Prototyping a 10 Gigabit-Ethernet Event-Builder for the CTA Camera Server

Dirk Hoffmann, Julien Houles
CPPM, Aix-Marseille Université, CNRS/IN2P3, Marseille, France
E-mail: Dirk.Hoffmann@In2p3.Fr, Julien.Houles@In2p3.Fr

Abstract. While the Cherenkov Telescope Array will end its Preperatory Phase in 2012 or 2013 with the publication of a Technical Design Report, our lab has undertaken within the french CTA community the design and prototyping of a Camera-Server, which is a PC architecture based computer, used as a switchboard assigned to each of a hundred telescopes to handle a maximum amount of scientific data recorded by each telescope. Our work aims for a data acquisition hardware and software system for the scientific raw data at optimal speed. We have evaluated the maximum performance that can be obtained by choosing standard (COTS) hardware and software (Linux) in conjunction with a 10 Gb/s switch.

1. Introduction

The Cherenkov Telescope Array [1] is the next generation High Energy Gamma Ray instrument. One observatory is planned in each of the Northern and Southern hemisphere. Each site will consist of about one hundred telescopes of three or four different types, ranging from 4 meters to 24 meters in diameter and using different technologies (single/dual mirror), but with a pixeled, fast photodetection camera (photo-multipliers). We made several assumptions for our first studies: The imaging camera of each telescope is composed of about 2100 pixels grouped in clusters of seven photo-multipliers (PM). The resulting overall schema is depicted in Figure 1. Each cluster contains its readout electronics connected to the data acquisition system via an Ethernet link using the UDP protocol. We have developed a system collecting data from the front-end electronics and reconstructing the complete events for the whole camera.

We describe in this paper the choices that we have made to reach the instrument requirements, first measurements of performance, our conclusions and perspectives for significant improvement on networks at transfer speeds of 10 Gb/s or more. Ethernet standards for largely available 10 Gb/s technology and coming 40 Gb/s or 100 Gb/s are known to deliver less than 50% of the theoretically available bandwidth under standard conditions, as has also been presented by running experiments planning to upgrade to 10 Gb/s technology [2]. Whereas our first studies confirm this fact, we also indicate promising solutions to obtain results up to the nominal bandwidth.

2. Bandwidth considerations

A first evaluation of the data rate generated by the 300 front-end boards has been made in internal studies of the CTA consortium [3]. We are using the orders of magnitude issued from that document. Considering a first level (L1, local to camera/telescope) trigger rate of 10 kHz,
each waveform from one PM containing 72 16-bit samples (sampling rate of 1 GHz), we obtain a rate of about 3 GB/s (24 Gb/s) for the whole camera and 80 Mb/s for one front-end board. As opposed to more conservative approaches, where the genuine data volume would be reduced at the level of front-end electronics by a factor of ten or more, we do assume a “maximum data to ground” approach, which will enable us to do any further treatment of data in a standard COTS and software environment, which can be modified and adapted easily at any time. Each front-end board records seven waveforms for each event, corresponding to the seven PM managed by the board. Each event is labeled with an event number common to all the front-end boards for a given L1 trigger (the L1 trigger is propagated to all front-end boards). Thus, an event contains $72 \times 2 \times 7 = 1008$ bytes for each cluster. With the usual overhead for event number, time stamp etc. we count 1024 bytes for one event from a front-end board. We consider that the boards will be able to send data event by event or by assembling chunks of eight events. Hence, translated into UDP/IP terms, the payload of the data packets passing through the network will be 1024 bytes (regular frames) for one event or 8192 bytes (jumbo frames) for eight-events chunks.

3. Camera Server Architecture
Data coming from the 300 front-end boards, each with an ethernet connection, must be concentrated to a readout computer. We have chosen a set of stacked 48-port switches with a common interconnect backplane for this purpose. Most vendors nowadays offer commercial switches with gigabit (1000baseT) connections and 9K jumbo frames compatibility at low price range already. On the output side, the output rate requires three 10 Gb/s links. The first element of such a switch, purchased for tests is a DELL Powerconnect 6248 with 48 1000baseT ports, four 10 Gb/s ports, four (unused) 1 Gb/s SFP ports and a 184 Gb/s backplane. The full stack will be interconnected with two 10 Gb/ps links except at both ends which will be connected to the camera server (three links).
4. Test Setup
In order to evaluate this solution, we are using one DELL Powerconnect 6248 switch and one DELL Precision T7500 workstation containing a dual port 10 Gb/s network adapter (Intel 82599EB SFP+), two 6-core processors (Intel Xeon X5650) and 24 GB RAM (DDR3-1333 in triple channel configuration). The 10 Gb/s links between switch and computer are carried out in SFP+ standard with copper (not fibre) as a support. As the front-end electronics are presently not available, we simulate data emission by a set of ten HP ProLiant DL145 servers. Each server is equipped with a dual channel 1 Gb/s network adapter and sends data through fifteen different logical ports from each physical port. For each server, the first physical port sends data to the first physical port on the dual 10 Gb/s adapter of the camera server and the second one to the second physical port of the readout computer. Each first physical port is on the same subnet which is different from the second ports. No further “hints” are given to the server in order to organise correct data transfer between the simulated front-end and the camera readout. We have a total of thirty nodes per server, three hundred nodes for the whole setup. Data emission on each server is presently triggered by a timer in order to avoid a too big drift between the servers for any reasons. This configuration is a first approximation to real life conditions. CTA plans to build of a full size data generator made of three hundred physical ports with a synchronised data emission, corresponding to an L1 trigger in the real system later.

![Figure 2. Overview of the test setup hardware: Ten standard with 1 Gb/s 1000baseT outputs feed data into the central ethernet switch. Data run then downstream through two 10 Gb/s connections on a copper SFP+ support to the computer which functions as a Camera Server.]

5. Software Architecture
We developed the software for data reception and event building. The precise electronics specifications are not yet known and will certainly be heterogeneous, depending on the type of telescope. Hence we made a modular design for all aspects of the software. In order to explore the technical limits, we are interested in the most ambitious case in terms of bandwidth. Our approach is to build a prototype able to measure the highest bandwidth possible with standard tools, with minimal CPU load in order to save costs and enable further data processing for software triggering, data reduction, compression and the like. Although we will barely construct more than two test systems in the present phase, cost (and reliability) will become an essential parameter for the final experiment, as the DAQ system will be deployed on about one hundred telescopes, which will be spread over the order of one square kilometer on one site.

All computers run a standard Scientific Linux distribution (SL6 kernel 2.6.32), we use standard blocking UDP/IP sockets, pthread [4] and NUMA [5, 6] libraries. The operating system has been optimized to deal with high speed network devices, in particular, the receive buffers have been enlarged to 16 MB.

In order to balance the load between several cores, we have chosen a multitasks architecture. The incoming flow can be divided and addressed to different channels depending on their
incoming logical port. The different buffers have been allocated on the same memory node and the tasks are bound to different cores, but on the same node. Therefore they will allocate memory blocks in the same physical memory device.

Figure 3. Three-Stage Architecture, from top to bottom: Each front-end board sends data to an individual task (\texttt{taskReceiveDaq}), which is idle until it receives an interrupt, and then stores the received packets in a buffer. One or several tasks (\texttt{taskBuilderEventDaq}, with programmable cardinality) pick up the data from the first buffer and concatenate events in the ring buffer. Inter-task communication is carried out with semaphores. The final task (\texttt{taskComputeEventDaq}) verifies completeness of the events in the ring buffer and gathers them, presently with the only aim to count statistics and destroy them.

We have explored two different architectures for the event building and receiving tasks. In the first architecture (three-stage architecture, Figure 3), we spread data reception — processed by one task for one channel — and event building — which could eventually be spread in turn over a set of tasks in the same channel. Communication is made through a ring packet buffer storing the incoming packets and synchronisation by semaphores. The event building tasks concatenate the events labeled with a same event number in a determined order and store them in a deep ring event buffer accessible to all the channels and in particular to the downstream task which — for the time being — simply checks event integrity and will process the reconstructed events in the future.

The data reception and event building have been integrated in a single task per channel (two-stage architecture, Figure 4) quickly after we had obtained some indications from the results (see below) that the CPU load needed for these tasks was moderate and ran rather naturally in a synchronous manner. In addition this avoids overhead by synchronisation mechanisms and improves temporal locality of data.
Figure 4. Two-Stage Architecture, from top to bottom: As opposed to the three-level architecture (Fig. 3), each front-end board sends data to an individual task (TaskReceiveDaq+BuildEvent, of cardinality 1), which performs reception of data on reception of an interrupt and then stores the partial events directly in the ring buffer. Inter-task communication is carried out with semaphores again. The final task (taskComputeEventDaq) verifies completeness of the events in the ring buffer and gathers them, presently with the only aim to count statistics and destroy them.

6. Results
We obtained the following results for a continuous run with sustained data generation, data reception, event building and permanent checking.

6.1. Three-Stage Architecture
• Jumbo frames (8192 bytes):
  – 19.2 Gb/s (2.4 GB/s) with no loss
  – CPU usage: 300% (3 cores/12)
  – ≈ 8000 events/s reconstructed
• Regular frames (1024 bytes):
  – 6.5 Gb/s (0.8 GB/s) with no loss
  – CPU usage: 300% (3 cores/12)
  – ≈ 2700 events/s reconstructed

The presented bandwidths represent the payload of the UDP packets passing through the both physical ports of the 10 Gb/s adapter, 150 clients emitting to each of these ports. The CPU load is indicative.

Event reconstruction at almost wire speed with the use of jumbo frames loads the CPUs to 300% spread over four cores. Two channels were used, one for each physical port of the
10 Gb/s adapter. For each channel we had one core for data reception task and one core for event building task respectively. The contribution of the compute event task is negligible. This leaves a sufficient headroom for complementary calculations on the same computer. However, regular frames divide the bandwidth by a factor of three at similar CPU load!

Figure 5. Diagram of results: Three aspects (dimensions) of the results are represented in this diagram: 2-stage architecture (left group of bars) versus 3-stage architecture (right group), standard sized UDP/IP frames (blue) versus jumbo frames (red), and finally transfer speed (dark colours, left axis) versus CPU load (light colours, right axis).

6.2. Two-Stage Architecture
As expected, the results for the architecture with integrated data reception and event building are significantly better for small packets

- Jumbo frames (8192 bytes):
  - 19.2 Gb/s (2.4 GB/s) with no loss
  - CPU usage: 160% (1.6 cores/12)
  - ≈ 8000 events/s reconstructed
- Regular frames (1024 bytes):
  - 8.0 Gb/s (1 GB/s) with no loss
  - CPU usage: 170% (1.7 cores/12)
  - ≈ 3300 events/s reconstructed

This time, for a wire speed data rate, the CPU load is only 160 % (spread over two cores) when using jumbo frames. The software performance has been significantly improved with a simplified architecture. However with regular frames, albeit slightly diminished, the performance loss in transfer is still very relevant.

All results are summarised in Figure 5 for direct comparison: Use of Jumbo frames always yields maximum performance. Two-stage architecture allows faster transfer for smaller (regular size) UDP/IP packets, while it decreases CPU load significantly.

7. Conclusion and Perspectives
We developed a standard event-builder system including data generation, which has been used to evaluate two aspects of the event-building, applied to the case of a 2100-pixel/300-IP-node camera:

- data-transfer rates from (simulated) front-end to the event-builder server on one hand
- and the capacity to dispatch, collect and buffer network data as well as the reconstruction computing capacity on the other hand, by means of a standard COTS ethernet switch and multi-core PC respectively.
We see that significant improvements can be obtained by modifying the software architecture and that the full available physical bandwidth can be easily reached with jumbo frames. Comparative investigations showed that these rates are very far from those we reach when data reception is locally replaced by packet generation in memory, while saturating the memory bandwidth. Hence the limiting factor is by far the data reception through the network. Our next efforts will concentrate on this point. Jumbo frames consequently reduce the number of system calls and interrupts. However we have to expect front-end electronics in CTA without the capacity to generate jumbo frames.

As discussions during this conference have shown [7], the problem to exploit modern network capacities fully in point-to-point connections without using jumbo frames is rather common, and a solution that integrates in standard linux distributions without interference or even breaking compatibility is an interesting challenge. Our next step is to improve the performance in reception of regular frames. The situation has been addressed by other research groups (outside physics) already, and first contacts have yielded very promising results.

References
[1] http://cta-observatory.org
[2] Andrea Petrucci et al., Upgrade of the CMS Event Builder, this CHEP conference (2012)
[3] K. Bernlöhr et al., Report of the TRIG Subtask, internal report (19–08–2009)
[4] pthread is the abbreviation for the POSIX implementation of threads and IPC, see for example https://computing.llnl.gov/tutorials/pthreads
[5] NUMA is the abbreviation of non-uniform memory access, see for example http://software.intel.com/en-us/articles/optimizing-software-applications-for-numa
[6] A fundamental introduction into memory handling: Ulrich Drepper, RedHat Inc., What Every Programmer Should Know About Memory (November 2007)
[7] Azher Mughal et al., Evaluation of 40 Gigabit Ethernet technology for data servers, this CHEP conference (2012)