNIC [20] and GROFV RDMA [8]. Both claim to be compatible with RDMA NICs at 100Gbps. ERNIC is limited to Reliable Connection and up to 127 QPs, while GROFV supports all transport services except DC and is scalable to more than 1023 QPs. Unfortunately, these implementations are a black box model and require significant licensing costs or costs to have the implementation adapted to the targeted implementation.

Korolija et al. [28] implement an open-source RoCEv2 stack operating at 100GbE into their Coyote Framework [44]. 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 [38], and StRoM [41]. The StRoM network stack has been evaluated between two FPGAs by T. Song [42]. 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 RNIC, but has not yet been validated between an FPGA and RNIC.

The SKA concortium, whose RNIC implementation was discussed earlier [5], implemented the RoCE SEND over unreliable connection in VHDL to be used between two FPGAs. Results for this were shared in

internal documentation, however, the implementation was not available to us.

To summarise, the IPs provided by Xilinx and BittWare are the only ones currently claiming to support RoCE communication with RNICs. Other implementations, such as RASHPA and the NAA framework, provide a custom RDMA protocol or redefine an RDMA operation, making it very unlikely to work with network cards. The Coyote framework provides a scalable and open-source available RoCE implementation which seems to require small modifications to support Unreliable Connection since UC requires a subset of the functionalities of RC. Moreover, Coyote is an actively maintained framework and a continuing development of the (no longer actively maintained) Limago and StRom implementations. Given these points, we decided to use the Coyote framework as a starting point for our implementation and tests.

6.2. Coyote

Coyote is a framework that offers a configurable "shell" for FPGAs supporting various OS abstractions (such as processes and virtual memory). Coyote provides three essential components; first, it provides a reconfigurable FPGA implementation for various Xilinx FPGAs. Secondly, a high-level C++ Coyote API provides a runtime manager and an application interface to control the user-defined logic in the FPGA. Lastly, Coyote contains a custom low-level Linux kernel driver responsible for communication with the FPGA and memory mappings.

The FPGA implementation can be divided into two parts: the static and dynamic regions. A high-level overview of Coyote's FPGA structure is shown in Figure 6.1. The dynamic region consists of vFPGAs, which can be reconfigured at runtime. These vFPGAs contain user-defined logic within a static wrapper. The wrapper provides an interface to the rest of the system, the static region. The static region includes logic to enable partial reconfiguration and communication with the CPU's OS and optional logic such as memory and network stack. The memory stack contains logic to use the HBM or DRAM memory located at the FPGA and the host's CPU memory through XDMA. While network services such as TCP/IP and RoCEv2 are implemented in the network stack.

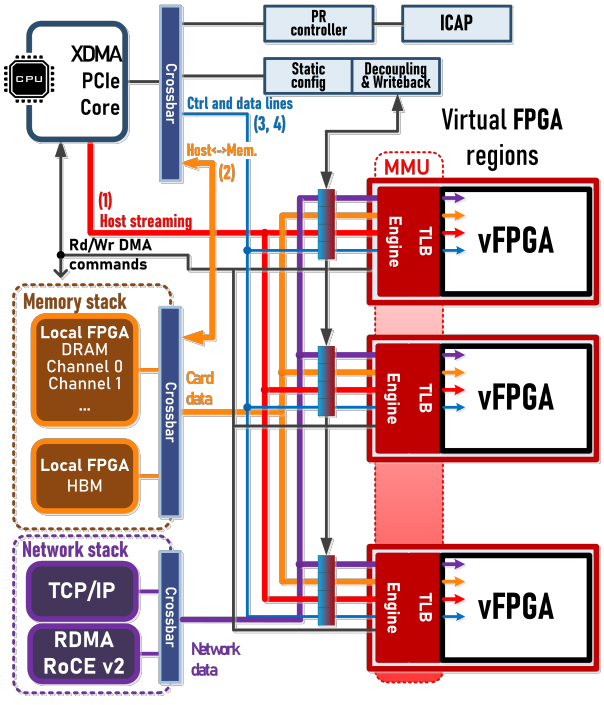


Figure 6.1: Coyote high-level overview (Fig. 1, [28])

The network stack is based upon the open-source work of Sidler [40, 12]. This work supports TCP/IP, RoCEv2 and UDP/IP at 10-100Gbit/s. The network stack is mainly developed in HLS, improving readability and ease of adjustment compared to HDL programming. We are mainly concerned with the RoCE stack, shown in Figure 6.2, which is built upon the UDP/IP implementation. The implementation supports the Reliable Connection transport service and three operations; SEND, RDMA WRITE and RDMA READ. Each

protocol header is processed in several stages and, if successful, removed after which the remainder is passed on to the next stage. The overview shows a clear separation between the data path at the edges and the state-keeping structures in the middle. The CPU application and user-defined logic in the vFPGAs can fill these tables and queues. The sizes of these tables and queues can be adjusted at compile time though it already supports 500 QPs by default.

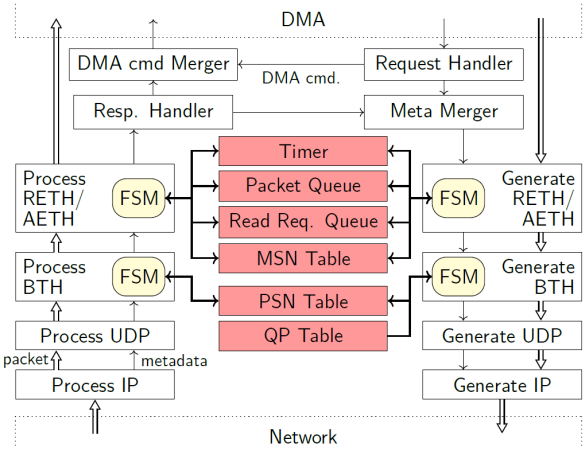


Figure 6.2: RoCEv2 FPGA architecture overview (Fig 5.4, [40])

6.3. Methodology

6.3.1. Experimental setup

We use two clusters to validate the RoCE stack of Coyote, namely the ETH Zurich hybrid accelerated compute cluster (HACC) and the DAS6 cluster at ASTRON. The AMD and Xilinx university program provides five compute clusters (HACCs) across the world.[1] [2] An overview of the available hardware in ETHz HACC is provided in Section C.1. In our test, we used two alveo U55C nodes as these nodes were provided with an easy-to-use hotplug boot script. This script eliminates a warm reboot of the system to discover the new PCIe memory mapping of Coyote. Besides, the FPGA on the U55C is the same as for the U280 which we have available in DAS6. The main difference between them is the availability of DRAM and HBM memory. The available hardware in the DAS6 cluster is covered in Appendix 3.1. In this cluster, we used node505 as it is equipped with the U280, and another standard node is used to receive and inspect the RoCE Ethernet packets. The framework is built using Xilinx 2022.1 tooling and python for some automation. The Coyote API and test program are compiled using C++17.

6.3.2. FPGA - FPGA validation

The FPGA RoCE network stack will be validated with two FPGAs in HACC leveraging host CPU memory. The RoCE FPGA network stack provides the message size as a configuration parameter. So the stack does not provide features such as SRQ and linked requests. These tests will be performed over a reliable connection using WRITE operations since the FPGA stack does not support UC service and write with immediate operations. We vary the message size between 128B and 2GB with increments of factor 2. We use the *rdma_perf* example application provided in the Coyote repository as a basis for these tests. The throughput and latency measurements are performed with CPU counters. The timing measurements, therefore, also include PCIe, driver and interrupt latencies. As a result, measurements may be more negative than reality. However, this is unimportant because the primary concern is to validate the protocol between FPGAs and, later, between an FPGA and network card. The *rdma_perf* application has been extended to initialise and print the memory content before and after data transport. This allows us to verify that the transport correctly took place.

[1] `https://www.xilinx.com/support/university.html`
[2] `https://www.amd-haccs.io/`

6.3.3. FPGA - RNIC validation

The second phase uses the DAS6 cluster to test the RoCE FPGA implementation between the available U280 and a single Mellanox NIC. During this setup, we use *tcpdump* to analyse the ethernet packets created by the FPGA. These tests will again be performed over a reliable connection using WRITE operations. The application controlling the FPGA is based on the *rdma_perf* application as used in the previous setup. It has been modified so that we can manually transfer the RDMA values between the two remote RoCE nodes. In addition, we designed a new, less complicated CPU application based on the implementation discussed in Chapter 4. The new CPU application leverages the same *RDMA API*. If the RoCE packets are generated correctly as required by the protocol and validated by the network card, then we will test this for different message sizes.

6.4. Results

6.4.1. FPGA - FPGA validation

The FPGA - FPGA setup has been validated using two U55C nodes. The first problem we encountered during the tests of the RoCE stack inside the Coyote framework was the inability to send messages larger than 1024B. Table 6.1 shows the test result with message sizes ranging from 128B up to 1024B. Each message size is sent 1000 times, after which the average is taken. The results show a stable latency of 6 microseconds and a maximum throughput of 20Gbps with a message size of 1024 Bytes. The throughput increases linearly with the message size, indicating a bottleneck in the message rate. Besides, the maximum theoretical goodput is lower when the message is lower than the maximum PMTU of 4096B. The terminal output of this test is given in Appendix C.2.

Table 6.1: Coyote RoCE throughput and latency between two U55C FPGAs

Message size [B]	Throughput [Mbps]	Latency [us]
128	2504	5.77
256	5048	5.63
512	10248	5.78
1024	20420	6.02

From conversations with the Coyote developer and HACC administrator, the message size limitation appears to be caused by the network switch between the FPGAs. The switch either split the jumbo frames or did not forward them. Via TCPdump, we found that the "do not fragment flag" was not set in Coyote packets; see Section C.3 for the TCPdumps. However, activating this flag had no positive result. The message size limitation is acceptable in the current development phase because we aim for a proof-of-concept implementation between an FPGA and RNIC.

In conclusion, the Coyote RoCE networking stack is proven to work between two FPGAs with a maximum message size of 1024B. In addition, measurement results indicate that goodput will increase if larger packets and message sizes can be sent.

6.4.2. FPGA - RNIC validation

This section discusses the results of the tests between the Coyote framework and the Mellanox connectX-6 inside the DAS6 cluster. The test between the FPGA and RNIC revealed various faults, which will be discussed in this section. First of all, we discovered, with the aid of *TCPdump*, an error in the transmitted packet sequence number (PSN). The PSN appears in each RoCE packet in the base transport header; besides, each QP has its own PSN value. The PSN must match the locally stored values and should be incremented for each packet. However, the FPGA stack sends the wrong PSN, namely the PSN of the local QP, instead of the PSN belonging to the remote QP. This problem was not noticed between the two FPGAs because each FPGA compared the received PSN with the incorrect known PSN value.

Manually adjusting the PSN numbers revealed a second problem related to the Invariant CRC (ICRC) check. The ICRC error was reported by the *NIC_Snapshot*, discussed in Subsection 4.3.2. The RoCE protocol adds a RoCE-specific four Bytes ICRC check at the end of each Ethernet frame. Initially, the ICRC calculation on the FPGA appeared to be off by default and contained a default value. Moreover, the ICRC was also not checked in the default configuration and, thus, not in the previous setup between two FPGAs. However, activating the pre-existing ICRC components did not resolve the error notification. Despite trying to trace the specific error, we could not resolve it as it is complicated to check and adjust the ICRC calculation.

Finally, we encountered kernel panic in the DAS6 server in which we were using the FPGA and Coyote. This kernel panic caused the entire server to shut down for security reasons. We expect an illegal memory operation in the Coyote driver caused this issue. However, tracing the cause is very difficult because the server immediately freezes and shuts down, making log data recovery difficult and time consuming.

To summarise, we identified several problems in the FPGA network stack that prevented it from working with an RNIC. We were not able to resolve these issues with the framework within the scope of this project. However, we were able to identify that it should be possible to use this protocol between an FPGA and RNIC. The vast majority of the RoCEv2 protocol is already well implemented in the framework, which was confirmed by checking the Ethernet packets via an Ethernet packet analyser.

7

Conclusions and future work

The demand for more efficient data transport is ever-increasing and also applies to radio astronomical systems to be able to observe the universe in greater detail. In this work, we have studied three sub-research questions to answer our main research question;
Can we use RoCE to reduce system load and increase throughput in sending and receiving antenna data for applications in radio-telescopes?
The three sub-research questions and the way they are addressed is listed as follows:

1. What are the characteristics and requirements of applications for radio-telescopes?
   We investigated data transport in radio astronomy systems using three use cases and drafted relevant requirements for these. More details about this investigation are discussed in Section 7.1.1.

2. Which configuration of RoCE is best for the intended application?
   We examined the RoCE protocol and its numerous settings, then compared the properties of fundamental settings with the radio astronomical data transport conditions to conclude whether the protocol is viable for this application domain. More details about these comparisons are discussed in Section 7.1.1 and 7.1.4.

3. Can RoCE deliver the required performance and reliability at the scale of large distributed telescope systems?
   We developed a RoCE test application between RoCE-enabled NICs to profile, analyse and compare several RoCE settings for their impact on the throughput and CPU utilisation. We also worked to investigate the feasibility and performance of a RoCE implementation on FPGAs. The answer to the research questions is: yes RoCE can deliver the required performance under specific conditions. More details about these results are discussed in Section 7.1.4 and 7.1.5.

Section 7.1 provides a consistent summary of Chapters 2 to 6, after which this chapter concludes with an overview of recommendations for future work in Section 7.2.

7.1. Conclusions

7.1.1. Chapter 2

The chapter began with an introduction to the processing pipeline of radio telescope systems and three radio astronomy use cases. Each use-case has distinctive characteristics; The first has many transmitters distributed across Europe with relatively low data rates per transmitter. The second has higher data rates over a private network. In comparison, the last one has exceptionally high data rates from relatively few transmitters. These use cases formulated requirements for a data transport technology between the antennas and the central computer, by which we addressed sub-research question 1.

The chapter proceeded with a brief overview of several RDMA technologies, after which we discussed the RoCE protocol in greater detail. We discussed the notion of converged Ethernet and highlighted existing and new approaches to dealing with (RoCE) congestion. Using multiple data streams and protocols over a single Ethernet network can cause congestion and unfair bandwidth distribution. Therefore, it is common practice

to transport (less) important data streams over a network via dedicated buffers and individual congestion management methods. An Ethernet network where this division of data streams over available resources is implemented is called a converged Ethernet network. This investigation also indicated that RoCE is a mature and widely used technology that will scale to higher speeds in the future, as desired by requirement S3 and S2.

After that, we explained RoCE core components and settings. In addition, we also discussed the opportunities of these settings to achieve better speeds or lower CPU utilisation. Based on the study of the operation and transport capabilities of RoCE and the characteristics of the use cases, it has been made evident that the unreliable connection transport service with the WRITE with immediate data operation is the best fit for the application in radio telescope systems. This ensures there is no obligation for retransmission and the immediate data allows us to flag memory regions with which we meet requirement R5 and R6, respectively, based on the technology choice.

7.1.2. Chapter 3

Chapter 3 discussed the available hardware and software and described the method to analyse the RoCE protocol between RNICs. The key performance indices were defined as the goodput and CPU utilisation during data transfer. We discussed different ways to measure these, after which it was decided to measure the CPU utilisation at thread level and determine the goodput via the number of work completions. In addition, RNIC counters are used to detect network congestion.

The second part of the chapter presented four different setups to investigate the performance of the RoCE protocol and the various settings. First, we define a setup to demonstrate the differences in throughput and CPU utilisation between data transport over UDP and RoCE.

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

The third setup uses a many-to-one topology to study the scalability of RoCE. The aim is to simulate radio telescopes' distributed and scalable aspects using the available cluster.

The fourth and last setup is the one-to-many topology to analyse the scalability of a sender.

7.1.3. Chapter 4

In Chapter 4, we provided an overview of the system we developed to evaluate the design space of RoCE in accordance with the methods and metrics discussed in Chapter 3. The developed program can be used for one-to-one connectivity as well as many-to-one and one-to-many. Besides, our implementation facilitates the registration of GPU memory to RoCE so that the RNIC can read from and write into GPU memory without the involvement of the main memory or CPU. This system was implemented with COTS products, and when feasible, open-source software was used, satisfying non-functional requirements S1 and S4.

Furthermore, we address our application's three major components: the RDMA API, profiling tools and the main program. The main program uses the RDMA API and profiling tools to establish and measure data transport via RoCE.

The RDMA API uses ibVerbs to query, create and modify all relevant RoCE attributes. The API also provides transport methods to the main program, which posts RoCE work requests in a separate thread to the RNIC 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 RNIC counters and measure and track function calls. Hence, the implementation includes a few abstractions which satisfy non-functional requirement S5.

7.1.4. Chapter 5

In Chapter 5, we examined the results of the different configurations and setups. First, we briefly demonstrated the performance difference between UDP and RoCE as transport methods. Here, the UDP protocol had a maximum of 50Gbps with a CPU utilisation of 146%, while RoCE could achieve 97Gbps with 43% utilisation using equal message sizes. RoCE thus yielded almost twice as much throughput for a third of the CPU utilisation and therefore 6 times better performance.

In the second setup, we studied the performance of several RoCE settings in a one-to-one topology. 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 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, as was expected, 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 third setup using the many-to-one topology has proven that a configuration using an SRQ with up to a total of 2000 QPs in the responder and 16kiB messages, regardless of GPU memory usage, can transport a goodput of 90Gbps from four requesters to one responder. However, when using one thread per QP, many congestion problems were recorded and the goodput did not stably exceed 73Gbps when using 400 or 2000 QPs. The last setup demonstrated that our application could use RoCE over a one-to-many topology. It 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 RNIC.

Regarding the second and third sub-research questions, the transport service and operation determined in Chapter 2 achieve optimal performance using an SRQ and the solicited events mechanism. In this case, we use 2000 QPs, solicited events interval of 100 and receive and completion queue of 20000 elements. This achieved a CPU utilisation of 40% on a single core and a goodput of 90Gbps (which is the maximum achievable due to RoCE Ethernet frame efficiency of 98% and an aggregated rate limit of 92Gbps). Yet the optimal configuration depends on the use case, with the number of QPs and message size being important metrics. For example, we showed the benefit of an increase in linked WR and solicited events stagnates as the message size increased. With our application and setups, we showed that our RoCE test system meets the performance (R4) and network (R2, R3) requirements without retransmission (R5). Reliability was validated in a research cluster over short periods while using PFC and considered satisfactory. However, our test set-up lacks the scale of a radio telescope system. In addition, we have not tested over public Ethernet; we are confident that it can be accomplished as we use protocols supported by public Ethernet. Given the resources at our disposal and the results achieved, we argue that RoCE should be able to satisfy this application area.

7.1.5. Chapter 6

In Chapter 6, we started by exploring available FPGA implementations of RoCE. From this, the Coyote framework was selected and adapted to evaluate data transport from an FPGA to RNIC 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.

Due to network limitations, it was not possible to use Ethernet frames larger than 1500B. As a result, the largest possible message size was 1024B, for which it achieved a goodput of 20.4Gbps between two FPGAs. Yet it is possible to achieve a data rate of more than 80Gbps if the maximum packet size of 4096B can be used. This is because the buffers can scale along by a factor of 4 without adversely affecting the frequency at which packets are sent, allowing throughput to scale proportionally.

Follow-up testing between an RNIC and FPGA revealed some other areas of improvement. First, an incorrect packet sequence number was found to be sent and compared in the FPGA implementation. Secondly, the ICRC calculation did not meet the strict conditions set by the RoCE specification, which could not be solved within the scope of this project. Consequently, we have not been able to receive the data transmitted by an FPGA on an RNIC; hence, requirement R1 has not been confirmed with a real-world test setup. Nonetheless, this study showed that it is possible to implement RoCE on FPGAs, and it is evident that the FPGA implementation will perform significantly better if larger Ethernet packets can be transported.

7.2. Future work

Two main areas for future work are identified from our study. First, we observed many fluctuations in data rates, which could not be sufficiently managed by rate limiting via ETS. Besides millisecond-level fluctuations, microbursts have also been an issue, leading us to use Priority Flow Control (PFC) to improve this. In addition, when using the RNICs as transmitters, as in our set-up with the DAS6 system, it is unclear in which order packets from a QP are sent if multiple QPs are used, partly because TCPdump cannot handle the high data rate. This can be addressed using an FPGA as a transmitter (requester) because it can be configured to send packets without bursts at a stable rate. This allows better investigation of sustained throughput without using PFC or other congestion methods. Furthermore, sending an exact sequence of packets via an FPGA is possible, allowing it to test interleaved QP packets.

Second, more research on GPU memory in combination with RoCE as a transport technique is recommended. Our study showed systematic short-lived reductions in goodput when a goodput rate of 98Gbps was achieved, for which the cause is still unknown. In addition, the data in the GPU memory was not used for further processing by the GPU during the measurements, which might impact the performance. Implementing a GPU kernel and additional research into possible bottlenecks and methods to quantify them may uncover and resolve these issues.

In addition to the aforementioned items for future work, the following list of issues is also considered worth investigating:

1. Analyse IP services such as VLAN and AES. In the real world, VLANs are often used to separate networks and data flows from each other, e.g. to increase security. In addition, in some use cases, the data travels over public Ethernet, for which additional encryption may be required, for instance, via AES-XTS. The use of these methods causes additional loads on the network card and can therefore affect performance. The use of VLANs and encryption methods is not included in our study and is recommended if the protocol's security is to be improved.

2. Enhance the SRQ implementation. The SRQ provided improvements in both CPU utilisation and goodput. However, CPU utilisation increased drastically with more QPs to a point where it no longer offered an advantage over QP handling in separate threads. The use of the SRQ in our implementation is limited to 1 thread, limiting the maximum number of QPs. By distributing the total number of QPs over several SRQs (in different threads), the amount of cache per SRQ increases, and the number of QPs per SRQ decreases. Making it easy to find and update the correct QP data.

3. Implementation and validation improvements for RoCE on an FPGA. First, it is worth investigating what causes the problem in the Coyote RoCE network stack when using packets with a payload of 4096B. Due to this obstacle, achieving the desired goodput rate is impossible. In addition, the use cases ultimately require the RoCE stack to be used without a CPU activating the operations. This can be tested by extracting the RoCE network stack from the Coyote framework and using it in its own implementation. This way, driver and kernel panics can also be addressed. Lastly, the functioning of the ICRC calculation should be checked and improved so that an RNIC can further validate the RoCE package.

4. Identify and measure non-network related bottlenecks. Currently, we notice backpressure in the receiving RNIC. However, we cannot distinguish whether this is due to computing load in the RNIC, cache or TLB misses in the RNIC, a bottleneck in memory or a bottleneck in the PCI express bus. Identifying and quantifying these performance degradation sources makes it possible to mitigate them through other settings. It is also possible that a server's architectural (dis)advantages emerge, which can be considered when designing new systems.

A

DAS6 overview

A.1. Hardware overview

Table A.1: Overview of available hardware in DAS6 at ASTRON

Machine	Component	Value
node501-504	CPU	dual AMD EPYC 7282 16-Core Processor (min 1.5GHz, max 2.8GHz)
	RAM	135GiB per NUMA domain
	OS	Rocky Linux 8.6 (Green Obsidian) 5.18.14-1.el8.elrepo.x86_64
	NIC	Mellanox Technologies MT28908 Family [ConnectX-6]
	GPU	2x NVIDIA A4000
node505	CPU	dual AMD EPYC 7H12 64-Core Processor (min 1.5GHz, max 2.6GHz)
	RAM	1TB of 64GB SK HYNIX HMAA8GR7AJR4N-XN, 3200 MT/s
	OS	Rocky Linux 8.5 (Green Obsidian) 5.4.167-1.el8.elrepo.x86_64
	NIC	Mellanox Technologies MT28908 Family [ConnectX-6], NUMA node 0
	GPU	NVIDIA Corporation GA100 [A100 PCIe 40GB], NUMA node 0
	FPGA	Xilinx U280 2x100GbE connected, NUMA node 1
headnode	CPU	dual AMD EPYC 7F72 24-Core Processor (min 2.5GHz, max 3.2GHz)
	RAM	512GB
	OS	Rocky Linux 8.6 (Green Obsidian) 5.4.195-1.el8.elrepo.x86_64
	NIC	Mellanox MT28908 Family [ConnectX-6 Lx]
	GPU	none
switch	Type	SN2100, 16MB shared buffer memory
	OS	Onyx 3.8.2008
	Settings	TC3: lossless PFC ECN minimum-absolute 300 ECN maximum-absolute 3000

A.2. Software overview

Table A.2: Software used for the CPU setup

Whireshark 3.6.7
Tcpdump with libpcap 1.8
python 3.9.9
rdma-core 39.1
mellanox ofed 5.4
Nvidia driver version 510 with peerdirect
cuda 11.6.2
cmake 3.22.1, GCC 9.4.0, C++17
Xilinx 2022.1 tools

B

Additional results

B.1. One-to-one configuration

All RNICs and the network switch have separate buffers for Traffic Class 3, for which PFC is enabled. The RNIC is set to DSCP and has a separate ingress buffer for TC 3. No bandwidth limiting is used in the one-to-one setup. The Mellanox switch uses shared buffer pools for which the egress buffer has an infinite alpha, and the ingress buffer has an alpha of 1. These pools are set as lossless pools and have an ECN minimum threshold of 300kB and an ECN maximum threshold of 3MB. The MTU is set to 9000 in the RNICs and switch, and RoCE uses the maximum Path MTU of 4096B.

Table B.1: RoCE settings one-to-one setup

Send queue size	1000
Receive queue size	1000
Completion queue size	1000
Shared receive queue size	2000
Shared completion queue size	2000
memory regions per QP	10
Linked work requests	10

B.2. One-to-one additional results

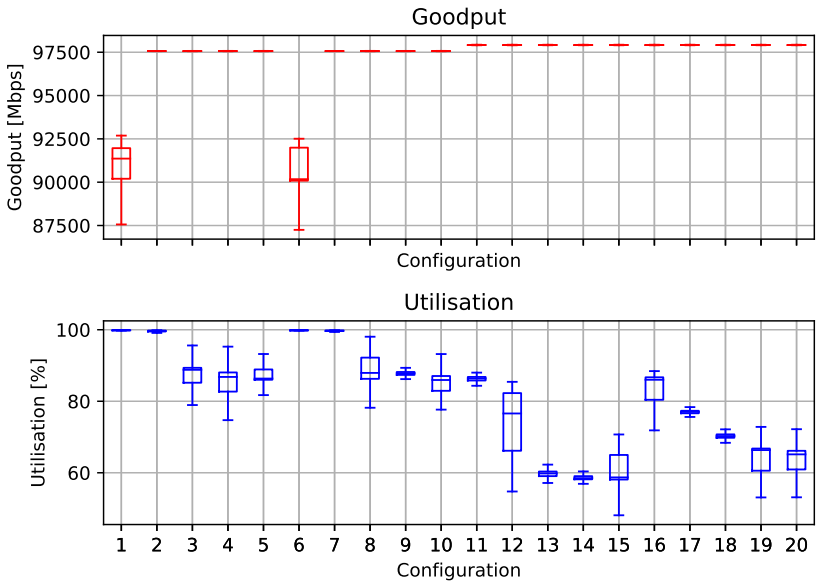


Figure B.1: Results of one-to-one linked WR and SE test at requester node (configurations provided in Table 5.3)

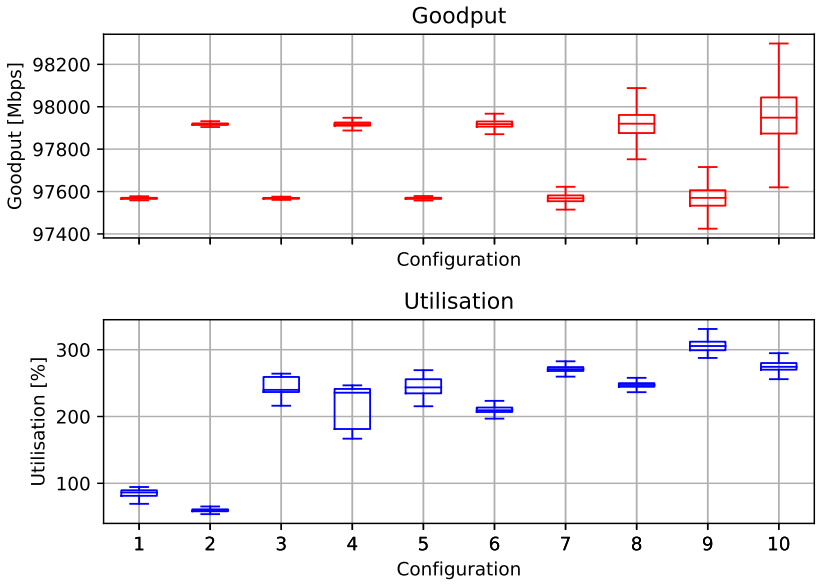


Figure B.2: Results of one-to-one QP scalability test at requester node without SRQ (configurations provided in Table 5.4)

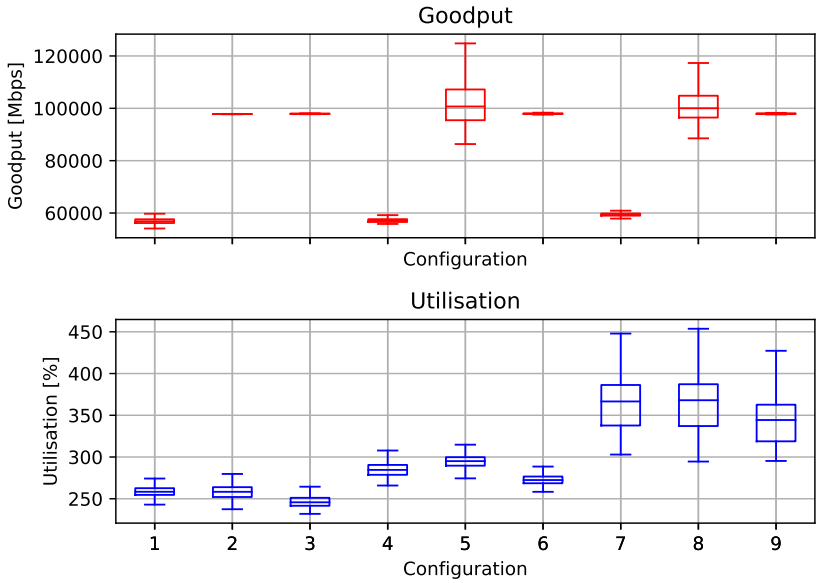


Figure B.3: Results of one-to-one shared receive queue test at requester node (configurations provided in Table 5.5

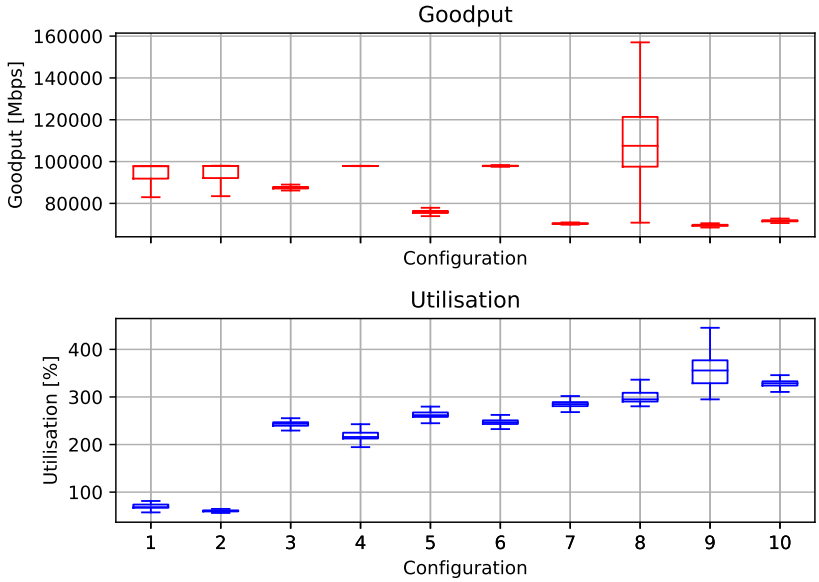
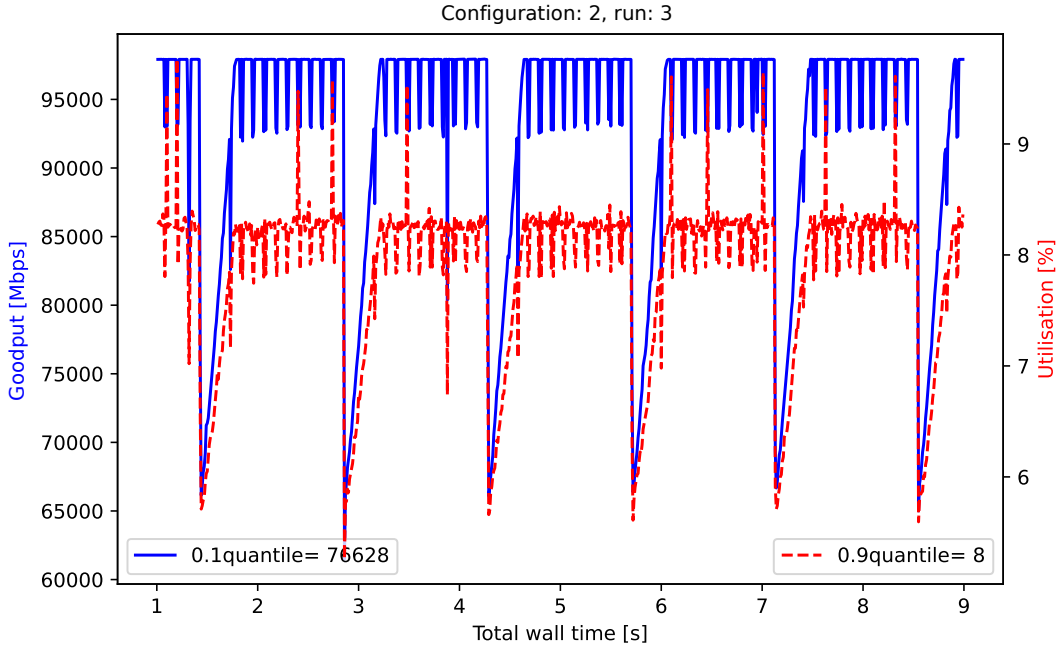
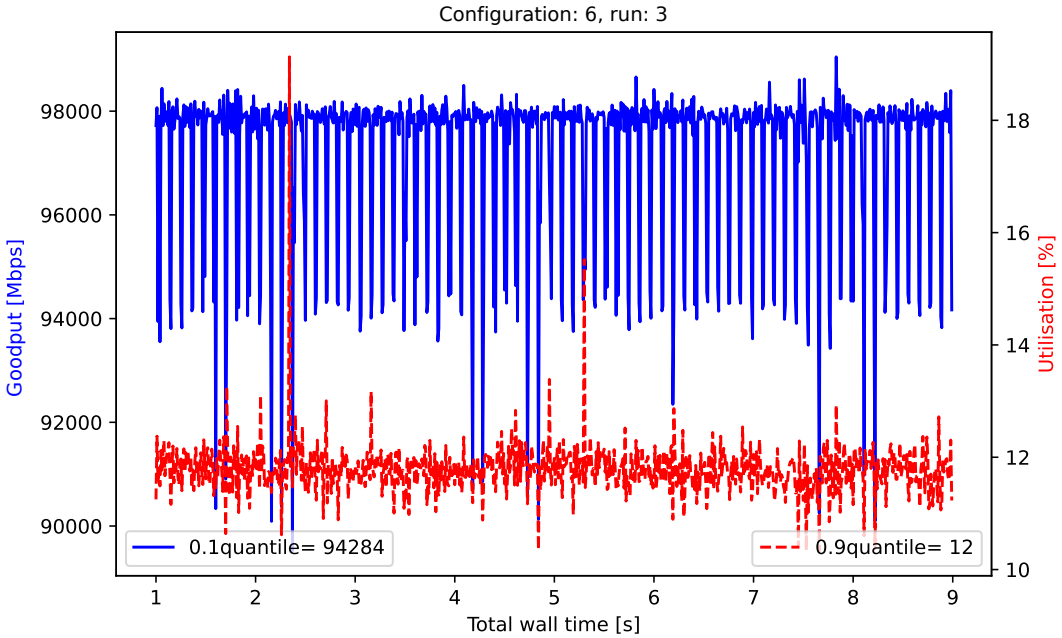


Figure B.4: Results of one-to-one GPU tests at responder node (configurations provided in Table 5.7)



(a) GPU throughput and utilisation using 1QP and message size of 16kiB



(b) GPU throughput and utilisation using 100QP and message size of 16kiB.

Figure B.5: Time series plots of throughput and utilisation in one-to-one setup using GPU memory

B.3. Many-to-one configuration

The modified network settings compared to those discussed in Section B.1 are provided in this section. This setup does use bandwidth limitation through ETS, the RNICs of the 4 requesters are limited to an egress rate of 23Gbps for TC 3 and the egress scheduling is kept at the default value, i.e. vendor dependent. In addition, the ECN minimum and ECN maximum thresholds have been increased to 4MB and 6MB, respectively.

Table B.2: RoCE settings many-to-one setup

Send queue size	1000
Receive queue size	1000
Completion queue size	1000
Shared receive queue size	20000
Shared completion queue size	20000
memory regions per QP	10
Linked work requests	100

B.4. Many-to-one additional results

The results of one requester are given in this appendix because the others show equivalent results.

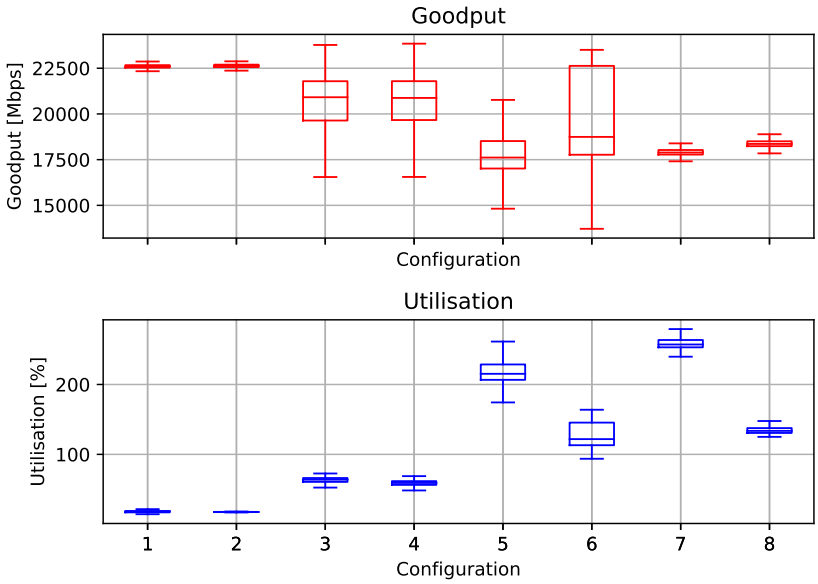


Figure B.6: Results of many-to-one scalability at requester node501 (configurations provided in Table 5.8)

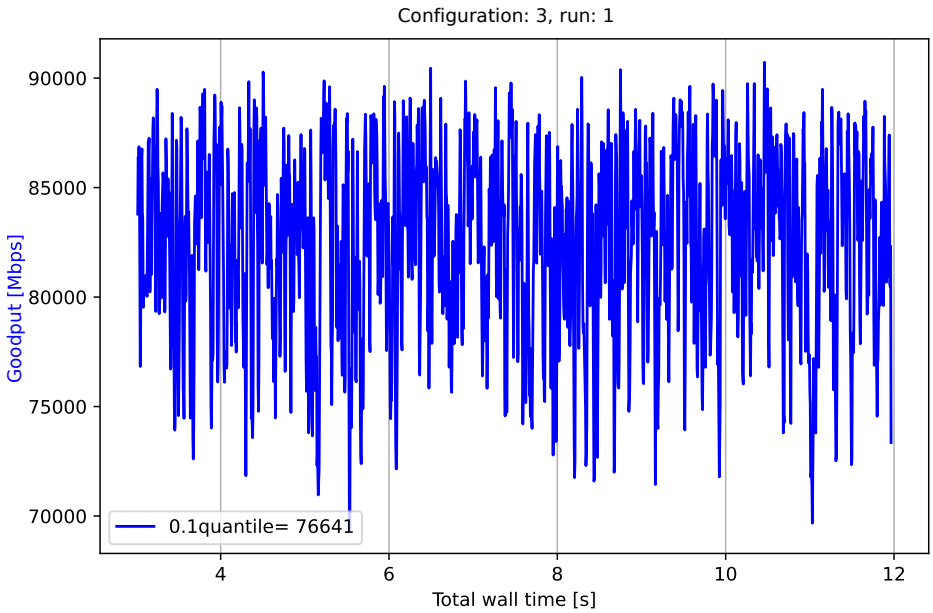


Figure B.7: Time series plots of responder throughput in many-to-one setup configuration 3 (given in Table 5.8)

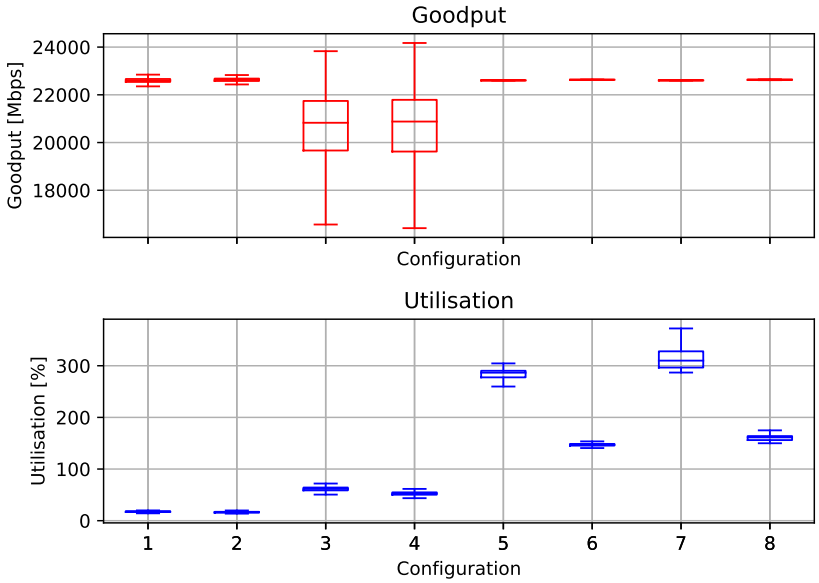


Figure B.8: Results of many-to-one SRQ at requester node501 (configurations provided in Table 5.8)

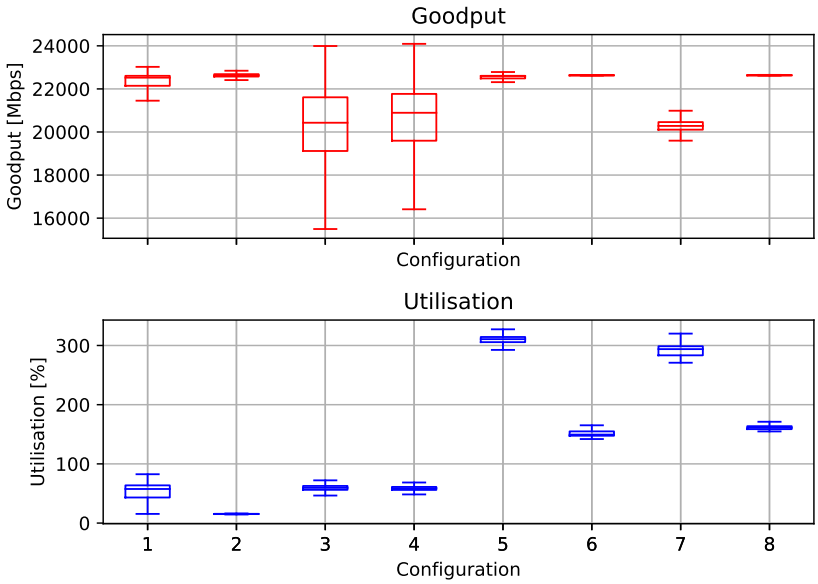


Figure B.9: Results of many-to-one GPU SRQ at requester node501 (configurations provided in Table 5.9)

B.5. One-to-many configuration

The network configuration is similar to that used for the many-to-one setup, discussed in Appendix B.3. This is because the bandwidth limitation via ETS has no effect on incoming traffic, and previously there was no rate limit set on node505. This was a responder in the previous setup and is used as a requester in this setup.

Table B.3: RoCE settings one-to-many setup

Send queue size	1000
Receive queue size	1000
Completion queue size	1000
Shared receive queue size	20000
Shared completion queue size	20000
memory regions per QP	10
Linked work requests	10

C

Coyote FPGA implementation and results

C.1. ETHZ-HACC infrastructure

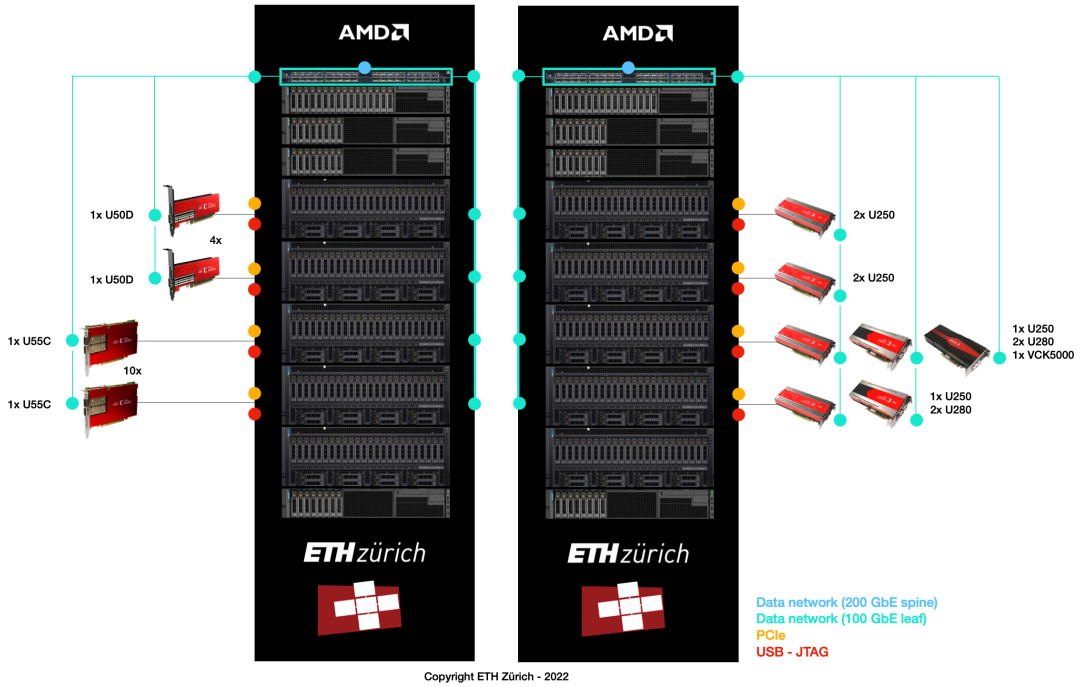


Figure C.1: ETH Zurich HACC infrastructure overview [45]

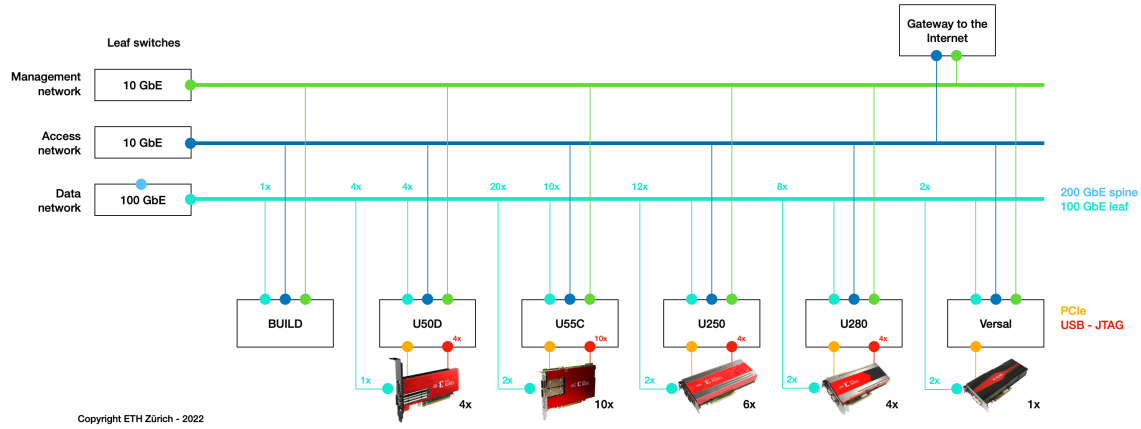


Figure C.2: ETH Zurich HACC networking overview [45]

C.2. Coyote RoCE test terminal output

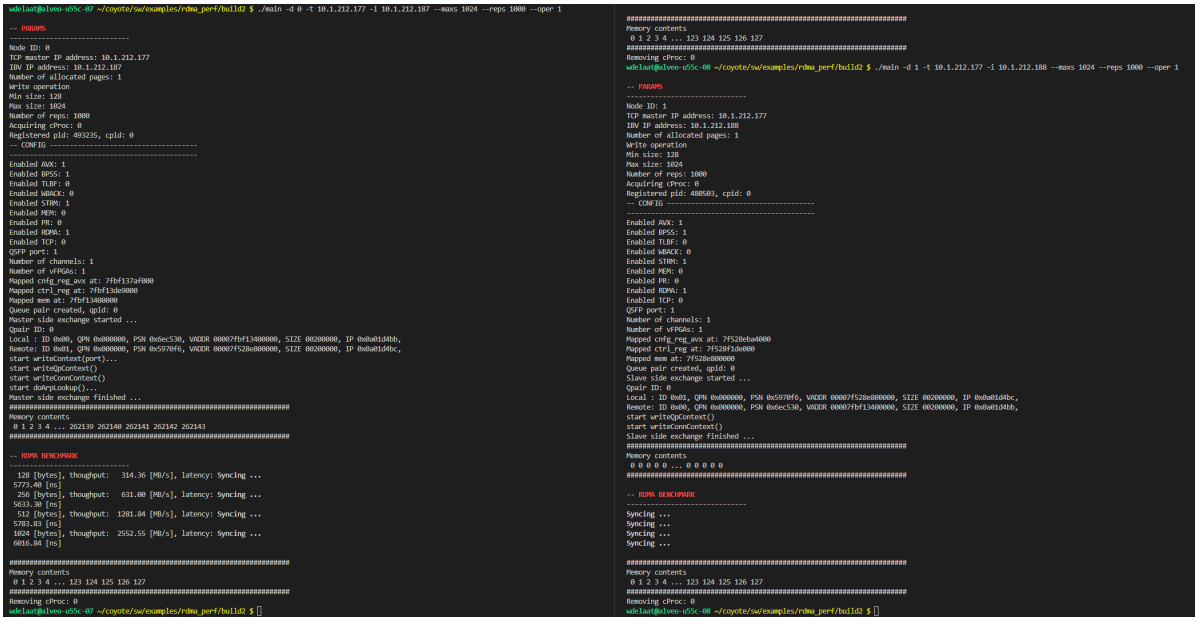


Figure C.3: Terminal output of ETHz HACC U55C RDMA WRITE result

C.3. TCPdump of Coyote and RNIC

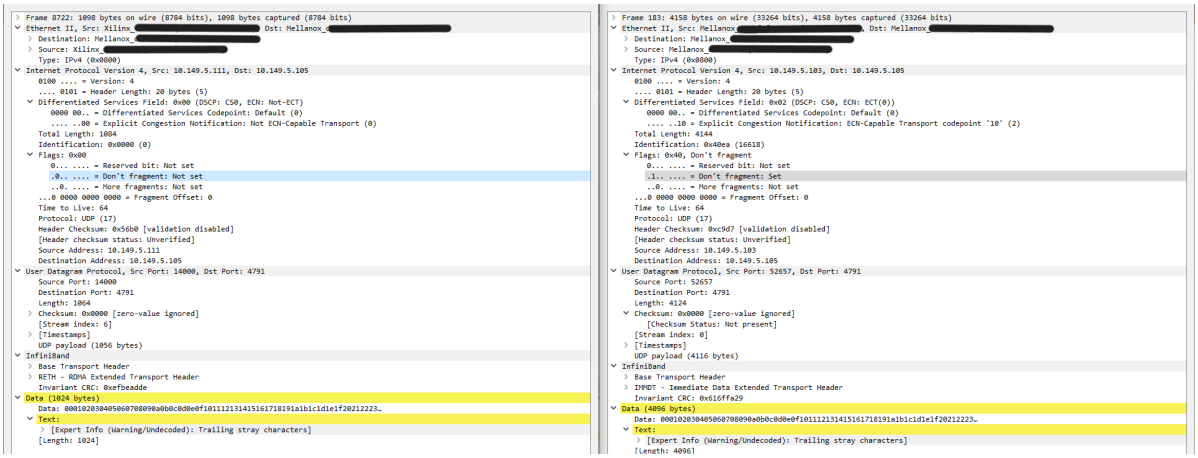


Figure C.4: TCP dump of RDMA WRITE result: FPGA ETHz HACC U55C RoCE packet (left side) and RNIC RoCE packet (right side)

Bibliography

[1] Sergio Otarola- ALMA (ESO/NAOJ/NRAO. *In the Chajnantor Plateau, amazing picture of the antennas under the Milkyway*. URL: https://www.almaobservatory.org/wp-content/uploads/2021/08/50538953736_ee641c7c9b_o.jpg.

[2] I. Ozery A. Barnea and B. Claman. *Scaling Zero Touch RoCE Technology with Round Trip Time Congestion Control*. URL: https://developer.nvidia.com/blog/scaling-zero-touch-roce-technology-with-round-trip-time-congestion-control/.

[3] OpenFabrics Alliance. *OFED for Linux*. URL: https://www.openfabrics.org/ofed-for-linux/.

[4] Vrije Universiteit Amsterdam. *DAS-6 clusters*. URL: https://www.cs.vu.nl/das/clusters.shtml. (accessed: 01.12.2022).

[5] SKA consortium Andrew Ensor. *RDMA-data-transport*. URL: https://gitlab.com/ska-telescope/rdma-data-transport. (accessed: 01.12.2022).

[6] InfiniBand Trade Association. URL: https://www.infinibandta.org/. (accessed: 01.12.2022).

[7] *Atacama large millimeter/submillimeter array*. URL: https://public.nrao.edu/telescopes/alma/. (accessed: 30.10.2022).

[8] BittWare. *GROVF RDMA IP core*. URL: https://www.bittware.com/ip-solutions/grovf-rdma/.

[9] P. Chris Broekema et al. "Cobalt: A GPU-based correlator and beamformer for LOFAR". In: *Astronomy and Computing* 23 (Apr. 2018), pp. 180–192. ISSN: 2213-1337. DOI: 10.1016/j.ascom.2018.04.006. URL: http://dx.doi.org/10.1016/j.ascom.2018.04.006.

[10] CMS Collaboration. *The Phase-2 Upgrade of the CMS Data Acquisition and High Level Trigger*. Tech. rep. This is the final version of the document, approved by the LHCC. Geneva: CERN, 2021. URL: https://cds.cern.ch/record/2759072.

[11] Alexandros Daglis et al. "Manycore Network Interfaces for in-memory rack-scale computing". In: *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*. 2015, pp. 567–579. DOI: 10.1145/2749469.2750415.

[12] ETH Zurich David Sidler Systems Group. *Scalable Network Stack supporting TCP/IP, RoCEv2, UDP/IP at 10-100Gbit/s*. URL: https://github.com/fpgasystems/fpga-network-stack. (accessed: 10.10.2022).

[13] Rob Fender et al. "The LOFAR Transients Key Project". In: (2006). DOI: 10.48550/ARXIV.ASTRO-PH/0611298. URL: https://arxiv.org/abs/astro-ph/0611298.

[14] Johann George. *qperf*. URL: https://github.com/linux-rdma/qperf. (accessed: 01.12.2022).

[15] Jean-Mathias Grießmeier, Philippe Zarka, and Michel Tagger. "Radioastronomy with LOFAR". In: *Comptes Rendus Physique* 13.1 (2012). The next generation radiotelescopes / Les radiotélescopes du futur, pp. 23–27. ISSN: 1631-0705. DOI: https://doi.org/10.1016/j.crhy.2011.11.002. URL: https://www.sciencedirect.com/science/article/pii/S1631070511002234.

[16] Chuanxiong Guo et al. "RDMA over Commodity Ethernet at Scale". In: *Proceedings of the 2016 ACM SIGCOMM Conference*. SIGCOMM '16. Florianopolis, Brazil: Association for Computing Machinery, 2016, pp. 202–215. ISBN: 9781450341936. DOI: 10.1145/2934872.2934908. URL: https://doi.org/10.1145/2934872.2934908.

[17] Joost Hoozemans et al. "FPGA Acceleration for Big Data Analytics: Challenges and Opportunities". In: *IEEE Circuits and Systems Magazine* 21.2 (2021), pp. 30–47. DOI: 10.1109/MCAS.2021.3071608.

[18] Ernst Joachim Houtgast et al. "GPU-Accelerated BWA-MEM Genomic Mapping Algorithm Using Adaptive Load Balancing". In: *Architecture of Computing Systems – ARCS 2016*. Ed. by Frank Hannig et al. Cham: Springer International Publishing, 2016, pp. 130–142. ISBN: 978-3-319-30695-7.

[19] Yongrui Hu et al. “DCQCN Advanced (DCQCN-A) : Combining ECN and RTT for RDMA Congestion Control”. In: *2021 IEEE 5th Information Technology,Networking,Electronic and Automation Control Conference (ITNEC)*. Vol. 5. 2021, pp. 1192–1198. DOI: 10.1109/ITNEC52019.2021.9586872.

[20] Xilinx Inc. *ERNIC (Xilinx Embedded RDMA enabled NIC) IP*. URL: https://www.xilinx.com/products/intellectual-property/ef-di-ernic.html. (accessed: 25.09.2022).

[21] *Integrated ground-based remote sensing stations for atmospheric profiling*. URL: https://cordis.europa.eu/project/id/2343. (accessed: 30.10.2022).

[22] Anuj Kalia, Michael Kaminsky, and David G. Andersen. “Design Guidelines for High Performance RDMA Systems”. In: *Proceedings of the 2016 USENIX Conference on Usenix Annual Technical Conference*. USENIX ATC ’16. Denver, CO, USA: USENIX Association, 2016, pp. 437–450. ISBN: 9781931971300.

[23] Joseph P. Kenny and Craig D. Ulmer. “RoCE: Promising Technology for Ethernet as a High Performance Networking Fabric.” In: (Oct. 2019). DOI: 10.2172/1573446. URL: https://www.osti.gov/biblio/1573446.

[24] Michael Kerrisk. *clock_gettime(3) —Linux manual page*. URL: https://man7.org/linux/man-pages/man3/clock_gettime.3.html. (accessed: 30.08.2022).

[25] Michael Kerrisk. *perf(1) —Linux manual page*. URL: https://man7.org/linux/man-pages/man1/perf.1.html. (accessed: 30.08.2022).

[26] Michael Kerrisk. *proc(5) —Linux manual page*. URL: https://man7.org/linux/man-pages/man5/proc.5.html. (accessed: 30.08.2022).

[27] Xinhao Kong et al. “Collie: Finding Performance Anomalies in RDMA Subsystems”. In: *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. Renton, WA: USENIX Association, Apr. 2022, pp. 287–305. ISBN: 978-1-939133-27-4. URL: https://www.usenix.org/conference/nsdi22/presentation/kong.

[28] 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.

[29] W. de Laat. *RDMA-data-transport*. URL: https://gitlab.com/astron-misc/benchmark-rdma. (accessed: 01.12.2022).

[30] Yanfang Le et al. “On the Impact of Cluster Configuration on RoCE Application Design”. In: *Proceedings of the 3rd Asia-Pacific Workshop on Networking 2019*. APNet ’19. Beijing, China: Association for Computing Machinery, 2019, pp. 64–70. ISBN: 9781450376358. DOI: 10.1145/3343180.3343190. URL: https://doi-org.tudelft.idm.oclc.org/10.1145/3343180.3343190.

[31] Przemyslaw Lenkiewicz, P. Chris Broekema, and Bernard Metzler. “Energy-efficient data transfers in radio astronomy with software UDP RDMA”. In: *Future Generation Computer Systems* 79 (2018), pp. 215–224. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2017.03.027. URL: https://www.sciencedirect.com/science/article/pii/S0167739X17304715.

[32] Yuliang Li et al. “HPCC: High Precision Congestion Control”. In: *Proceedings of the ACM Special Interest Group on Data Communication*. SIGCOMM ’19. Beijing, China: Association for Computing Machinery, 2019, pp. 44–58. ISBN: 9781450359566. DOI: 10.1145/3341302.3342085. URL: https://doi.org/10.1145/3341302.3342085.

[33] 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.

[34] Qingkai Meng and Fengyuan Ren. “Lightning: A Practical Building Block for RDMA Transport Control”. In: *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*. 2021, pp. 1–10. DOI: 10.1109/IWQOS52092.2021.9521326.

[35] Radhika Mittal et al. “TIMELY: RTT-Based Congestion Control for the Datacenter”. In: *SIGCOMM Comput. Commun. Rev.* 45.4 (Aug. 2015), pp. 537–550. ISSN: 0146-4833. DOI: 10.1145/2829988.2787510. URL: https://doi.org/10.1145/2829988.2787510.

[36] Jiwoong Park et al. “SoftDC: Software-Based Dynamically Connected Transport”. In: *Cluster Computing* 23.1 (Mar. 2020), pp. 347–357. ISSN: 1386-7857. DOI: 10.1007/s10586-019-02926-0. URL: https://doi.org/10.1007/s10586-019-02926-0.

[37] The Chromium Projects. *Trace event profiling tool*. URL: https://www.chromium.org/developers/how-tos/trace-event-profiling-tool. (accessed: 01.12.2022).

[38] 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)*. IEEE. Sept. 2019, pp. 286–292. DOI: 10.1109/FPL.2019.00053.

[39] 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.

[40] David Sidler. “In-Network Data Processing using FPGAs”. en. Doctoral Thesis. Zurich: ETH Zurich, 2019. DOI: 10.3929/ethz-b-000362532.

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

[42] Tianli Song. *Roce based 100gbe RDMA Network Stack on FPGA hardware*. Nov. 2021. URL: https://repository.tudelft.nl/islandora/object/uuid%5C%3Abdc8259e-43e7-4e81-b573-c2f8c1442892?collection=education.

[43] 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.

[44] ETH Zurich Systems Group. *Coyote framework repository*. URL: https://github.com/fpgasystems/Coyote. (accessed: 10.10.2022).

[45] ETH Zurich Systems Group. *ETH’s Heterogeneous Accelerated Compute Cluster*. URL: https://github.com/fpgasystems/hacc. (accessed: 10.10.2022).

[46] Parvin Taheri et al. “RoCC: Robust Congestion Control for RDMA”. In: *Proceedings of the 16th International Conference on Emerging Networking EXperiments and Technologies*. CoNEXT ’20. Barcelona, Spain: Association for Computing Machinery, 2020, pp. 17–30. ISBN: 9781450379489. DOI: 10.1145/3386367.3431316. URL: https://doi.org/10.1145/3386367.3431316.

[47] Kenji Tanaka et al. “Communication-Efficient Distributed Deep Learning with GPU-FPGA Heterogeneous Computing”. In: *2020 IEEE Symposium on High-Performance Interconnects (HOTI)*. 2020, pp. 43–46. DOI: 10.1109/HOTI51249.2020.00021.

[48] Brian Tierney et al. “Efficient data transfer protocols for big data”. In: *2012 IEEE 8th International Conference on E-Science*. 2012, pp. 1–9. DOI: 10.1109/eScience.2012.6404462.

[49] Tianshi Wang et al. “Congestion Detection and Link Control via Feedback in RDMA Transmission”. In: *2022 International Conference on Service Science (ICSS)*. 2022, pp. 1–4. DOI: 10.1109/ICSS55994.2022.00010.

[50] 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.

[51] Yiwen Zhang et al. “Performance Isolation Anomalies in RDMA”. In: *Proceedings of the Workshop on Kernel-Bypass Networks*. KBNets ’17. Los Angeles, CA, USA: Association for Computing Machinery, 2017, pp. 43–48. ISBN: 9781450350532. DOI: 10.1145/3098583.3098591. URL: https://doi.org/10.1145/3098583.3098591.

[52] Xiaolong Zhong et al. “PACC: Proactive and Accurate Congestion Feedback for RDMA Congestion Control”. In: *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*. 2022, pp. 2228–2237. DOI: 10.1109/INFOCOM48880.2022.9796803.