Energy-Efficient Data Transfers in Radio Astronomy with Software UDP RDMA

Przemyslaw Lenkiewicz[a,*], P. Chris Broekema[b], Bernard Metzler[c]

[a]*IBM Research Netherlands, David Ricardostraat 2-4, 1066 JS Amsterdam, The Netherlands*
[b]*ASTRON Netherlands Institute for Radioastronomy, Oude Hoogeveensedijk 4, 7991 PD Dwingeloo, The Netherlands*
[c]*IBM Zurich Research Laboratory, Saumerstrasse 4, 8803 Ruschlikon, Switzerland*

Abstract

Modern radio astronomy relies on very large amounts of data that need to be transferred among parts of astronomical instruments, over distances that are often in the range of tens or hundreds of kilometres. The Square Kilometre Array (SKA) will be the World's largest radio telescope. The data rates of transfers between its components will reach Terabits per second. This will impose a huge challenge on the data transport infrastructure, especially with regard to power consumption, as high-speed data transfers using modern off-the-shelf hardware impose a significant load on the system, including CPU and DRAM usage. The SKA has a strict energy budget which demands a new, custom-designed data transport solution. In this paper we present SoftiWARP UDP, an unreliable datagram-based Remote Direct Memory Access (RDMA) protocol, which can highly increase the energy efficiency of high-speed data transfers and meets the requirements of modern radio astronomy. We have implemented a fully functional software prototype of such a protocol, supporting RDMA Read and Write operations and zero-copy capabilities. We present the measurements of achieved power consumption and data bandwidth and also investigate the behaviour of all examined protocols with respect to packet loss.

Keywords: RDMA, power-efficiency, radioastronomy

1. Introduction

Modern radio telescopes, such as the LOw Frequency ARray (LOFAR) [1] and the upcoming Square Kilometre Array (SKA)[2] are in essence large, distributed sensor networks, characterised by large numbers of sensors producing staggering amounts of data. This data is often generated by custom hardware in remote areas. Receiving and processing these data streams is a computationally intensive task that may consume considerable amounts of energy. In LOFAR, stations produce around 250 Gb per second of sensor data, to be transported over 65 km to the central processor. The SKA will produce several orders of magnitude more data, which is then significantly reduced by the on-site correlator. This reduced data stream, about 3 Tbps per telescope, is then transported to Perth or Cape Town, several hundred kilometres away.

The detailed break-down of such a highly complex science instrument is beyond the scope of this paper, but both the antenna processing, as well as the on-site correlator are expected to use highly specialised custom hardware to generate, reduce and transmit large volumes of data. The Science Data Processor, located in Cape Town and Perth, will be based on general-purpose compute systems. These will turn high volume intermediate data into science-ready data products, a task that is both compute and data intensive.

In [3, 4] the authors have presented a study, which shows that traversing the Linux IP stack for traffic patterns such as these described above may be a significant challenge for underpowered hardware. Considering the very high data rates projected for the SKA it is likely that receiving such data streams will impose a significant load on the receiving systems and consume a lot of energy. The Linux network stack was designed with robustness as the most crucial feature and therefore includes a strict separation between user and system resources. This means that received data is copied several times, triggering several interrupts and context switches, before the user application gains access to it. In [5] the authors have presented how in the Blue Gene system a related bottleneck, namely software handling of Translation Lookaside Buffer (TLB) misses, can be mitigated by bypassing conventional kernel processing. This may significantly decrease resource consumption for specific applications. In this paper we propose a similar approach. Since the Linux IP stack may be a bottleneck while receiving large volumes of sensor data, we propose to avoid the host operating system and place data directly into user memory. While this also bypasses several of the security checks that are essential in typical networking, in a tightly controlled and private network, such as found in a scientific instrument, these are less crucial. We expect a reduction in resource consumption and

*Corresponding author
Email addresses: Lenkiewicz@nl.ibm.com (Przemyslaw Lenkiewicz), Broekema@astron.nl (P. Chris Broekema), BMT@zurich.ibm.com (Bernard Metzler)

therefore a reduction in the amount of consumed energy. Since the SKA Science Data Processor is expected to be bound by very tight budgets, both capital and energy, reducing the cost of receiving data would allow for more science, improving the scientific efficiency of the Science Data Processor.

In the field of modern high-speed networks the Remote Direct Memory Access (RDMA) technology has been proposed to resolve similar issues, namely to allow higher bandwidth, lower latencies and lower CPU utilization. RDMA provides this by moving data directly from the user space memory of one machine to that of another, without involving either of the host operating systems. The application layer is involved only on the side where the request is issued and it can access the contents of memory buffers on a different host thanks to memory pre-registration. The RDMA technology is a very good example of how the data movement process can be optimized for a specific scenario, helping to utilize the full capabilities of the hardware. However, the currently-available RDMA solutions lack some of the features that are necessary for a scenario such as the SKA. In particular, the target scenario both requires a more efficient handling of the expected very high bandwidth-delay product of the data transfer channel, and imposes application specific requirements on time sensitive, partial data transfer reliability [6].

In this paper we address the data transport challenges for modern radio astronomy instruments. We introduce a possible solution that measurably reduces the consumption of CPU resources and energy associated with that data transport. In particular, we design and implement an efficient communication protocol for transferring high rates of astronomical data over long distances with the goal of being more energy efficient at the receiving end.

The main contributions of this paper are:

1. we design and prototype a partially reliable, RDMA-based transport protocol suitable for modern radio-astronomy applications;
2. we present experiments with results showing that the energy efficiency of the prototyped transport stack is improved compared to standard UDP data transfer;
3. we argue that further, more dramatic improvements in efficiency are possible when support for this protocol is implemented in hardware.

2. The Square Kilometre Array

The Square Kilometre Array (SKA) is a new-generation radio telescope which is currently being designed by a large international science and engineering team. Building is expected to commence in 2018 and the first phase is expected to become operational in 2022. SKA phase 1 will consist of two instruments: SKA1-Low, located in Western Australia and SKA1-Mid in South Africa [7]. SKA1-Low is an aperture array instrument consisting of 512 stations, each with 256 dual-polarised antennas, operating between

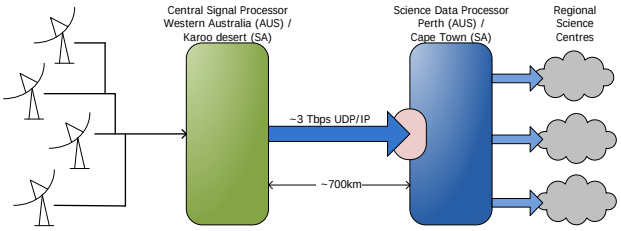

Figure 1: Simplified data transport chart in the SKA, leading from the SKA stations to the Central Signal Processor and the Science Data Processor.

50 and 350 MHz. The stations will be 35 m in diameter and placed at most 65 km apart. The antenna signals are coherently summed per station into a station beam. SKA1-Mid will consist of 133 dishes, with an additional 64 MeerKAT [8] dishes to be integrated into the SKA1 instrument, each with a diameter of 15 m (13.5 m for the MeerKAT dishes), capable of receiving signals between 350 MHz and 14 GHz. The processing of data produced in SKA1-Low and SKA1-Mid is similar and therefore we will not distinguish between them in this paper.

A simplified diagram depicting the SKA data flow is presented in Fig. 1. Data from the receivers is transported to the Central Signal Processor (CSP), located in a highly radio frequency-shielded building in the centre of the telescope. Here, data from all receiver or station pairs are combined into visibilities data by the CSP correlator. The resulting data is transported to the Science Data Processor (SDP), similarly to CSP - one for each instrument, located in Perth and Cape Town, several hundreds of kilometres away. The correlator is expected to be a dedicated FPGA-based system, however GPU-based software correlators are considered. Each of the SDP instantiations receives a continuous data stream of about 3 Tb per second.

The SDP produces science-ready calibrated data products for analysis by the radio astronomer, a task that is highly data intensive and is expected to require compute resources in the 100 PetaFlop range. Data from the SDP are distributed to Regional Science Centres for further analysis and dissemination.

The SKA Science Data Processor is expected to be bound by strict energy and capital budgets that will severely limit the scale of the system. In Section 1 we cite previous work that showed that receiving large volumes of data itself also requires significant resources. The resources consumed just receiving data do not directly contribute to the scientific output of the SDP. In this work we aim to reduce these compute resources required to receive the incoming data stream by avoiding a well known system bottleneck: the Linux IP stack and the associated kernel overhead.

Considering the large distances and volumes of data, it is not feasible to use a reliable data transport protocol for the data transport between CSP and SDP. This would require constant buffering of the transmitted packets at the sending side until confirmations from the receiving side arrive. In a highly optimized real-time environment,

such as the CSP correlator system, this would incur very significant cost and performance overheads. The chosen transmission protocol for this data stream is therefore unreliable, based on UDP/IP over Ethernet, with far lower sender-side overhead. At this point the transported data is highly redundant. Loss of a fraction of this data will result in reduced signal-to-noise ratio in the end-product, but this is, within reason, acceptable. Goal of this work is to maximise the scientific output of the Science Data Processor per invested Euro and/or Joule by minimising cycles spent on data transport that don't directly contribute to the science output.

The specific set of requirements for this particuluar SKA data transport component can be summarised as follows:

- very high data rates, several Terabits per second
- almost entirely uni-directional traffic
- UDP/IP over Ethernet
- prioritising bandwidth over latency
- desire for very high energy efficiency
- full reliability is not crucial, some data loss is tolerable

In the remainder of this paper we investigate how an existing industry standard RDMA implementation can be modified in such a way that it can be used to transport SKA specific data streams. By avoiding a known bottleneck we expect to save a measurable amount of computational resources and energy. This can immediately be translated into increased scientific performance for the same investment.

3. RDMA, iWARP and SoftiWARP

Receiving multiple high-bandwidth UDP/IP data streams requires significant CPU resources. Since CPU cycles can be translated into consumed energy, it can be assumed that a more efficient way to receive large data streams will consume less energy. In addition, compute resources spent on receiving data cannot be utilised for data reduction or processing.

Implemented as an operating system service, the Linux network I/O stack was designed with the main focus on robustness and security while maintaining good performance. Applications access network services via the socket API. To achieve separation and protection, all communication data are copied between application buffers (user space memory) and operating system (kernel) memory within the socket layer. On the transmission path, after copying data into the kernel, network protocol output processing packetises the data, stores it for potential retransmission and informs the network adapter to fetch the packets for wire transmission. In the network packet input path data are first moved from the network card into kernel memory and an interrupt is issued, which handles network protocol processing within the kernel. As a result of protocol processing, kernel data buffers containing the received data are queued to the socket receive queue for application retrieval. Within a system call, the application eventually copies those data from kernel memory to application receive buffers, which typically involves waking up the application thread waiting for data reception. Both in the sending and receiving path, traversing the Linux networking stack incurs non-negligible overhead (interrupt handling, context switches, network protocol processing, data copy operations), which degrades application-available CPU processing power, while limiting achievable communication bandwidth and adding to end-to-end communication latency. Moving the data directly between the network device and application buffer would avoid such overhead, but if not done properly, would violate the data protection and separation principles of the operating system. However, in a tightly controlled and private environment, such as in a scientific instrument, these limitations might be acceptable.

In the past decade the Remote Direct Memory Access (RDMA) technology has been gaining more and more relevance in the field of high-speed communication. Its development was driven by the need for high throughput and low latency networking, especially in High Performance Computing. RDMA provides this by moving data directly from the user space memory of one machine to that of another, without involving host operating system and minimising host CPU usage. The application layer registers memory buffers with the local RDMA-capable network adapter (RNIC) for remote write or read access. Under the control of local and remote RNIC, RDMA write operations transfer data from a local buffer to a tagged remote buffer that was advertised by the peer, whereas the RDMA read operation transfers data from a tagged remote buffer to a tagged local buffer. The application layer is involved only on the side where the request is issued. Any application buffer used as a source or target for an RDMA operation must be pre-registered with the local RNIC device, and is typically pinned into physical host memory. This allows the RDMA device to access the buffer in physical memory without further OS intervention. To allow overlapping communication and computation, RDMA offers an asynchronous communication interface. RDMA operations are posted as Work Requests (WRs) to a communication endpoint and are asynchronously processed by the RDMA device. Work completions are signalled and retrieved asynchronously as well.

RDMA is provided through several network technologies, including Myrinet [9], Infiniband [10], RDMA over Converged Ethernet (RoCE) [11, 12] and iWARP [13, 14]. The functionality and performance of these standards has been evaluated and compared in various studies [15, 16]. Well-known programming interfaces, like the Message Passing Interface, may be used in order to access the RDMA

functionality on different hardware [17].

Both RoCE and iWARP are deployed over Ethernet, which makes them very interesting candidates for the SKA data transport service. RoCE defines the transmission of InfiniBand packets directly over Ethernet, which limits its scope to the Ethernet broadcast domain and thus leaves it non-routable. To solve that issue, a recent protocol extension (RoCEv2) puts it on top of UDP/IP. On the other hand, iWARP defines RDMA operations on top of TCP/IP networks, giving it the advantage of being compatible with the existing Internet infrastructure. Unfortunately, both RoCE and iWARP rely on the implementation of a rather complex protocol state machine (TCP or InfiniBand) meant to provide a level of data transmission reliability which is not needed and even obstructive for the intended use: data to be transmitted have a limited relevance in time – in case of partial data loss the protocol should favor the transmission of new data over the retransmission of lost fragments. Lost data fragments shall result in just dropping the entire affected application level message at RDMA protocol level, while keeping the end-to-end connection intact.

In our work towards an energy-efficient protocol for modern radio astronomy we have chosen the iWARP standard as the baseline, but extended it with an unreliable service. This was achieved by replacing the TCP protocol with UDP and modifying the semantics of the RDMA application interface.

3.1. Implementation of iWARP in software

Although the full range of advantages of RDMA is only available through hardware support for iWARP (in order to offload I/O and protocol processing from the CPU), a software implementation can also be well motivated. iWARP is still a relatively young technology and therefore it is useful to be able to rely on a software solution for testing and development purposes. Furthermore, the software version can be introduced in the less stressed parts of the infrastructure, whereas the more utilised parts would be equipped with iWARP-capable NICs – provided that the software implementation can operate in such a mixed scenario. Thanks to the RDMA semantics and the asynchronous API, even a software implementation can provide benefits such as a zero-copy data transmit path and less application interaction/scheduling, which can lead to increased performance and lowered CPU load and power consumption. Software iWARP can also be used for migrating existing applications to the RDMA interface without the need for RDMA hardware. Finally, it can ease the development of new, experimental extensions to the RDMA stack without hardware prototyping. The SKA scenario is a good example of such a case, as we want to experiment with an implementation of iWARP that is tailored specifically for our needs.

The idea to implement the iWARP protocol fully in software has been already approached and there are solutions available, such as the Software iWARP implementation by the Ohio Supercomputing centre [18],[19] or the SoftiWARP [20] implementation by IBM Research.

3.2. SoftiWARP

The work presented in this paper is based on the SoftiWARP (SIW) open source software implementation of the iWARP protocol suite, developed at the IBM Zürich Research Lab and available from GitHub[1]. SoftiWARP comprises two main building blocks: a kernel module, which implements the iWARP protocols on top of TCP kernel sockets, and a user level library. SoftiWARP integrates with the industry standard OpenFabrics[2] RDMA host stack and thus exports the OpenFabrics RDMA API to both user space and kernel space applications. Due to close integration with the Linux kernel socket layer, SoftiWARP allows for efficient data transfer operations. On the sending side, it supports zero copy data transfers out of application buffers. On the receiving side, the implementation makes use of target buffer address information available with the RDMA protocol headers: the packet payload is directly copied form their in-kernel representation (`sk_buff`) to the final application buffer without scheduling the receiving application. Since the implementation conforms to the iWARP protocol specification, it is wire compatible with any peer network adapter (RNIC) implementing iWARP in hardware.

3.3. Implementing an unreliable connected SoftiWARP service

In order to fulfil the requirements of the SKA we have defined and implemented a new unreliable, connection oriented RDMA transport protocol based on SoftiWARP. Here, communication between hosts is implemented over UDP kernel sockets instead of the reliable, connection-oriented TCP. This holds true for the connection management operations, as well as the data transfer. After connection setup, the application data transfer does not enforce reliability, but is implemented in an unreliable, message-oriented manner: the sender segments the RDMA message into a set of UDP datagrams, which are reassembled on the receiver side into the original message and, if completely received, delivered to the application. Messages which remain incomplete due to UDP packet loss are silently dropped at the receiver.

To retain the efficiency of the original implementation, any inbound, in-sequence data are still directly placed into the application target buffer without intermediate queueing. At API level, error handling has been implemented as simple as possible: if a message remains incomplete due to data loss or corruption, the content of the target buffer remains undefined. If the lost message belongs to an RDMA Send/Receive operation, the current Receive operation remains incomplete and the receive buffer gets

[1] https://github.com/zrlio/softiwarp
[2] https://www.openfabrics.org

re-used for placing the next inbound RDMA Send. Corrupted RDMA Write messages just leave the application buffer in undefined state. While originally not defined for the iWARP protocol, an 'RDMA Write with Immediate Data' operation might further improve the handling of unreliable RDMA Writes at target side: only if the RDMA Write operation completes successfully, the 'Immediate Data' are delivered to the application indicating the complete placement of a new RDMA Write. These data could carry additional application level information such as a message sequence number. Only InfiniBand and ROCE currently define this optional 'Immediate Data' semantics for RDMA Writes. With that, it is currently up to the application to detect corrupted data placed via RDMA Writes.

Unreliable RDMA Read operations are currently supported at an experimental level only. First of all, this operation is not required for the SKA use case: Data streaming is strictly uni-directional and only dictated by the sender delivering radio-astronomic data to a data processing entity. Secondly, supporting unreliable RDMA Reads requires a further extension of the protocol state machine at RDMA Read initiator side, since it must detect permanently lost RDMA Read Request/Response pairs. A timer based detection of message loss appears to be a viable solution to the problem, but is currently not implemented.

The extended SoftiWARP implementation runs on both UDP and TCP and allows to select reliable connection (RC) or unreliable connection (UC) services on a per connection basis. For the UC service, the client side must first create a connection endpoint with an appropriate OpenFabrics service attribute, namely `IBV_QPT_UC`, which represents an Unreliable Connection Queue Pair. On the server side a listener endpoint for the same service type must exist. If the client connects its endpoint with the listener, a new server side endpoint will result, which is associated with the connecting client endpoint. After connection setup, both sides can use the new RDMA association for unreliable data transfer operations.

4. Experiments

In this section we present in-depth tests of SoftiWARP UDP and analyse how a software implementation of iWARP standard is able to perform in terms of achieved bandwidth and power consumption in comparison to standard TCP and UDP sockets. Our test platform comprises two server machines equipped with Intel Xeon E3-1240 v3 CPUs running at 3.40 GHz, 16 GB RAM and Chelsio T5-580 40 Gb RDMA-capable Ethernet cards. The machines are interconnected with a direct connection using a QSFP+ cable. The tests have been performed with:

- The Netperf[3] benchmark tool with additional tests implemented, which carry traffic over RDMA protocols, both over TCP and UDP,
- The LOFAR telescope traffic generator, which creates data packets at rates that correspond to that of a LOFAR telescope station. TCP and UDP Sockets as well as TCP and UDP iWARP is supported for data transport.

We use two measurement points in our experiments to precisely assess the energy consumption of the data transfers. Using the RAPL Technology [21] the values from Intel Processor's registers can be read and the power consumption of the CPU and DRAM can be estimated in a very accurate way. We use the Performance Application Programming Interface (PAPI) library[4] and the Likwid tool[5] to read the power meters. We have also constructed a custom-made power meter based on an Arduino board and voltage sensors attached to the PCI-Express slot. Using this device we can measure the power consumption of the NIC with an accuracy of 1/100 Watt and 1 millisecond sampling rate.

4.1. Power consumption of Chelsio T5

The power consumption of the Chelsio T5 NIC has been measured using the power meter mentioned in the previous section, under numerous different test scenarios. The results of these tests are shown in Fig. 2 in a consecutive manner. The blue line presents the trace of power consumption of the Chelsio T5 NIC. The value of 9 W shows the idle state of the NIC and each peak of around 13.5 W represents one test being carried out. Peaks 1 to 6 represent Netperf tests over different transport protocols in the following order: SoftiWARP TCP, sending side; SoftiWARP UDP, receiving side; TCP sockets, sending side; TCP sockets, receiving side; Hardware iWARP, sending side; Hardware iWARP, receiving side. Tests 7 and 8 represent 50 instances of the LOFAR traffic generator, first the sending side, then the receiving side.

We can see from Fig. 2 that the power consumption of the NIC card is very similar in all cases and doesn't depend on the kind of transport protocol used. Further tests have been performed with varying message sizes and all available transport methods, on sending and receiving side. All of them have shown nearly identical results of 9 W for idle state and 13.5 W for full link speed. Therefore, we can conclude that the power consumption of the RNIC is very consistent and doesn't show a dependency from the type of traffic. In the following sections we will focus only on the CPU and DRAM power consumption, as this is where all of the tested protocols show significant differences.

4.2. Radio astronomy data flow

In this section we mimic the data flow from LOFAR, an operational radio telescope with very similar characteristics to the future SKA. A traffic generator is used to

[3] http://www.netperf.org

[4] http://icl.cs.utk.edu/papi/

[5] https://github.com/RRZE-HPC/likwid

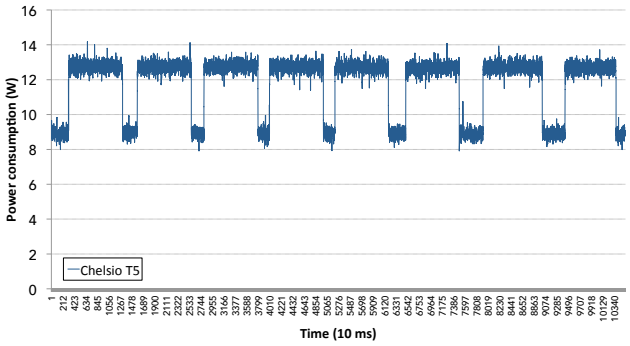

Figure 2: Power consumption of Chelsio T5 during eight consecutive tests using Netperf (tests 1-6) and the LOFAR traffic generator (tests 7-8).

emulate the data produced by a LOFAR Remote Station Processing (RSP) board. This is a UDP/IP data stream, measuring approximately 760 Mb/s, transmitted in packets of 8 kB, which is a limit imposed by local memory on the station FPGA boards. Each LOFAR antenna field produces four of these data streams, totalling slightly more than 3 Gb/s per antenna field. LOFAR currently has 73 antenna fields, 24 core stations which may be split into two independent antenna fields, 18 remote stations and 7 international stations. Three more international stations are under construction, which brings the maximum LOFAR input data rate to almost 230 Gb/s. We generate 50 data streams in our experimental setup, which at 37.5 Gb/s corresponds to roughly $\frac{1}{6}$th of the total LOFAR data flow. Preliminary designs of the SKA system data flow make it likely that data transported between the CSP and SDP will have very similar characteristics, albeit with much higher data rates at longer distances. Our generator is capable of transmitting the said data stream using TCP and UDP sockets and also with TCP and UDP SoftiWarp. In Fig. 3 we show the power consumed by receiving 50 emulated LOFAR data streams using TCP Sockets on

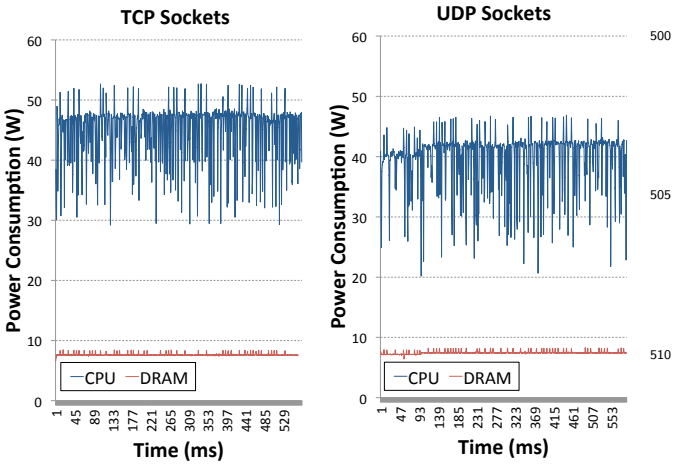

Figure 3: Power consumption of CPU and DRAM for receiving a transfer of LOFAR-like traffic over TCP and UDP sockets.

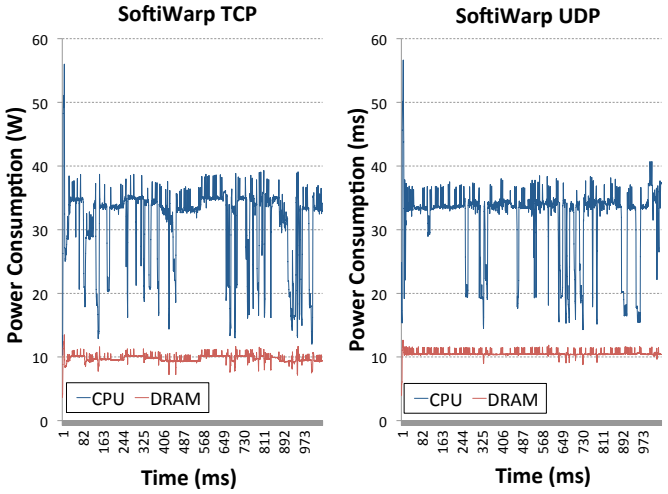

Figure 4: Power consumption of CPU and DRAM for receiving a transfer of LOFAR-like traffic over SoftiWarp TCP and SoftiWarp UDP.

the left image and UDP sockets on the right one. The energy consumption for receiving TCP traffic is measurably higher than when using UDP due to the additional overhead of the TCP/IP protocol stack. This is a clear indication that reducing this protocol overhead will result in a smaller energy consumption. The average power consumption for TCP in this experiment is 45.09 W and for UDP it is 40.05 W. In Fig. 4 we present the power consumption measurements obtained with the LOFAR traffic generator using SoftiWarp TCP on the left image and SoftiWarp UDP on the right image. The power consumption during transfers with software iWarp implementation is clearly lower than in the case of TCP and UDP sockets, presented in the previous image. The average value for the TCP experiment was 32.38 W and for the UDP experiment it was 31.01 W. The power efficiency difference between the TCP and UDP transfer in this case isn't as clear as with the sockets scenario, however the SoftiWarp UDP transfers achieved a better bandwidth, which can be seen on Fig. 5. We can explain this with the fact that the used message size in these transfers is relatively low (8kB) and TCP-based protocol may have a problem achieving full link speed. The UDP-based protocol is more likely to achieve better speeds with smaller messages due to the lower overhead of the unreliable protocol. We will look further into the matter of achieved bandwidth in the following sections and present more results on this subject.

4.3. Power consumption of SoftiWARP TCP

In this section we carry out a set of transfers with the Netperf tool for the Sockets- and RDMA-based protocols with varying message size used. This will allow to observe the behaviour of different transport methods in different scenarios and allow to calculate the theoretical energy efficiency for all the transport methods. The tests have been

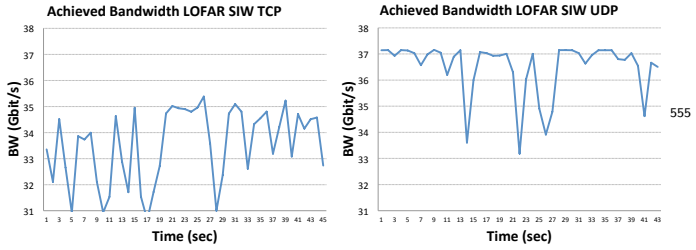

Figure 5: Achieved bandwidth of LOFAR-like traffic on the receiving side using SoftiWarp TCP (left image) and SoftiWarp UDP (right image).

performed with all of the offloading features of the NIC switched off, which was done for two reasons: firstly, we want to assess the direct effect of the transport protocol on the power consumption when no hardware support is available. Secondly, the offloading features available in modern NICs offer significantly more support for TCP protocol compared to UDP protocol, which means that with the offloading turned on the solutions based on the UDP protocol would be penalised. First we present the power consumption traces of different protocols and in Sec. 4.5 we present the complete set of numerical values and evaluate the normalised power consumption per achieved bandwidth. As mentioned before, we are interested in the power

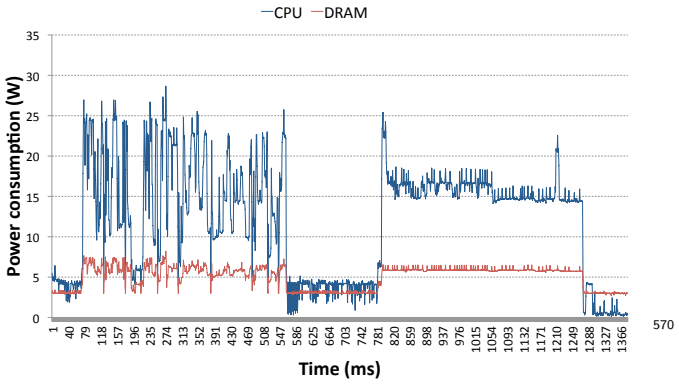

Figure 6: Power consumption of CPU and DRAM for data transfer over TCP sockets (first peak) and SoftiWARP TCP (second peak), receiving side.

consumption on the receiving side of the connection, therefore we initially focus on these results. This is motivated by the fact that the receiving sides of the data transfers in the SKA (CSP and SDP) will most likely be HPC systems, so experiments such as ours can give a good indication on the expected power consumption. Most of the sending side devices, on the other hand, will be custom-built devices. Therefore their power consumption patterns will be significantly different from a standard HPC system and the problem of their power efficiency needs to be addressed on their design level.

Fig. 6 shows the system power trace on the receiving side during data transfer with TCP sockets (first peak) and then SoftiWARP TCP (second peak). The blue line represents the power consumption of the CPU whereas the red line shows the DRAM power consumption. We performed six tests for both TCP sockets and SoftiWARP TCP and compared them to confirm that the power consumption follows very similar patterns in all cases. The data bandwidth achieved during the tests shown in Fig. 6 is 25.1 Gb/s for TCP sockets and 27.85 Gb/s for SIW TCP. As we can see, neither protocol is able to achieve the full link speed when the offloading features are switched off and we are communicating between just two instances of the testing application. However, already we can note that the bandwidth achieved when using SoftiWARP TCP is slightly larger and the power consumption is smaller. The average power consumption from six TCP socket tests is 17.4 W and for SoftiWARP the average is 15.89 W.

4.4. Power consumption of SoftiWARP UDP

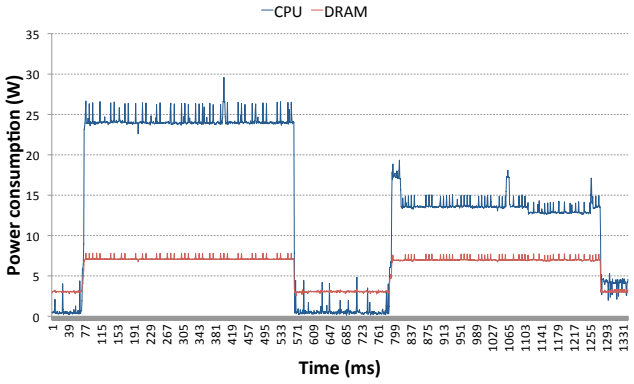

Figure 7: Power consumption of CPU and DRAM for data transfer over UDP sockets (first peak) and SoftiWARP UDP (second peak), receiving side.

Similarly to Sec. 4.3 we have performed the comparison between UDP sockets and SoftiWARP UDP. Fig. 7 presents the system power trace during the execution of two Netperf tests: first peak shows the test using UDP sockets and the second one presents a SoftiWARP UDP test. It is clearly visible that in this case the power consumption difference between standard sockets and SoftiWARP is significant. The average energy consumption in the UDP socket-based tests is 24.21 W and 13.72 W for SoftiWARP-based tests. Furthermore, the near-full link speed of the connection is achieved in both cases: 39.37 Gb/s for the UDP stream test and 38.24 Gb/s for SoftiWARP.

4.5. Comparison of power efficiency

In order to quantify and directly compare the power efficiency of different transport protocols we performed a set of experiments in which we measured the power consumption used by the entire data transfer, including the CPU, DRAM and the NIC. Then we have calculated the

Table 1: Results of power consumption tests with the offloading features of the NIC disabled.

	BW	Send CPU	Send DRAM	Recv CPU	Recv DRAM
TCP sock	24.63	13.44	6.53	15.29	5.61
UDP sock	39.55	24.14	9.59	23.35	7.09
SiwTcp read	17.23	10.77	5.58	23.59	5.66
SiwTcp write	28.83	26.02	7.42	16.15	6.06
SiwUdp read	26.24	14.55	6.64	23.70	6.40
SiwUdp write	38.83	26.58	8.03	13.34	6.98

Table 2: Results of power consumption tests with the offloading features of the NIC enabled.

	BW	Send CPU	Send DRAM	Recv CPU	Recv DRAM
TCP sock	39.35	24.15	7.07	23.19	7.33
UDP sock	39.59	23.96	9.54	23.39	7.12
SiwTcp read	39.41	19.77	8.21	15.64	5.41
SiwTcp write	32.48	22.02	7.01	15.03	6.38
SiwUdp read	39.13	21.57	6.63	24.34	5.29
SiwUdp write	38.79	26.57	8.05	13.64	6.96

normalised power efficiency, which we define as follows:

$$E = \frac{BW}{P} \tag{1}$$

$$[E] = \frac{\text{Gb/s}}{\text{W}} \tag{2}$$

From (1) it can be seen that our metric, the normalised power efficiency (E), is defined as the data bandwidth (BW) divided by the total power consumption (P), expressed in Gigabits per second per Watt (2). With this metric we are able to provide a good comparison on how much power is needed by specific transport protocols in a manner that is independent from the variations in bandwidth in different experiments. We perform six experiments for each value, using message sizes in the range of 8 kB to 2 MB. The tested transport services include: TCP sockets, UDP sockets, SoftiWARP TCP and SoftiWARP UDP – both using RDMA Read and RDMA Write operations. The UDP sockets have only been tested for message sizes up to 64 kB as such size is the largest supported by this transport protocol. As before, during the first tests all of the offloading features of the NICs have been turned off. However, this time we have also performed tests with the following offloading features enabled: rx and tx checksumming offloading, generic receive offload (GRO) and generic segmentation offload (GSO). This was done to see the impact of such features on the results and compare them with the no-offload scenario.

Tables [1] and [2] show example results with hardware offloading features disabled and enabled, respectively. Both tables present results for the following message sizes: 256 kB for TCP-based protocols and 64 kB for the UDP-based protocols. At these values the given protocols have achieved their maximum bandwidth.

The tests performed without hardware offloading demonstrate that the processing of the full TCP stack is a significant load for the CPU. Even a relatively modern system is unable to achieve full link speed using a single core. Only the UDP-based protocols have been able to achieve the near-full link speed, however with UDP sockets this was coupled with significant power consumption on the sending and receiving sides. On the other hand, the SIW UDP tests using RDMA Write have been able to achieve 38.79 Gb/s bandwidth with only 13.64 W of average power consumption on the receiving end. The power consumption on the sending side remains among the highest in the above table, but this is not a crucial issue for radio astronomy applications as the sending side will most likely not be a standard computer but rather a custom-built FPGA unit, designed specifically to issue RDMA Write operations. Therefore, the power consumption of the sending side is a research topic on its own and cannot be evaluated using experiments similar to those presented in this paper.

The results presented in Tab. 2 confirm our assumptions from Sec. 4.3, namely that the TCP-based protocol family receives significantly more support of hardware offloading. In the second set of tests almost all protocols achieved full link bandwidth, except for SIW TCP RDMA Write. The plain UDP Socket test didn't receive any support from the hardware offloading features, achieving the same bandwidth and power consumption. The SIW UDP RDMA Read test has achieved the full link speed due to the Receive Offload and Segmentation Offload features. Finally, it is important to notice that the SIW UDP Write test still offers the lowest power consumption on the receiving side of all the protocols, even when competing with the hardware-supported TCP sockets or SIW TCP.

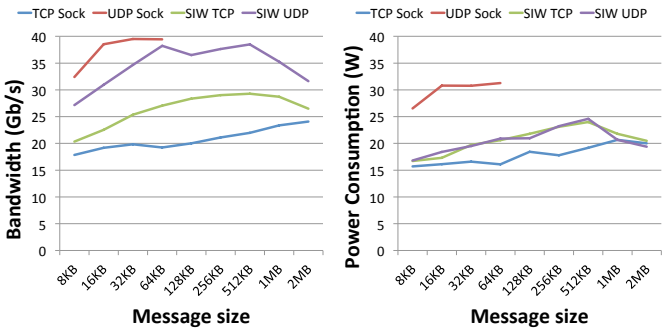

Figure 8: Bandwidth (left panel) and power consumption (right panel) results for tests with varying message sizes.

Fig. 8 depicts how individual bandwidth values (left panel) and power consumption (right panel) correspond to varying message sizes. These charts allow for visualising the trends and optimal values for specific protocols. The achieved bandwidth, but also the resulting power consumption, increases with increasing message size for all protocols. The optimal value is around 512 kB for the

TCP protocols and around 64 kB for the UDP protocols. UDP sockets are the highest consumer of energy of all protocols, but also they are the only ones that achieve the best link speed without hardware support. SIW UDP is able to achieve very similar results with regard to bandwidth, but shows much lower power consumption, therefore its achieved power efficiency is higher.

The above results are used to calculate the normalised values of the power efficiency as expressed by (1) and (2). The calculated values are depicted in the efficiency chart shown in Fig. 9. Comparing the TCP and UDP groups, the former one is less efficient, which can be explained by the low bandwidth achieved by TCP protocols as shown in Fig. 8 left. Comparing SoftiWARP protocols to plain sockets, both over TCP and UDP, we can see that SIW is more power efficient in both cases. In all of the experiments SIW TCP performs better than TCP sockets and SIW UDP better than UDP sockets. This advantage results from the design of the SoftiWARP receive path implementation: after receiving iWARP packets into kernel memory, SoftiWARP directly copies their content into the target application buffers. Making use of the one-sided semantics of RDMA communication this final data placement does not involve the scheduling of the receiving side application process.

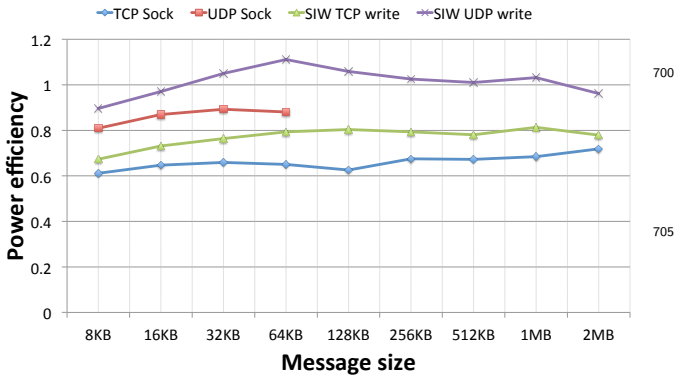

Figure 9: Power efficiency (in Gigabits per second per Watt) of 10 s Netperf tests, receiving side. X axis - message size, Y axis - Power efficiency.

Although these results are still based on software prototype of SoftiWARP UDP, we can already confirm that the reduced data touching and the decreased overhead from the OS lead to very desired characteristics and promising results. The power consumption of SoftiWARP is lower than TCP or UDP sockets in all cases and the achieved bandwidth is at least as good.

4.6. Behaviour in case of packet loss

Finally, we wanted to assess the behaviour of all the tested protocols in case of significant packet loss. We have done this by emulating packet loss using the Netem network emulation tool[6] in the range of 0.1% to 10%. The

[6] https://wiki.linuxfoundation.org/networking/netem

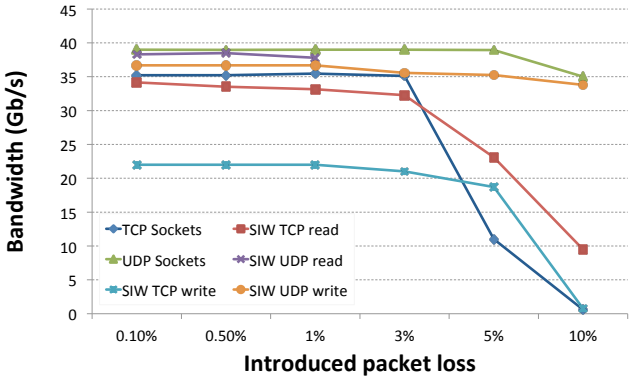

Figure 10: Achieved bandwidth with introduced packet loss.

achieved bandwidths can be seen in Fig. 10. The difference between TCP-based and UDP-based protocols is significant. The former tend to sustain their original bandwidth in the initial part of the tests as all the lost packets are re-transmitted. However, with larger packet loss the network is no longer capable to keep up with re-transmission and the bandwidth gets significantly reduced. The UDP-based protocols do not rely on the retransmission-based reliable communication implemented by the TCP protocol and are able to maintain the transfers on the same level, regardless of the problems occurring along the link. The only decrease in bandwidth is the actual amount of packets that have been dropped. As we can see in the chart, this doesn't hold true for the results of SIW UDP RDMA Reads, which - as discussed earlier - are not yet fully supported in our implementation. The protocol does not yet recover from completely lost RDMA READ request/response pairs, which results in transfer breakdown as soon as the packet loss reached 3%.

The above results show that the use of a protocol that relies on two-way communication and tries to provide full reliability on the transport level, such as the TCP, can be infeasible for a scenario such as the SKA. It is true that the introduced packet loss in our experiments was very high, but the tests were performed for a short, local connection. In the case of the SKA, where the connections spread over hundreds of kilometres in length, we would see a much more drastic influence of packet loss on the achieved bandwidth. This result confirms another reason for the choice of an unreliable transport protocol for our purposes. The power consumption in different packet loss scenarios didn't show any noteworthy behaviour. It corresponded to what we have seen in our previous experiments, namely that with growing packet loss the energy consumption was lower, because the achieved bandwidth was also lower.

5. Conclusions and Future Work

In this paper we presented the data transport requirements of the world's largest radio telescope, the Square

Kilometre Array (SKA). We proposed a solution to meet these requirements, namely an unreliable, datagram-based iWARP protocol implementation. We have then presented a prototype of such a protocol, called SoftiWARP UDP, and evaluated its performance and power efficiency together with those of TCP and UDP sockets. We have confirmed that UDP is a very good choice for long distance transfer of astronomical data. The protocol overhead is lower, which leads to lower power consumption. Furthermore, the use of a reliable transport protocol is not feasible in a scenario such as the SKA, as it (1) leads to higher power consumption, and (2) the data transfer quality soon becomes unacceptable in case of non-negligible data packet loss.

Our software prototype of SoftiWARP UDP is already capable of outperforming TCP and UDP sockets in terms of power efficiency. This is a very desired result, however we expect a much higher improvement of the power efficiency with implementation of the SoftiWARP UDP protocol in hardware, e.g. with FPGAs, which we leave for future work on this subject. In typical RDMA solutions all four lower network layers are handled in hardware. This means that the efficiency of SoftiWARP UDP can be increased significantly, reaching or even surpassing the efficiency of today's hardware RDMA implementations.

Finally, our tests show that the DRAM power consumption has a significant impact on the total consumption and solutions for its reduction should be explored. For this we will look into using flash storage technology instead of DRAM for data ingress, which is energy efficient and offers high bandwidth and low-latency access.

Acknowledgment

This work is conducted in the context of the joint ASTRON and IBM DOME project and is funded by the Netherlands Organisation for Scientific Research (NWO), the Dutch Ministry of Economic Affairs (EL&I), and the Province of Drenthe.

References

[1] M. P. van Haarlem, M. W. Wise, A. W. Gunst, G. Heald, J. P. McKean, et al., LOFAR: The LOw-Frequency ARray, Astronomy & Astrophysics 556. arXiv:1305.3550.

[2] P. E. Dewdney, P. J. Hall, R. T. Schilizzi, T. J. L. W. Lazio, The Square Kilometre Array, Proceedings of the IEEE 97 (2009) 1482–1496. doi:10.1109/JPROC.2009.2021005.

[3] J. W. Romein, P. C. Broekema, J. D. Mol, R. V. van Nieuwpoort, The LOFAR Correlator: Implementation and Performance Analysis, in: ACM Symposium on Principles and Practice of Parallel Programming (PPoPP'10), Bangalore, India, 2010, pp. 169–178.

[4] J. Romein, J. Mol, R. van Nieuwpoort, P. Broekema, Processing LOFAR Telescope Data in Real Time on a Blue Gene/P Supercomputer, in: URSI General Assembly and Scientific Symposium (URSI GASS'11), Istanbul, Turkey, 2011.

[5] K. Yoshii, K. Iskra, H. Naik, P. Beckman, P. C. Broekema, Performance and Scalability Evaluation of Big Memory on Blue Gene Linux, International Journal of High Performance Computing Applications 25 (2011) 148–160, first published online on May 12, 2010. doi:doi:10.1177/1094342010369116.

[6] P. C. Broekema, R. V. van Nieuwpoort, H. E. Bal, Exascale high performance computing in the square kilometer array, in: Proceedings of the 2012 Workshop on High-Performance Computing for Astronomy Date, Astro-HPC '12, ACM, New York, NY, USA, 2012, pp. 9–16. doi:10.1145/2286976.2286982. URL http://doi.acm.org/10.1145/2286976.2286982

[7] P. C. Broekema, R. V. van Nieuwpoort, H. E. Bal, The Square Kilometre Array Science Data Processor Preliminary Compute Platform Design, Journal of Instrumentation 10 (07) (2015) C07004. URL http://stacks.iop.org/1748-0221/10/i=07/a=C07004

[8] J. L. Jonas, MeerKAT - The South African Array With Composite Dishes and Wide-Band Single Pixel Feeds, Proceedings of the IEEE 97 (8) (2009) 1522–1530. doi:10.1109/JPROC.2009.2020713.

[9] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, W.-K. Su, Myrinet: A gigabit-per-second local area network, IEEE micro (1) (1995) 29–36.

[10] G. F. Pfister, An introduction to the infiniband architecture, High Performance Mass Storage and Parallel I/O 42 (2001) 617–632.

[11] H. Subramoni, P. Lai, M. Luo, D. K. Panda, RDMA over EthernetA preliminary study, in: Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on, IEEE, 2009, pp. 1–9.

[12] M. Beck, M. Kagan, Performance evaluation of the RDMA over ethernet (RoCE) standard in enterprise data centers infrastructure, in: Proceedings of the 3rd Workshop on Data Center-Converged and Virtual Ethernet Switching, International Teletraffic Congress, 2011, pp. 9–15.

[13] T. Gross, D. R. O'Hallaron, iWarp: anatomy of a parallel computing system, Mit Press, 1998.

[14] M. J. Rashti, A. Afsahi, 10-Gigabit iWARP Ethernet: comparative performance analysis with InfiniBand and Myrinet-10G, in: Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International, IEEE, 2007, pp. 1–8.

[15] J. Liu, B. Chandrasekaran, J. Wu, W. Jiang, S. Kini, W. Yu, D. Buntinas, P. Wyckoff, D. K. Panda, Performance comparison of MPI implementations over InfiniBand, Myrinet and Quadrics, in: Supercomputing, 2003 ACM/IEEE Conference, IEEE, 2003, pp. 58–58.

[16] M. J. Rashti, A. Afsahi, 10-Gigabit iWARP Ethernet: comparative performance analysis with InfiniBand and Myrinet-10G, in: Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International, IEEE, 2007, pp. 1–8.

[17] J. Liu, J. Wu, D. K. Panda, High performance RDMA-based MPI implementation over InfiniBand, International Journal of Parallel Programming 32 (3) (2004) 167–198.

[18] D. Dalessandro, A. Devulapalli, P. Wyckoff, Design and Implementation of the iWarp Protocol in Software, in: Proceedings of Parallel and Distributed Computing and Systems 2005, ACTA Press, 2005.

[19] D. Dalessandro, A. Devulapalli, P. Wyckoff, iWarp protocol kernel space software implementation, in: Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International, 2006, pp. 8 pp.–. doi:10.1109/IPDPS.2006.1639565.

[20] F. Neeser, B. Metzler, P. Frey, SoftRDMA: Implementing iWARP over TCP kernel sockets, IBM Journal of Research and Development 54 (1) (2010) 5:1–5:16. doi:10.1147/JRD.2009.2036396.

[21] E. Rotem, A. Naveh, D. Rajwan, A. Ananthakrishnan, E. Weissmann, Power-Management Architecture of the Intel Microarchitecture Code-Named Sandy Bridge, Micro, IEEE 32 (2) (2012) 20–27. doi:10.1109/MM.2012.12.