RapidIO as a multi-purpose interconnect

Simaolhoda Baymani[1], Konstantinos Alexopoulos[1,2] and Sébastien Valat[3]

[1] IT Department, European Organization for Nuclear Research, Geneva, CERN CH-1211 Geneva 23, Switzerland

[2] Electrical and Computer Engineering Department, National Technical University of Athens, Heroon Polytechniou 9, 15780 Zografou, Greece

[3] Experimental Physics Department, European Organization for Nuclear Research, Geneva, CERN CH-1211 Geneva 23, Switzerland

E-mail: sima.baymani@cern.ch, konstantinos.alexopoulos@cern.ch, sebastien.valat@cern.ch

Abstract. RapidIO (http://rapidio.org/) technology is a packet-switched high-performance fabric, which has been under active development since 1997. Originally meant to be a front side bus, it developed into a system level interconnect which is today used in all 4G/LTE base stations world wide. RapidIO is often used in embedded systems that require high reliability, low latency and scalability in a heterogeneous environment - features that are highly interesting for several use cases, such as data analytics and data acquisition (DAQ) networks. We will present the results of evaluating RapidIO in a data analytics environment, from setup to benchmark. Specifically, we will share the experience of running ROOT and Hadoop on top of RapidIO. To demonstrate the multi-purpose characteristics of RapidIO, we will also present the results of investigating RapidIO as a technology for high-speed DAQ networks using a generic multi-protocol event-building emulation tool. In addition we will present lessons learned from implementing native ports of CERN applications to RapidIO.

1. Introduction

RapidIO is a high-performance, low pin count, packet-switched system level interconnect standard, applicable to several domains, see Figure 1. Since its first specification, over 200 million RapidIO fabric ports have been deployed. The technology is primarily hardware implemented, guaranteeing low latencies, and error-handling that offloads the CPU. Destination-based routing and support for heterogeneous technologies, see Figure 2, enable the deployment of any network topology. These characteristics establish RapidIO as a fast, robust, and flexible technology that sparks interest in a data center context, where a wide range of technologies and applications define diverse requirements. In this paper we present an overview of our efforts evaluating the technology as well as our experience porting applications to the RapidIO software stack.

2. Background

2.1. Previous work

The main reference to RapidIO is the set of specifications released by RapidIO.org [1]. The volume of academic research around the technology is very limited and focuses on native, on-chip solutions and embedded applications. Relevant work includes the evaluation of RapidIO in systems with high requirements on real time and throughput capabilities [2] as well as heterogeneous, signal processing systems [3].

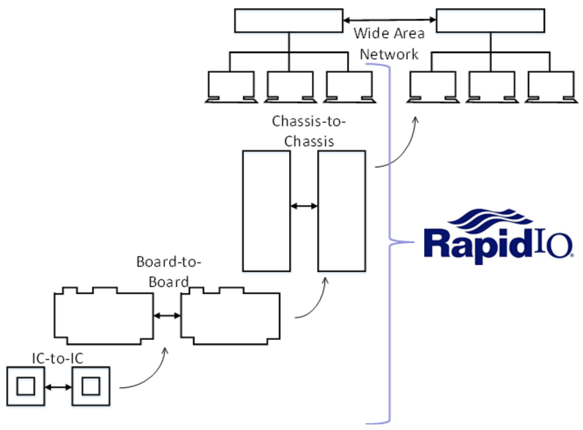


Figure 1. Interconnect application domains where RapidIO is applicable.

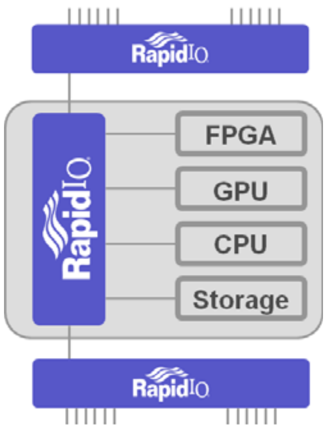


Figure 2. RapidIO use in heterogeneous systems.

With [4] the first investigations in local area networks were done, where work on a data analysis application, ROOT, and an event-building emulation tool, DAQPIPE, is demonstrated. The present contribution builds upon this work.

2.2. The protocol

The RapidIO protocol is based on request and response transactions. All transactions are acknowledged with the use of control symbols. Error handling is done immediately and entirely in hardware, guaranteeing reliable, in-order delivery.

The protocol specifies the following logical operations, among others.

rDMA Remote Direct Memory Access operations are fast and can carry an arbitrary amount of data. They are implemented with zero copy, meaning no additional copy takes place between the user space request and the transfer on the wire. rDMA write operations are also supported as multicast

CM Channelized Messages use a connection-oriented interface, encapsulate data buffers of up to 4KB, and lend themselves well for orchestration, such as exchange of memory mapping information and synchronization of operations

Doorbells Doorbells are RapidIO hardware signals that include the identity of the sender and are often used as acknowledgments

The majority of the above characteristics are implemented in hardware, off-loading the CPU. This not only frees processor cycles but also yields lower latency and lower power consumption.

According to the latest protocol specification, which was released in 2016, speeds of up to 25Gbaud per lane can be achieved, enabling the use of a 100Gb/s network.

2.3. Hardware Setup

Our current setup consists of four 2U Quad units with four Intel Xeon L5640@2.27 Ghz nodes, each with 48GB of RAM, totaling to 16 nodes. Each node is equipped with an IDT Tsi721 PCIe to RapidIO Bridge, offering a line rate of up to 16Gb/s. Every NIC is connected to a 38-port Top of Rack (ToR) RapidIO Generation 2 switch box using QSFP+ cables. The switch ports are configured to 20Gb/s.

The servers are running CERN CentOS release 7.3.1611.

2.4. Library and Interface
The RapidIO software consists of Linux kernel drivers and the RRMAP libraries, provided by IDT. The software package is under active development [5, 6]. User space programs access RapidIO operations through the RRMAP libraries, which interface the underlying drivers.

A considerable change from development versions of the software stack, is that with current versions, 1.x and forward, it is possible to reserve any amount of physical memory from the OS and subsequently use it for RapidIO operations. This enables the rDMA buffers to be virtually any size, limited only by the available physical memory. In order to manage the reserved memory, we implemented a memory pool that allows for dynamic handling of the available memory.

In addition to the packages mentioned, the Centaurus Computing team has developed a kernel module that implements a standard TCP/IP interface over the RapidIO fabric. This enables the use of socket interfaces around RapidIO. These interfaces are called RIOsockets [7].

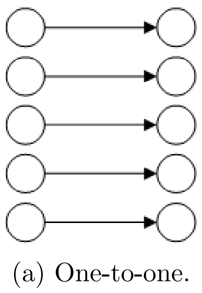
2.5. Link speeds
The RapidIO Tsi721 Gen2 Bridge Card supports 20Gbaud with a 4x lane configuration. Because of the 8b/10b encoding scheme used, this number translates to line rates of 16Gb/s for the NIC. Furthermore, the protocol mappings between PCIe and RapidIO are asymmetric, and require higher overhead in the transmit direction compared to receive. This leads to higher efficiency receive operations that reach a theoretical maximum of 14.8Gb/s compared to 13.4Gb/s for the transmit operations.

To investigate the performance of RIOsockets, we used iperf [8], a commonly used network testing tool. We evaluated two different communication patterns, one-to-one, Figure 3a, and many-to-one, Figure 3b.

In the one-to-one scenario, the speed never surpasses 11Gb/s. The reason is that the reported value for a single client cannot exceed the maximum sending speed. Consequently the transmit speed caps at 11Gb/s, hinting that the network stack overhead approaches 2.4Gb/s.

In the many-to-one scenario, it is evident that increasing the number of clients allows for better receive speeds, see Table 1, which peak at 14.95Gb/s. As the receiver is not limited by the individual client's transmit speed, it is able to serve at the full receive capacity. This is due to the fact that with more clients, the server can better utilize the RapidIO receive queues, yielding even higher speeds.

In both cases we can see that increasing the number of pairs or clients does not affect the speed, see Table 2, suggesting that the switch is overprovisioned and would allow for further scaling.


(a) One-to-one.

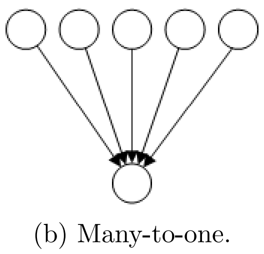

(b) Many-to-one.

Figure 3. Tested iperf communication patterns.

Table 1. Iperf, many-to-one scenario (Gb/s).

#Clients	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Aggregate	10.80	14.47	14.47	13.27	14.81	14.85	14.70	12.89	14.95	14.91	14.95	13.08	13.85	14.43	13.40

Table 2. Iperf, one-to-one scenario (Gb/s).

Nodes\#Pairs	1	2	3	4	5	6	7	8
rio-q1-a	10.9	10.9	11	11	11	11	11	11
rio-q1-c	-	10.7	10.7	10.8	10.7	10.8	10.7	10.7
rio-q2-a	-	-	10.8	10.9	10.8	10.8	10.8	10.9
rio-q2-c	-	-	-	11	10.9	11	10.9	10.9
rio-q3-a	-	-	-	-	10.9	10.9	11	10.9
rio-q3-c	-	-	-	-	-	10.8	10.9	10.8
rio-q4-a	-	-	-	-	-	-	10.8	10.8
rio-q4-a	-	-	-	-	-	-	-	10.8

3. ROOT

ROOT is a data processing framework developed at CERN [9]. Originally targeted at data analysis and simulations related to high-energy physics, ROOT is now used by numerous teams around the world in applications ranging from data mining to astronomy.

3.1. Previous work

ROOT was previously extended to use the RapidIO protocol, instead of the TCP/IP stack, for data transactions [4]. The different operations offered by the RapidIO protocol were individually evaluated, before exploiting all of them, each for different needs.

3.2. Implementation

3.2.1. Initial work. ROOT is designed to expect a stream-oriented socket interface which is inherently incompatible with RapidIO's message based paradigm. To address this the user data is split into chunks before sending, and reassembled on the other side, thus encapsulating the RapidIO-specific logic.

Initially, for orchestration and acknowledgments, CM was used. Orchestration, as well as acknowledgments, require the transfer of information for setting up the connections and defining various parameters. CM can transfer this kind of information in a convenient way. In addition, the establishment of a CM socket is a straightforward procedure. For data transactions, rDMA operations were used, as rDMA is inherently faster and allows for individual buffers of considerably bigger sizes. At the time of this particular implementation, an rDMA buffer was restricted to at most 2MB.

In order to evaluate the performance gap between CM and rDMA, a full CM implementation was explored. Due to the lower measured performance of CM, we subsequently used rDMA for acknowledgments as well, polling the buffer for incoming data, achieving a slight performance improvement.

3.2.2. Current State. In order to reduce the time the program spent polling, a circular buffer design was implemented, see Figure 4. The implemented circular structure consists of several buffers that can be used to send or receive data through rDMA operations. This design allows the sender to post data at will, until the buffer is full, eliminating the need to wait for the receiver to consume the previous batch. The size and length of the circular buffer are calculated at runtime depending on the number of active connections the receiver expects, and the available rDMA-enabled memory.

To further improve the circular buffer extension, the subpar performance of polling rDMA memory had to be addressed. For this purpose doorbells were used, offering a clean and effective implementation. Specifically, after a successful rDMA write, a doorbell is sent, carrying the writer's identification and the circular buffer position written. The doorbell extension remains to be thoroughly tested and thus results presented here are based on rDMA memory polling for acknowledgments.

ROOT is now exploiting the rDMA-enabled memory pool mentioned earlier, which allows for more flexibility as the memory can be freed and reallocated dynamically.

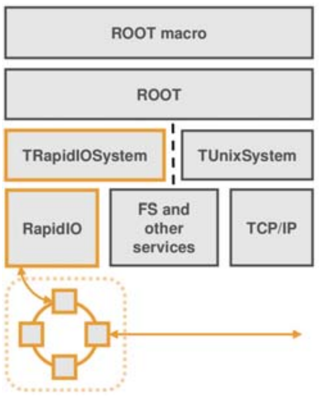


Figure 4. ROOT Architecture.

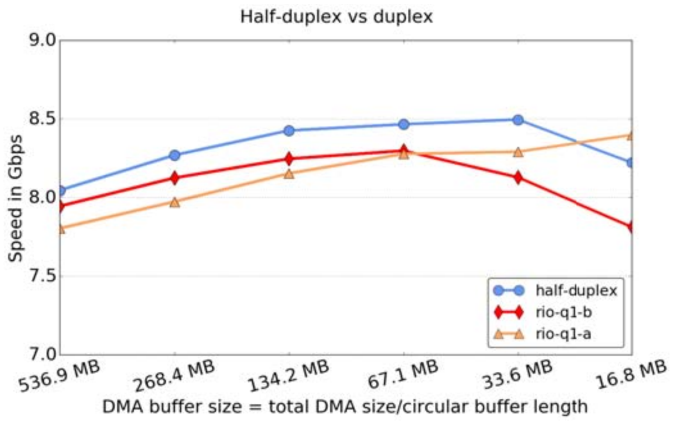


Figure 5. ROOT - half-duplex vs duplex.

3.3. *Results*

We tested the implementations in two different scenarios. In the first scenario, one node assumes the role of the server, which will gather data from a number of client nodes. For the second scenario, pairs of nodes with two-way communication are formed, in order to evaluate the duplex capabilities of RapidIO. For the former, the results are on par with the iperf tests, with the server distributing bandwidth and the clients occupying the resources allocated. The results from the latter scenario are given in Figure 5, showing a speed comparison between half-duplex and duplex communication patterns for a pair of nodes.

The results indicate optimal performance for medium-sized rDMA buffers, roughly between 30 and 130MB, for a bandwidth of about 8.5 Gb/s. However, compared to the iperf performance presented above, ROOT shows a consistently poorer performance. This is due to numerous factors that relate to the complexity of the application. ROOT utilizes its internal bookkeeping, adding an overhead to our calculations. The poorer performance is exacerbated by the acknowledgment mechanism which involves active polling of the rDMA data.

4. Hadoop

Hadoop is a framework used for distributed storage and processing of large data sets. The need for low latencies and high throughput among the nodes of a Hadoop cluster warrants an investigation into the RapidIO technology as a transport layer. Hadoop, as the majority of modern applications, is designed to use the TCP/IP networking stack. Lacking an abstraction class for the networking layer, the advantages of a Hadoop port to RapidIO would be outweighed by the effort required.

Consequently, the relevant Hadoop configuration parameters were set so that RIOsocket interfaces are used in place of the default Ethernet interfaces. This results in the traffic being routed through the RapidIO fabric in a clean manner, eliminating the need for any porting work.

The validation of the setup was done using Intel's HiBench suite [10]. The execution of the suite was successful but proper testing is yet to be done. Next steps involve identifying or developing fitting benchmarks. This will enable the direct evaluation of RapidIO in terms of speed and latency, as well as allowing for a comparison to Ethernet.

5. DAQPIPE

LHCb-DAQPIPE [11], DAQ Protocol-Independent Performance Evaluator, is a protocol-agnostic benchmark application to test network fabrics in preparation for the future upgrade of the LHCb experiment at CERN.

DAQPIPE emulates the behavior of the event-builder in the DAQ network of LHCb, allowing for independent evaluation of protocols, topologies and transports. DAQPIPE has already been ported to several technologies, including Infiniband and 100Gb/s Ethernet [12].

The event-building network is a fully connected network where each node receives a portion of event data from the detector readout system. Event data is therefore spread in fragments across the network and needs to be sorted into distinct events, see Figure 6. This setup generates a bursty many-to-many communication pattern, pushing different parts of the network to their limits. Using the available configuration parameters, changes can be made to the aggregation algorithm, the number of parallel requests, the fragment size, and the number of concurrently handled events (credits). In our work, we have used a pull aggregation scheme where nodes ask other nodes for event data in sequential order and then focused our benchmarks on the number of parallel requests, the fragment size, and the number of credits.

5.1. Implementation

5.1.1. Initial work. The RapidIO port of DAQPIPE uses CM for orchestration and rDMA for data transfers. In the previous description of this work [4], DAQPIPE was ported to an earlier version of the RapidIO libraries. These libraries constrained the rDMA allocations to a maximum of 2MB per allocation, with a maximum of 8 allocations. This limits the possible configurations of DAQPIPE, restricting the number of nodes and credits, as well as the fragment size.

This led to two different versions of DAQPIPE. In the first, the standard one, a buffer is allocated per event and readout units use offsets to write fragments to the event builder units. The second implementation instead allocates one buffer per event fragment, using a multi-segmented approach. This extension was prompted in order to better utilize the network capacity, by matching the maximum rDMA buffer size and the fragment size.

As CM operations are blocking, synchronous rDMA operations were used to keep the implementation simple. Task threads handle send and receive for each open connection.

5.1.2. Current state. The initial update focused on lifting the above limitations, by using the reserved memory feature made available by the latest library versions. The memory pool allows to allocate buffers of arbitrary sizes, enabling DAQPIPE to operate without constraints. Any number of credits or nodes using fragments of any size, is now possible.

Continued refactoring will involve organizing the task threads through a thread pool, which will improve resource utilization in possible edge cases. Another planned change is to run credit processing in lockstep across all nodes, to ensure balanced network activity.

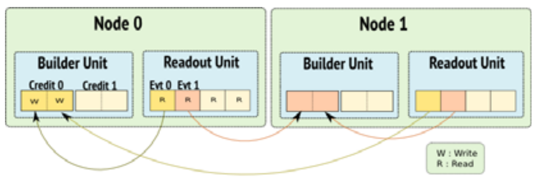


Figure 6. Data aggregation in DAQPIPE [11]. Fragments of events that are spread out.

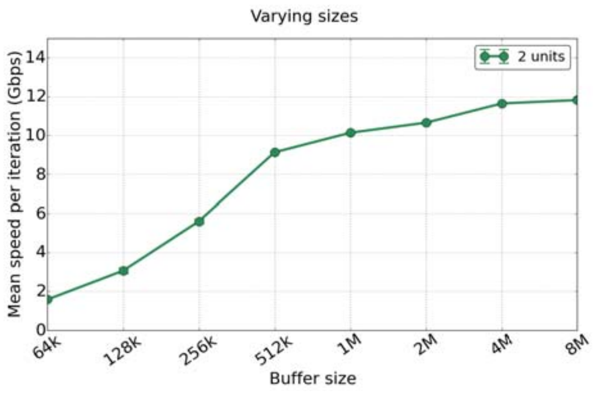


Figure 7. DAQPIPE - Varying buffer sizes (work in progress).

5.2. Results

The memory pool was evaluated by performing a scan of buffer sizes in a point-to-point scenario, see Figure 7. As can be seen on the plot, the increase in size now allows for better utilization of network capacity, resulting in a significant speed increase. In previous results we saw values of up to 10Gb/s, whereas in this implementation allowing for larger allocation sizes, DAQPIPE peaks at 12Gb/s, slightly below the theoretical maximum. These results suggest further investigations into whether larger buffer sizes would increase the speed even more.

Compared to the nominal link speeds reported by iperf, the maximum speed achieved by DAQPIPE falls in between the one-to-one and many-to-one scenarios. This can partly be explained by application overhead, as DAQPIPE uses CM for acknowledgments after each data write which could clearly affect the maximum achievable speed, similarly to ROOT. On the other hand, the above results concern point-to-point communication, which is limited by transmit speed, as analyzed before. Conclusively, the overhead introduced by DAQPIPE is significantly lower than the network stack overhead. Taking into account how DAQPIPE operates, effectively using many-to-many communication, we should be able to see higher speeds. An additional improvement would therefore be to use either rDMA or doorbells for acknowledgments.

6. Conclusions

RapidIO is a technology that has been maturing over many years in embedded and on-chip applications. Our aim is to explore the possibilities of RapidIO in a context outside of its current setting. The results presented in this paper highlight the interesting aspects of this endeavor. However, in its current state, its full capabilities are not properly exposed.

Building on previous work on ROOT, we implemented several improvements since its initial porting. These presented tangible improvements in performance and refined implementations, and helped evaluating the RapidIO technology in a real-world application. Moreover, ROOT has since constituted the test bed for classifying the different operations and coupling them with the functions in which they excel. As ROOT relies on commonly used TCP/IP interfaces, the need for a standardized API becomes evident.

DAQPIPE, on the other hand, demonstrated the impact of paradigm correspondence between application and technology on network utilization. The work done in this paper further supports this conclusion. From the benchmark results it is clear that RapidIO excels when it comes to larger data transfers. Furthermore, the appropriate choice of RapidIO operations highly affects results. Considering the improvements gained by switching from CM to rDMA acknowledgments for ROOT, we should expect similar improvements if the change is implemented for DAQPIPE.

The introduction of the RIOsocket driver opens up many more possibilities, granting access to applications using standard TCP/IP interfaces, such as Hadoop and iperf. The features that RapidIO introduces, such as multicast, could be potentially interesting for alternative uses of Hadoop. Iperf, on the other hand, enables the evaluation of the network. It provides an easy interface to explore how the RapidIO fabric responds to different communication patterns, packet sizes and scaling.

Looking forward, we will assess the use of doorbells in signal exchange between nodes and scaling up to 16 nodes. Moreover, investigations are needed to study how performance will be affected by scaling, specifically in terms of switch provisioning and network load. For Hadoop specifically, efforts will be focused on establishing an appropriate interpretation approach.

Acknowledgments
The authors would like to thank the collaborators and engineers from Integrated Device Technology and our colleagues from the European Organization for Nuclear Research, for their support and collaboration. The authors would also like to thank IDT for funding this technical collaboration with CERN openlab.

References

[1] RapidIO.org, *RapidIO^{TM} Interconnect Specification Version 3.1.* [Online]. Available: http://www.rapidio.org/wp-content/uploads/2014/10/RapidIO-3.1-Specification.pdf.
[2] Bueno D, Leko A, Conger C, Troxel I and George A D 2004 Simulative analysis of the RapidIO embedded interconnect architecture for real-time, network-intensive applications *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*
[3] Changrui W, Fan C and Huizhi C 2010 A high-performance heterogeneous embedded signal processing system based on serial RapidIO interconnection *3rd IEEE International Conference on Computer Science and Information Technology* **2** 611-614
[4] Baymani S, Alexopoulos K and Valat S 2016 Exploring RapidIO technology within a DAQ system event building network *Proceedings of the 20th IEEE-NPSS Real Time Conference*
[5] RapidIO.org 2016 RapidIO Linux Kernel Driver *GitHub* `https://github.com/RapidIO/kernel-rapidio`
[6] RapidIO.org 2016 RapidIO Remote Memory Access Platform software *GitHub* `https://github.com/RapidIO/RapidIO_RRMAP`
[7] Centaurus Computing 2016 RIOsocket *GitHub* `https://github.com/centauruscomputing/riosocket`
[8] ESnet 2016 iperf v2 *GitHub* `https://github.com/esnet/iperf`
[9] Brun R and Rademakers F 1997 ROOT - An Object Oriented Data Analysis Framework *Proceedings, 5th International Workshop, AIHENP'96, Lausanne, Sep. 1996, Nucl. Inst. & Meth. in Phys. Res* A **389** 81-86. See also `http://root.cern.ch/`
[10] Intel® 2016 HiBench Suite *GitHub* `https://github.com/intel-hadoop/HiBench`
[11] Cámpora Pérez D H, Schwemmer R and Neufeld N 2014 Protocol-Independent Event Building Evaluator for the LHCb DAQ System *Proceedings of the 19th IEEE-NPSS Real Time Conference*
[12] Vőneki B, Valat S, Schwemmer R, Neufeld N, Machen J and Cámpora Pérez D H 2016 Evaluation of 100Gb/s LAN networks for the LHCb DAQ upgrade *Proceedings of the 20th IEEE-NPSS Real Time Conference*