40 Gbps data acquisition system for NectarCAM

Dirk Hoffmann and Julien Houles for the NectarCAM team and the CTA consortium

Aix Marseille Univ, CNRS/IN2P3, CPPM, Marseille, France

E-mail: houles@cppm.in2p3.fr

Abstract. The Cherenkov Telescope Array (CTA) will be the next generation ground-based gamma-ray observatory. It will be made up of approximately 100 telescopes of three different sizes, from 4 to 23 meters in diameter. The previously presented prototype of a high speed data acquisition (DAQ) system for CTA (CHEP 2012, [6]) has become concrete within the NectarCAM project, one of the most challenging camera projects with very demanding needs for bandwidth of data handling. We designed a Linux-PC system able to concentrate and process without packet loss the 40 Gb/s average data rate coming from the 265 Front End Boards (FEB) through Gigabit Ethernet links, and to reduce data to fit the two ten-Gigabit Ethernet downstream links by external trigger decisions as well as custom tailored compression algorithms. Within the given constraints, we implemented de-randomisation of the event fragments received as relatively small UDP packets emitted by the FEB, using off-the-shelf equipment as required by the project and for an operation period of at least 30 years. We tested out-of-the-box interfaces and used original techniques to cope with these requirements, and set up a test bench with hundreds of synchronous Gigabit links in order to validate and tune the acquisition chain including downstream data logging based on zeroMQ and Google ProtocolBuffers [8].

1. Introduction

The Cherenkov Telescope Array [1] will be the next generation ground-based gamma ray observatory. It will consist of two sites:

- One in the southern hemisphere in Chile made up of 4 Large Size Telescopes (LST) of 23 meters in diameter, 25 Medium Size Telescopes (MST) of 12 meters in diameter and about 70 Small Size Telescopes (SST) of 4 meters in diameter.
- One in the northern hemisphere in Canaries with 4 LST and 15 MST.

Half of the MST will be equipped with the NectarCAM camera [2]. Its sensitive area is made of 1855 photomultipliers (PMT) grouped by 7 in a module. For each module, a front end board digitizes the signals and transmits data through a Gigabit Ethernet link. Simulations give a maximum average random trigger rate of 9 kHz, generating an average data rate of 32 Gb/s for the whole camera. Data from all the modules are concentrated into four 10 Gb Ethernet links to a camera server distant of about 2 km which will perform the event building, reduce the data in real time, and send them to a central server. As the consortium encourages to resort to off the shelf equipment, the development of the DAQ requires careful hardware and software choices.

We will develop three main aspects of the system setup. In a first part, we will address the conception of a “DAQ stimulator” dedicated to the acquisition chain validation. In the second part, we will point the issue of data buffering needed to smooth the randomness of events and then how we managed the reception and processing of such big amount of data with standard hardware.
2. The DAQ stimulator

We started the DAQ development before the availability of the camera. Therefore we had to imagine an affordable solution to validate the data acquisition chain. As a pure hardware solution with FPGA would have been too expensive at full scale, we decided to head for a more standard solution using multiple single board computers (SBC) embedding several Gigabit Ethernet ports, each port simulating a module. Each SBC contains a dual-core Intel CPU running Scientific Linux.

2.1.1. An interrupt driven stimulator. When a module fulfils the requirements for a whole camera trigger, it propagates the trigger to all the modules and the data relative to this event are issued from all the modules within a time window of less than 6 ns. Regarding the occurrence of events, they follow a Poisson time-distribution resulting in a fluctuating instantaneous flow of data that we aim to reproduce. For this purpose, the stimulator has to reproduce not only the average data rate of the modules but also the timing profile of the data flow which requires an efficient synchronization of the packet sending.

In a first version of the stimulator we used interrupts triggered by an electrical signal for synchronization.

The time difference between the first packet and the last one issued by a same SBC ($\Delta t_1$ in figure 1) was around 30 $\mu$s and very jittery, which is very far from our goal. This is mainly due to the inefficiency of the Linux network stack and the uncontrolled serialization of the packet sending when using Linux sockets. For $\Delta T_p$, which is the maximum time difference between the packet sending from the SBCs’ first ports, mainly affected by the jitter of the Linux interrupt latency [3] combined with the $\Delta t_1$ jitter, the value was about 100 $\mu$s. With a real time Linux (Xenomai), we obtained 40 $\mu$s.

2.1.2. The Ethernet synchronized stimulator. We redesigned the system synchronization. Each board receives the same temporal scenario which is played synchronously by all the boards. The scenario

$$
\begin{array}{|c|c|c|c|}
\hline
\text{t(s)} & \text{t(ns)} & \text{action} & \text{nb_packets} \\
\hline
0 & 0 & \text{SCENARIO\_PREPARE\_DESCRIPTORS} & 1 \\
0 & 100000 & \text{SCENARIO\_SEND\_PACKETS} & 1 \\
0 & 190000 & \text{SCENARIO\_PREPARE\_DESCRIPTORS} & 1 \\
0 & 200000 & \text{SCENARIO\_SEND\_PACKETS} & 1 \\
0 & 390000 & \text{SCENARIO\_PREPARE\_DESCRIPTORS} & 1 \\
0 & 400000 & \text{SCENARIO\_SEND\_PACKETS} & 1 \\
1 & 0 & \text{SCENARIO\_PREPARE\_DESCRIPTORS} & 10 \\
1 & 100000 & \text{SCENARIO\_SEND\_PACKETS} & 5 \\
1 & 200000 & \text{SCENARIO\_SEND\_PACKETS} & 5 \\
\hline
\end{array}
$$

The scenario

figure 1: Main quantities to tune to reproduce the real camera data flow timing

figure 2: Example of a scenario for packet broadcast from all boards
contains the exact time offsets (from a common start time) when one or several packets have to be sent (figure 2). This requires a very precise synchronization of the internal clocks between the different boards and a reduced response time for packet sending.

The boards are equipped with Intel 82574L Ethernet controllers which provide as standard a PTP hardware clock (PHC) needed to obtain the optimum from the Precision Time Protocol. We dedicate one of the five Gigabit ports of the board to synchronization and slow control purposes. With a common time reference, the boards are able, from a start time given via slow control, to play the common scenario synchronously and send packets synchronously. We chose the linuxPTP [4] library with ptp4l to syntonize the PHC from the network and phc2sys to syntonize the system real time clock from the PHC.

As no built-in feature exists to measure the synchronization of two clocks with the SBC, we slightly modified two Intel i210 Ethernet adapters and the associated drivers in order to feed a 1PPS signal based on the PTP hardware clocks to an oscilloscope, thus measuring the time difference between the two signals as depicted in figure 3.

For a point to point connection, without any other traffic than PTP, we reached a synchronization better than 20 ns between the two PHC.

As the clock must be propagated to all the SBC, we need a switch to spread PTP packets. However PTP switches are too expensive for a low cost stimulator. With very common switches: Netgear

---

**figure 3:** Point to point connection test setup

**figure 4:** Setup for non PTP switches test

**figure 5:** Overall stimulator architecture
GS108E for the eight ports switch and GS724T for the 24 ports, we obtained a synchronization better than 100 ns between slave and master with the setup shown in figure 4.

Scaling up to several dozen slaves is problematic because the delay request message of the PTP is sent by every slave to the master clock. The delay value would be degraded by the queuing effect on the master clock side. We modified the linuxPTP library to disable the delay request message and replaced it by a constant value previously measured by the two i210 boards (one as slave among the other slaves and the other as master). The clock synchronization (measured with the i210 boards) is better than a few hundred ns with 48 slaves. The overall stimulator, which counts 64 SBC providing each 4 ports for data generation, simulates 256 front end boards as represented in figure 5.

2.1.3. Packet sending improvement. As standard Linux sockets do not fulfil the timing requirements, we had to improve the delay and jitter of packet sending. The main issues of Linux concerning our needs are the dynamic allocation of packet buffers, the pileup of numerous functions useless for us and the serialization of the entire sending chain (figure 7) from the socket send call to actual packet sending for each port (it introduces a great delta in sending time amongst the ports on a same board).

We chose to bypass the Linux network stack completely. The stimulator engine runs in kernel space. It statically allocates fixed-size DMA-able memory areas to store the packets to send, which are generated or uploaded from existing data (e.g. simulated data) before the start of the stimulation. The e1000e network driver (for the 82574L chip) has been modified to enable on demand inhibition of its sending chain for the previously registered physical ports. As the stimulator module directly talks to the network hardware, serialization is merely reduced to a write of the last packet to send descriptor address in the tail of each adapter ring which initiates the DMA transfers (figure 6). This action is performed in the main task of the stimulator (with real time priority) which polls on the system real time clock (with different levels of pauses depending on the delay between two consecutive instructions in the scenario) until the time of the next action in the scenario is reached. This polling introduces a granularity of about 200 ns between the boards due to the getnstimeofday() system call latency. Interrupts are disabled on the used core during the scenario execution to avoid uncontrolled context switches.

2.1.4. Performance measurements. We performed measurements with a Napatech NT4E-4T board, equipped with four Gigabit ports able to timestamp the arrival of packets with an accuracy of 7 ns. As it cannot monitor all the ports simultaneously, we measured the time difference between the packets arrival from the first ports of two different SBCs for 10000 events at 50 kHz (figure 8).
All the packets from a same event are sent within a few hundreds nanoseconds. The performance is slightly worse when measuring between ports of different ranks, due to the serialization of packet sending amongst the ports.

3. Data concentration

The 265 Gigabit links are concentrated in Ethernet switches to end up in four ten-Gigabit Ethernet links. The bursty nature of the data flow, due to the instantaneous data sending for a same event from all the producers, and the randomness of the events, constrain us to check if the chosen hardware will be able to cope with these characteristics. Whereas the average flow of 32 Gb/s fits in the uplink to the camera server, the modules can potentially generate 265 Gb/s peaks, which must be buffered. As the module buffers can store only a few packets, no data holding mechanisms, like Flow Control, or lossless protocol, like TCP/IP, are possible. Hence buffering must take place in switches.

This bursty data flow combined to a many producers to few consumers schema is not very common usage, and commercial switches generally do not embed much packet buffer memory. Nevertheless, some manufacturers propose the features we need.

3.1.1. Switches characterization. We performed tests on a low-end switch Dell Powerconnect 6248 (packet buffer 768 kB) and on a deep buffer switch Dell Force10 S60 (packet buffer 1.25 GB) in order to validate that the switch is able to receive packets on all its Gigabit ports exactly at the same time (with a 5 ns maximum jitter) without failing or losing packets and evaluate the actual buffer size available.

For this purpose, a synchronous packet generator has been developed at CEA [5]. It can send synchronous packets on 48 ports with a 1 ns jitter. This equipment validated the point on instantaneous arrival of 48 packets on all ports. We did not encounter any data loss as long as the total incoming data rate was lower than the uplink speed (10 Gb/s).

For the buffer size, we used the synchronous packet generator and the stimulator (and obtained the same results on both setups). When sending data at link speed on each Gigabit port, the PC6248 loses packets from the second event as soon as more than 28 input ports (out of 48) are connected. With the S60, the first packet loss with 40 ports connected occurs after 8900 packets (size on link 1078 bytes) sent per port (i.e. 8900 events). When sending data at a slightly above outbound link speed data rate spread on 40 Gigabit ports, we notice the same behavior, which yields a packet buffer size of about 380 MB.

This value is much lower that the announced 1.25 GB memory and is explained by an additional dynamic packet buffer of 340 MB in the switch and the use of only one out of eight priority queues for each Gigabit switch port. As the modules are not able to manage the class of service, we cannot use
the complete available memory. But such a packet buffer size is sufficient to make data loss probability negligible.

3.1.2. Switches architecture. The first architecture we defined was composed of 6 S60 switches interconnected as in figure 9. However the production of the S60 model has been stopped in 2016 without equivalent replacement.

Therefore we designed a two stages architecture (figure 10) with a first stage of low cost Gigabit switches. Each switch concentrates the data coming from less than 40 modules into four ten Gigabit, so that buffering is not needed. The second stage consists of a ten Gigabit deep buffer switch which concentrates the first stage data from the 28 ten Gigabit links into 4 ten Gigabit links to the camera server. We chose Dell N1548 low cost switches and an Arista 7280R as deep buffer switch with 3 GB of packet buffer memory. This setup will enable the use of the whole memory of the deep buffer switch by assigning a class of service to packets depending on their incoming port on the Gigabit switches. This new setup is being tested.

4. The camera server software
The camera server receives data, performs real time processing like event building (assemble the different fragments of an event coming from all the modules), gain swapping (modify order of the

received high gain and low gain), gain selection (select high gain if no saturation, low gain otherwise) and selection of regions of interest (select only significant pixels) and sends data to a central server with 0MQ (figure 11). The average inbound data rate is 32 Gb/s on four 10 Gb links using UDP with a packet size from 1 to 2 kB. As described in [6], data reception at high rates using regular packets and
the standard Linux stack is not trivial. With a first prototype, we couldn't exceed 8 Gb/s on two 10 Gb links.

4.1.1. Hardware considerations. We use a Dell R730 server with two Intel Xeon 10 cores E5-2660 V3 running at 2.6 GHz, the Network interfaces are three Dell x520 dual port 10 Gb. The best performance is achieved by configuring the driver ixgbe with one receiving queue of 256 buffers and a low interrupt throttle rate. The system embeds 64 GB RAM 2133 in a 4 channels configuration.

4.1.2. Linux stack bypass. The Linux network stack is too slow for high data rates mainly because it makes per packet dynamic allocation, memory copy from kernel space to user space and many system calls. In order to increase throughput we use netmap [7], a framework developed at the University of Pisa. It replaces the Linux network stack by a driver and library using preallocation of buffers at initialization, direct access to packets in Kernel space (with mmap) and access to batches of packets. This software is part of the BSD distribution and compatible with Scientific Linux using recent kernel versions.

4.1.3. Some software optimizations. In order to avoid scheduling actions from the OS, cores are reserved at boot time with isolcpu and each core is dedicated to a specific task. With our current needs in terms of processing, only one processor is used: four cores (four tasks) are used for data reception, two for processing and two for data sending. Inter-task synchronization is completely busy with polling in order to avoid context switching.

Memory placement and the network adapters PCIe slots that we use are NUMA aware. The future evolution of processing needs will be made in a two stages manner. Data reception and the first stage processing (which will strongly reduce data volume) will be bound to the first processor and the second stage processing (higher level processing) and data sending will be bound to the second processor in order to lower costly inter-processors communication. This way, six cores on the first processor will be available for the first stage processing and 6 cores on the second processor will be available for the second stage processing.

Finally, when receiving an event, memory for the next event is explicitly prefetched in cache.

4.1.4. Performances. We performed tests on the above described setup and compared the results with measurements we made on the former event builder prototype [6] (figure 12).

On the outbound side, data are sent through an API developed at the University of Geneva [8] at a rate of 18 Gb/s on two 10 Gb links using 0MQ and a 9000 bytes MTU.

![Speed on four 10 Gb links](image)

**Figure 12:** Data reception and processing rates measured with data from 240 sources on stimulator with rate = frequency × packet size × nb sources, cross-checked with switch bandwidth statistics, CPU usage spread on 4 cores.
This bandwidth overreaches the needs for NectarCAM and saves a comfortable amount of processing power for future needs.

5. Conclusion and perspectives
The NectarCAM DAQ has been in production for two years now with a partial camera at CEA Saclay. The development of the acquisition chain with the stimulator permitted a smooth integration phase, as it worked immediately and the performed tests make us confident for the transition to the full scale camera.

As a consequence of this success, the camera server software will be adapted to other cameras. The Large Size Telescopes cameras are very similar to NectarCAM except higher bandwidth needs and the use of TCP/IP as data transmission protocol from the modules to the camera server. Communication between camera server and the modules is made through 6 ten Gigabit interfaces with a maximum average data rate of 42 Gb/s. Regular switches can be used, as buffering is made in the modules. In order to manage TCP/IP communication we use a framework called mtcp [9], from the KAIST university, South Korea, which is a user level TCP/IP management library on top of netmap. We made modifications to permit reception of streams through several network adapters. First tests show a bandwidth of 14 Gb/s on two ten Gigabit interfaces with 80 clients which seems enough if we transpose to 6 interfaces and should be easy to improve.

Other cameras with higher bandwidth seem interested and software could also benefit to the numerous lower bandwidth cameras by regrouping several cameras in a same physical camera server, thus reducing costs by lowering the number of computers, as well as maintenance, energy consumption and space, which are critical parameters considering the remoteness of the telescope sites.

Finally, this development shows that we can reach high data rates with off the shelf equipment and relatively standard software in a Scientific Linux environment by exploiting hardware at its maximum, which is a more flexible and cheaper approach for DAQ systems compared to custom solutions.

Acknowledgments
NectarCAM is supported by grants from Agence Nationale de la Recherche, P2IO, OCEVU and OSUG2020 labex and by other agencies listed on page:

https://portal.cta-observatory.org/Pages/Funding-Agencies.aspx.

References
[1] Acharya BS, et al., Introducing the CTA concept. Astroparticle Physics, Elsevier, 2013, 43, pp.3-18.
[2] Glicenstein JF, et al., NectarCAM: a camera for the medium size telescopes of the Cherenkov Telescope Array. In ICRC 2015 Proceedings, 2015.
[3] Knutsen T, Performance Evaluation of GNU/Linux for Real-Time Applications. Uppsala University
[4] http://linuxptp.sourceforge.net/
[5] Gu W, et al., Stress testing Ethernet Switches for NectarCAM in the Cherenkov Telescope Array with a synchronous UDP frame generator, The 34th International Cosmic Ray Conference, The Hague, 2015
[6] Hoffmann D, Houles J, Prototyping a 10Gigabit-Ethernet Event-Builder for a Cherenkov Telescope Array. Computing in High Energy and Nuclear Physics, May 2012, New-York, United States.
[7] Rizzo L, netmap: a novel framework for fast packet I/O. Proceedings of the USENIX Annual Technical Conference, June 2012.
[8] Lyard E, et al., Modern middleware for the data acquisition of the Cherenkov Telescope Array, The 34th International Cosmic Ray Conference, The Hague, 2015
[9] Jeong EY, et al., mTCP: A Highly Scalable User-level TCP Stack for Multicore Systems. Proceedings of USENIX NSDI 2014