The Ethernet readout of the DUNE DAQ system

Roland Sipos for the DUNE Collaboration

***Abstract*—In 2023 the Deep Underground Neutrino Experiment (DUNE) Data Acquisition (DAQ) system transitioned to a new Ethernet based readout. This required an extension to the modular readout subsystem: in particular a new I/O device library was implemented, interfacing with the detector electronics; a firmware block was provided by the DAQ team to the electronics experts for the implementation of the data formatting and transmission; the Trigger Primitive Generation (TPG) software in the readout system was adapted to the modified data format. The I/O device library for controlling, configuring and operatinng the Network Interface Controllers (NICs) is built upon the Data Plane Development Kit (DPDK), supporting routing capabilities based on configurable rules. This feature allows the readout to split the data arriving on each 100 Gb/s link into individual data streams (each with a throughput of ∼2 Gb/s), that are passed down to their corresponding processing pipelines for trigger primitive generation and buffering. Extensive monitoring capabilities are also provided by the library, which monitors errors related to data consistency and integrity, and also aids the performance optimization work of the software stack.**

In this contribution we describe the new high-throughput Ethernet based readout integrated into the DUNE DAQ system, and the first performance results obtained at the ProtoDUNE hardware apparatus at the Neutrino Platform at CERN.

***Index Terms*—Data Acquisition, DPDK, Ethernet, HPC**

I. Introduction

THE Deep Underground Neutrino Experiment (DUNE)[1] represents a significant endeavor in particle physics, aiming to unravel the mysteries of neutrinos and their role in the universe's evolution. Central to the experiment is its Data AcQuisition (DAQ)[2] system, designed to capture and process vast amounts of data generated by the experiment's detectors. In a pivotal move towards the use of COTS hardware and standard communication protocols, which reduces the construction effort and cost and increases maintainability, the DUNE DAQ has transitioned to a fully Ethernet-based readout system.

This change was endorsed at the Final Design Review[3] in 2023, acknowledging that the overall system design could accommodate this modification without substantial impact on the rest of the DAQ system. Even at the level of the readout sub-system most of the design[4] could be preserved, with the exception of the data reception block. Instead of custom message exchange and aggregator I/O devices, the UDP/IP protocol over Ethernet was introduced, thus allowing the use fully commercial off-the shelf (COTS) hardware solutions and of third party, open-source, software components. The data flow diagram with the readout system's components and functionalities is shown in Figure 1.

Paper submitted for review on X April 2024.
Roland Sipos is with CERN, Geneva, Switzerland.

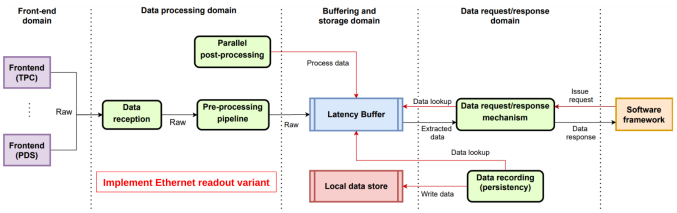

Figure 1: Data flow diagram of the readout subsystem, highlighting the different domains and sub-components. This contribution focuses on the data processing domain that receives data from the front-end electronics based on the Ethernet protocol. The implementation of the buffering and data request response domain can be maintained without sizeable modifications, simply by introducing a modified data format.

In addition to data reception the readout is processing all incoming data to carry out hit-finding and generate trigger primitives, is buffering data in DRAM while the trigger takes its decision, and upon command persists up to 100 seconds of all raw data in high-performance NVMe drives.

II. Front-end domain

The detector electronics transmits data over 10 Gbps links. Those are aggregated into 100 Gbps links via network switches and are fed to the readout unit servers. The overall aggregated data throughput for each of the 4 DUNE Far Detector modules is ∼15 Tbps. There are different detector types with variable throughput. In this contribution results refer to the TPC electronics of the Horizontal Drift and Vertical Drift Far Detector modules.

The DAQ team provides a firmware block developed at Rutherford Appleton Laboratory (RAL) Technical Division[5] that may be integrated into the front-end electronics FPGA boards. This transmitter block is responsible for sending Ethernet frames following the User Datagram Protocol (UDP) where the carried payloads are the front-end electronics data frames. It follows the architecture shown in Figure2.

Every data frame carries also a unified and versioned DAQ header that contains geographic and physical location information about the source of the data stream. It also contains the timestamp from the timing system of the detector, and a sequence identifier for data integrity and continuity checks. Following the header, the frame itself contains data from 64 channels' 64 time slices, resulting in 7200 Bytes long payloads. With the extra protocol headers 7243 Bytes long

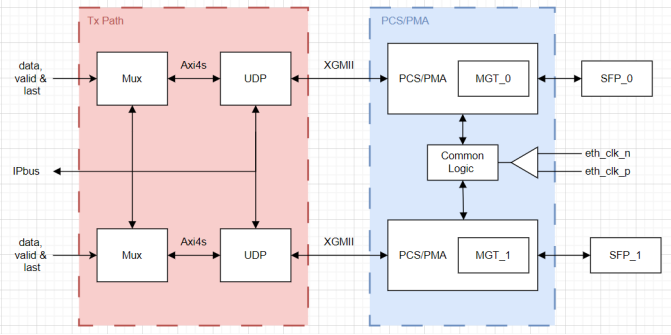

Figure 2: Architecture of the transmitter block provided for the front-end electronics. The PCS/PMA area contains modified Xilinx IP[6] components, and the Tx Path is a custom firmware block developed by engineers at RAL Technical Division. The overall block is responsible for equipping the detector data frames with IPv4 and UDP headers following the communication protocol.

JUMBO UDP frames are transmitted over a switched network to the readout units.

Table **??** describes the characteristics and the numbers of the data streams from these detector components.

Single detector components for charge readout	Links and Data Streams	Payload size and arrival rate	Total throughput (incl. IPv4 and UDP headers)
Anode Plane Assembly (APA)	10 links, 4 streams each Total: 40 streams	7200 Bytes @ 30.5 kHz x 40 streams	~82.5 Gbit/s
Charge Readout Plane (CRP)	12 links, 4 streams each Total: 48 streams	7200 Bytes @ 30.5 kHz x 48 streams	~98.9 Gbit/s

III. Data Plane Development Kit (DPDK)

During the initial integration of the firmware block (called *Hermes*), packet reception and processing tests were carried out with simple applications using posix sockets. Nevertheless, when scaling up the number of data streams, data losses were observed due to performance bottlenecks in the receiving software. Therefore, an alternative and more efficient data reception software was developed. The new software stack for the I/O device control, configuration, monitoring and readout of the NICs in the readout units is built upon the Data Plane Development Kit (DPDK)[7]. It enables more efficient packet processing than the standard interrupt processing that is available in the kernel.

A key element of DPDK is the Poll Mode Drivers (PMDs)[8], that consist of Application Programming Interfaces (APIs) through device drivers running in user space, allowing to configure the devices and their hardware queues. In addition the PMDs have direct access to RX and TX descriptors without any interrupts and extra copies in the kernel space. The run-to-completion model was chosen for our workflow: a specific interface's RX descriptor ring is polled for a burst of packets, which are copied to the user application space for further processing.

Many modern hardware architectures (including x86) now provide Direct Memory Access (DMA) and interrupt remapping facilities in order to ensure I/O devices are isolated within their allocated resource boundaries. The Virtual Function I/O (VFIO) driver is an Input–Output Memory Management Unit (IOMMU) and device agnostic framework for exposing direct access to devices in the userspace. The IOMMU protected environment allows running a safe, non-privileged, userspace driver that can be used in virtualized DAQ environments. The *vfio-pci*[9] poll-mode capable, fully DPDK compatible driver to interface the DAQ software applications was chosen. The readout units' system configuration is IOMMU enabled, automated to allocate huge-pages of memory on NUMA locations where the NICs are connected, such that the interfaces can be bound with the PMD driver.

DPDK has a wide set of core libraries and features, including lock-less multi-producer multi-consumer queues, and the capability of executing callback functions asynchronously on assigned packet processing cores. As the generic readout system has the buffering and data processing functionalities already implemented and specialized for the different front-ends, only a bare minimum set of libraries from DPDK is used, for moving data from the NIC DMA buffers to DAQ userspace applications and streaming them to specific data handler modules. This minimal set of libraries are the following:

- Environment Abstraction Layer (EAL) - Provides the main entry point for configuring and controlling the interfaces. It is responsible for gaining access to low-level NIC resources such as receiver (RX) and transmitter (TX) queues and descriptors. It also provides access for resources on the system like allowed CPU cores for packet processing and Non-Uniform Memory Access (NUMA) aware DMA buffers. Based on the provided configuration parameters its main initialization routine allocates these resources and capable of launching the application specific processing threads.
- Mbuf library - This library provides the ability to allocate and free memory buffers (mbufs) that may be used by the application to store network packets. The underlying header structures are kept as small as possible and currently use just two cache lines, with the most frequently used fields being on either of these. The packet buffer was designed to embed metadata within a single memory buffer followed by a fixed size area for the packet data.
- Mempool library - This library implements a memory pool that is an allocator of a fixed-sized object. It is identified by name and uses a handler to store and free objects. The implementation also offers optional features such as per-core object caching and alignment helpers for efficient padding to spread them equally on all DRAM channels.
- Flow API - It provides generic means to configure the interfaces to match specific traffic and alter its route according to any number of user-defined rules. Matching can be performed on packet data and properties. We use this feature in order to divert packets to specific RX queues based on the source fields in the IPv4 headers,

essentially load-balancing the traffic on available hardware RX rings and descriptors.

- Xstats API - This API allows the poll-mode driver to expose all statistics that are available to it, including statistics that are unique to the device. As calculating statistics of millions of data packets' integrity and validity in software would result in substantial processing overhead, we use this API as an interface to the DAQ's operational monitoring infrastructure.

A *dpdklibs* repository was developed within the DUNE-DAQ software project that includes C++ wrappers, helper functions, and classes to interface test applications and DAQ modules with the features and APIs described above. The currently used DPDK version is 22.11 that matches the one provided by the used operating system (Alma Linux 9) application stream repository.

IV. Software Implementation

Several simple applications were implemented to exercise the different APIs and test the individual features necessary in the DAQ. The DPDK based receiver implementation consists of a new dynamically loadable module implemented within the DUNE DAQ Application Framework. The module implements the standard DAQ module interfaces for configuring, controlling, and monitoring the underlying resources, in this case the network interfaces. The main purpose of this DAQ module is to process the aggregated stream of UDP frames and demultiplex the payloads to their destination modules for further processing and buffering in the generic readout modules. The data flow diagram and different components are shown in Figure 3. The main functional steps are described in the following subsections.

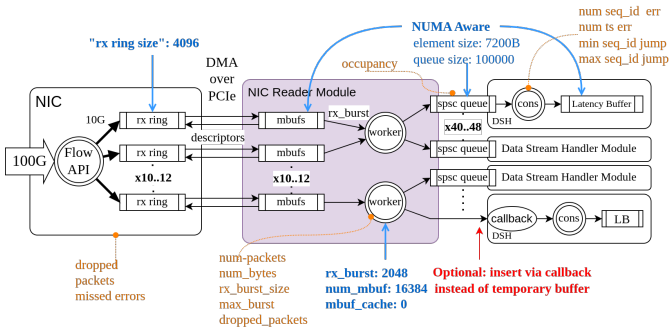

Figure 3: Overview of the DAQ modules within a readout application, with the buffering and processing components. The used configuration parameters (arrows) of these elements are also highlighted with the operational monitoring metrics (dotted lines). Data can be exchanged across DAQ Modules either via queues (as shown in the upper right part ot the diagram) or via a callback mechanism (as shown in the bottom right).

A. Initialization

The overall connection topology between the front-end and the readout modules are established through a detector readout map. This map describes the multiplexing topology between the NIC Reader module and a number of Data Stream Handler modules. The application framework supports intra-process message passing of the data between modules within the same application process via the *IOManager* component. It creates communication channels with buffering queues between sender and receiver modules. To eliminate this extra buffering and copy stage, the readout libraries introduced an optional path for data exchange using a static data move callback registry. Using this registry, the stream handler modules can advertise a payload move function during their initialization. This function can be invoked from other modules within the same application with the right connection identifier inferred from the readout map.

B. Configuration

A single NIC Reader module is capable to handle multiple interfaces, and it is configured with a set of *Interface Wrapper* configurations. Through EAL selected interfaces are initialized and configured with the provided parameters for the wrapper. The necessary steps to ensure that the interface is configured properly are the following:

1) A hardware resource map is established based on the initialized topology to identify the total number of expected IP sources and their corresponding destination streams.
2) Pools of memory buffers are allocated with the total number of expected RX/TX queues for the interface.
3) The requested interface is acquired based on the configured PCIe and MAC addresses, followed by a check for its availability to ensure that it is not occupied by other processes.
4) A reset is carried out on the interface to bring it to an initial and stopped state without any TX and RX queues setup.
5) Upon request the interface is configured with multi-queue Receive-Side Scaling (RSS) and corresponding offloading.
6) The interface configuration is issued with requesting a number of RX and TX queues. The total number of RX queues are the expected number of IP sources. A single TX queue is requested to provide a transmission channel for gratuitous Address Resolution Protocol (ARP) messages.
7) Each requested queue gets configured by binding the previously allocated memory pools for them.
8) Flow Steering and extended statistics are configured.

The Flow Steering configuration consists of pattern matching rules based on the source IP addresses in the IPv4 headers. The rules are defined to route every frame with the same source IP to a dedicated RX ring, resulting in load balancing of available NIC hardware resources. With a successful configuration of the interface, packet processing functions can be launched to do work using the configured RX and TX queues. Also in the configuration a CPU identifier set defines which virtual cores

will spawn the processing threads. The enabled RX queues are assigned to a CPU set in a round-robin fashion, resulting in each requested CPU responsible for processing a number of assigned RX queues. In the final step of the configuration, communication channels are established towards the destination modules either via IOManager or acquiring the callbacks of the downstream modules using the data move registry.

C. Packet processing

Worker threads are spawned on each CPU from the set defined in the configuration. They are polling the assigned descriptors for acquiring a burst of network packets, which are reinterpreted and copied out from the DMA buffers. This is where the main polling of the RX rings is implemented and opportunistic sleep is also added for being able to control the polling frequency in relation to the configured maximum burst size and RX queue depths. A burst call to an RX descriptor through the PMD driver retrieves a maximum number of input packets from a single RX queue of an interface. These packets are stored in the mbufs allocated in the memory pools of the RX queues. As the processing cores are handling multiple RX queues the function has a nested loop over assigned queue burst calls and the processing of received packets one by one.

The pseudo-code of the processing function is seen on Algorithm 1.

Algorithm 1 Packet processor function

```
iface ← confIfaceId                          ▷ Configured parameters
coreid ← confCpuCore
mbsize ← confMaxBurstSize
queues ← rxCoreMap[coreid]
mbufs                                        ▷ Assigned buffers available in scope
while !stopSignal.load() do
    for q : queues do                        ▷ Loop and RX burst queues
        qMbuf* ← mbufs[q.Id]
        nbRx ← rxBurst(iface, q.Id, qMbuf, mbize)
        if nbRx! = 0 then
            for buf : qMbuf do               ▷ Loop on burst results
                if isValidFrame(buf) then
                    payload ← getUdpPayload(buf)
                    handlePayload(payload)
                end if
            end for
        end if
        rxFreeBulk(qMbuf, nbRx)              ▷ Free processed
    end for
    if noFullBurst then                      ▷ Opportunistic sleep
        nanosleep(confSleepUs)
    end if
end while
```

In the pseudo-code the *rxBurst* function initiates the DMA transfer between the NIC and the target memory buffers provided in the parameters list. The *isValidFrame* represents a short sequence of data frame integrity checks for expected packet sizes and protocol headers (e.g.: packet is a UDP frame with correct size). The *getUdpPayload* function returns the memory location of the actual user payload in the Ethernet frame without the IPv4 and UDP headers. The *handePayload* function is where the interpretation of the data happens, and the uniform DAQ header is inspected. Based on the found stream identifier, the pointer to the buffer is routed to a function that carries out the copy into a target readout typed structure. The last step is sending the readout object to its destination stream handler DAQ module, either using the application framework or invoking the callback on the module. The *rxFreeBulk* function is releasing the processed packet buffers for reuse. The opportunistic sleep feature monitors the frequency of full burst occurrences and allows a fine-grain control on CPU core polling and therefore its utilization.

D. Other notable functionalities

If the processing of packets are taking too long, data might be overwritten in the hardware rings. In the DPDK nomenclature this is referred to as *missed packets*, and extended statistics on possible errors and back-pressure are periodically polled out from the NIC in a dedicated thread, and sent to the operational monitoring infrastructure through the appropriate DAQ module interfaces.

The other notable functionality is the gratuitous ARP sender thread. The NIC reader modules periodically send ARP messages per configured interface in order to keep the ARP table updated in the network switches.

E. Trigger Primitive Generation

In the DUNE far detectors all data are processed online, for selection of the interesting events to be stored long-term. The readout sub-system is carrying out the first stage of the processing, by analysing the wave-forms of each individual electronics channel and identifying activity not compatible with electronic noise: this is the so called Trigger Primitive Generation (TPG)[10], since the information about each activity is formatted into a Trigger Primitive data structure which is forwarded to the software based data selection sub-system. The TPG is highly-parallelized and relies on Single Instructions Multiple Data (SiMD) principles using the Advanced Vector Extensions (AVX) to the x86 instruction set. These algorithms are executed in the post-processing component (see Figure 1) that is processing the frames in the latency buffers. It is the most computing intensive and data locality sensitive part of the readout system. The TPG implementation was adapted to the new Ethernet based data format and fully integrated within the system.

V. Performance evaluation and optimization

This section describes the performance evaluation and optimization that were carried out on the individual components of the readout first and on the overall integrated system afterwards.

A. Key Performance Indicators (KPIs)

The overall readout system comes with specialized requirements for the readout units' hardware specification due to its

high-throughput needs. It combines several processing and I/O intensive components. Table I summarizes these elements and highlights criteria for the target servers to be capable to support the readout components requirements.

Table I: Overview of readout components' hardware resources with their utilization sensitivity

Component	Devices and interconnects	CPU	Memory	Persistent storage
Data reception	NICs and PCIe lanes	sensitive	sensitive	
Latency Buffer	Memory and its channels	marginal	sensitive	marginal
Data processing	CPU and cache lines	sensitive	sensitive	
Supernova Burst Data Store	Persistent storage	marginal	sensitive	sensitive

Readout applications consist of several hardware elements and software workloads running in parallel, that are both memory and CPU intensive. The KPI for a single readout unit is the achievable maximum total throughput handled without errors. This translates to the number of 100 Gbps input aggregated data streams handled with all necessary readout components operating in parallel. The underground facility where the readout servers will be located has a strict power budget, hence over-dimensioning the server specifications is not a feasible solution. Along the scaling-up KPI we also aim to identify the bare minimum resource requirements of the functionalities.

B. Integration readout unit specification

We integrated and tested the readout data reception block on a pair of x86 architecture based mid-range performance servers including Intel and AMD processors. The configurations of these servers are shown in Tables II and III.

Table II: Specifications of the Intel integration server

CPU	Intel® Xeon® Gold 6346 @ 3.10 GHz (3.60 GHz turbo), 16-core 2S (dual socket) Code name: Ice Lake 1.5 MiB L1d, 1 MiB L1i 40 MiB L2 72 MiB L3
DRAM	DDR4 512 GB, 3200 MT/s
NIC	Intel E810-CQDA2
OS, DPDK	Alma Linux 9.3, Linux kernel 5.4, DPDK 22.11

Table III: Specifications of the AMD integration server

CPU	AMD® EPYC® 7313 @ 3.00 GHz (3.70 GHz turbo), 16-core 2S (dual socket) Code name: Zen3 Milan 1 MiB L1d, 1 MiB L1i 16 MiB L2 256 MiB L3
DRAM	DDR4 512 GB, 3200 MT/s
NIC	Intel E810-CQDA2
OS, DPDK	Alma Linux 9.3, Linux kernel 5.4, DPDK 22.11

For the tests and results presented a baseline resource allocation strategy is used, where a single CPU socket and its interconnects handle all the readout requirements for a single 100 Gbps input data stream. This is also called a symmetric topology. Asymmetric topology is referred to when different sockets are responsible for certain functionalities: a single socket for data reception and buffering, another socket for data processing and the continuous persistence of data on high-speed storage. Certain architectures (e.g.: AMD Zen3) are latency optimized for dedicated I/O performance of devices on a single PCIe root complex, for which the asymmetric topology is expected to be a more efficient configuration. The baseline topology suits other CPU architectures with features like Direct Cache Access (DCA)[11] that enables the NIC to load and store data directly on the processor's LLC, as conventional Direct Memory Access (DMA) may result with latency bottlenecks between the NIC and CPU. One commercial implementation of DCA is Intel's Data Direct I/O[12], which showed clear and substantial benefits for the analysed workflow.

C. Hardware locality and tuning

Based on the previous experience with high speed I/O devices the servers are configured in performance oriented mode. For BIOS settings recommendations based on the DPDK performance benchmarks[13] and vendor specific tuning guides are used. System-wide power and performance profiles are set to performance mode but deep- and standard sleep states (P/C-states) are operating system driven instead of BIOS specified control. Simultaneous multi-threading features (e.g.: Hyper-threading) are enabled as some readout workloads may benefit from this feature for CPU pipeline utilization and reduced context-switching.

In the operating system low-latency networking tuning profiles for the data request and response low throughput output path are enabled. Power gated sleep states are disabled, performance governor and bias is set to lowest latency mode on the CPU cores executing readout functionalities. Kernel command line parameters ensure to reduce scheduling-clock interrupts and Read-Copy-Update (RCU) callbacks on the data reception sensitive cores. Kernel isolation techniques are also in-place to eliminate any possible kernel interrupts from critical resources. Despite the fact that the DAQ is not using a real-time operating system, acceptable and deterministic latency can be achieved with careful resource access and allocation policies for the readout subsystem's quasi real-time elements.

The high-speed NICs and NVMe drives are connected to dedicated PCIe root complexes without sharing bus resources with other devices. These are also mapped to the closest NUMA node and Last Level Cache (LLC) domains, identifying a set of CPU cores to be assigned to certain functionalities.

D. Monitoring and profiling tools

Standard Linux observability tools are used to gather high-level overview on resource utilization of certain components. On top of these also in-depth processor (e.g.: instructions per second, cache misses), and memory (e.g.: channel utilization) counters are collected with vendor specific monitoring tools

like the Intel Performance Counter Monitor (PCM)[14]. These are interfaced with the DAQ's operational monitoring infrastructure and host-specific metrics are stored, can be visualized on monitoring dashboards, and can be extracted to produce performance reports. The data reception module also publishes the NIC hardware counters provided by the DPDK extended statistics API. Application hot-spots and micro-architecture pipeline utilization are profiled with the Intel VTune[15] and AMD uProf[16] tools to carry out analysis, finding problematic parts in the code, and to devise mitigation strategies.

E. Application optimizations

EAL's runtime environment has strict requirements on hardware locality and requested resources on the system. Memory huge pages are allocated on the used NIC's NUMA node and the closest CPU cores to process the DMA buffers are assigned. The latency buffer implementation supports multiple memory allocation policies and for fixed size and rate payloads we use the *numactl* library provided NUMA aware cache aligned allocator. This makes it possible to have control on buffer placement on desired nodes and take advantage of sub-NUMA clustering on L3 domain features of certain server configurations. Every readout process has command line arguments with their name and its data reception and the processing pipelines threads have unique identifiers assigned. After launching the readout process the parent and internal threads are visible for a custom interrupt balance modifier that modifies the CPU affinity of each PID based on pinning configuration. This approach makes runtime tuning and relocation of threads and also possibility to introduce kernel controlled resource mapping via Control Groups (CGroups)[17].

Using callbacks for data exchange between DAQ modules, the payloads are directly written into the latency buffers. Based on results from standalone test applications the observed improvement is substantial in terms of reduced memory copies and CPU processing compared to the use of intermediate buffering. The callback feature excludes the receiver threads that come with CPU cycles spent in polling the buffers and copying every frame one more time. This leads to non negligible freed up resources as seen on Figure 4 This feature became the default communication model between the data receiver and handler modules and made it possible to use servers with limited resources for detector readout.

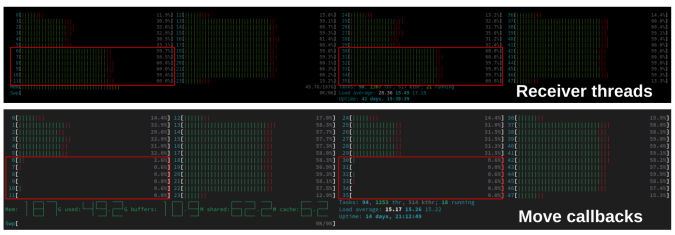

Figure 4: CPU utilization difference between using intermediate buffering and the data move callback functionality. On this dual socket Intel® Xeon® Gold 5118 (Skylake) using callback mode freed up 12 virtual cores, each at ∼60% utilization.

F. NIC configuration optimizations

Purely using system tuning and adding the callback feature still resulted with packets occasionally being missed and dropped. During the investigation of the underlying issue the parameters for NIC's hardware resources and the DMA processor function's configuration were modified. Monitoring metrics of the data receiver module indicated that the interface polling function's (Section IV-C) *rxBurst* method results with the maximum configured number of packets at high frequency. Due to our traffic characteristics the main goal is to poll as many packets as we can in one burst, but without saturating the used resources. We optimized the following configuration parameters:

- RX burst size - We increased the maximum number of packets and descriptors to poll for DMA from a previously set couple hundred to a couple thousand.
- Opportunistic sleep - We decreased the sleep duration to 10 microseconds that increases the polling frequency with the trade-off of higher CPU utilization in case of rare occurrences of empty bursts.
- DMA buffer - Increased the number of DPDK buffer segments (mbuf) to be allocated that essentially made the DMA buffer depth deeper.
- RX descriptors - The number of used hardware descriptors was changed to available hardware limits.

By tuning our processing stack to fix its overall behavior through the global loop policy, the optimized configuration increases the number of used hardware descriptors, allocates more DMA buffer segments, and raises the burst's maximum size. As seen on Figure5 the burst calls are more efficient with the new configuration.

Figure 5: With the optimized NIC hardware and packet processing loop configuration the occurrences of receive bursts resulting with maximum packet counts are drastically reduced. The plots show the number of packets in each burst call from a single interface's 40 RX queues.

VI. Results

The combination of hardware configuration and optimizations described in the previous section allowed for the elimination

of the occurrences of missed and dropped packets. Figure6 shows the acumulated errors statistics for few hours of running of the ProtoDUNE (4 APAs) detector at CERN. In this section the resources utilization of the readout systems components is described, using different types of readout servers.

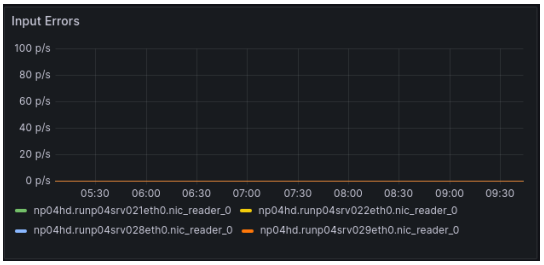

Figure 6: A few hours of a data taking run without errors, consisting 4 APAs of the ProtoDUNE detector at CERN read out by 4 readout unit. The shown metric is an aggregated counter of different error conditions, which also contains the number of dropped and missed packets that may occur due to packet processing performance issues.

A. Integration results

For handling 100 Gb input with a single application, its processing threads and buffers are assigned to certain CPU groups and NUMA nodes. During the integration of these components the utilization of certain system resources was measured and their assignments fine-tuned for each server.

1) Data reception resources: The data reception has ∼10 GB/s memory throughput with 2-4 physical cores needed to process the DMA buffers. CPU utilization varies based on the number of cores, frequency, and LLC size. The DMA buffer size in memory huge-pages is 10 GB.

2) TPG resources: The most CPU intensive component's resource utilization heavily depends on the used algorithm and produced TP rate. The assigned CPU core count varies based on the CPU model due to the available AVX engines, clock frequency, and cache line sizes. Using 10 cores (5 physical and their hyper-core pairs) of the integration servers, the utilization per core is ∼60%, running the simpler algorithm.

3) SNB recording resources: The recording threads are assigned to 4 cores, each at 100% utilization when the recording is active. It requires only 10 GB/s memory bandwidth, which is achieved using direct I/O from the latency buffers into the NVMe drives.

B. Scale-up demonstration

As a subsequent step, the network I/O capacity on a readout server was doubled, with the aim of running two readout applications, one per socket, each handling one CRP detector component. Table IV shows the characteristics of the readout server.

The number of components and their threads with the CPU mask and resource utilization footprint is summarized in TableV.

Table IV: Characteristics of the scale-up demonstrator server

CPU	Intel® Xeon® Gold 6448H @ 2.40 GHz (4.10 GHz turbo), 32-core 2S (dual socket) Code name: Sapphire Rapid 3 MiB L1d, 2 MiB L1i 128 MiB L2 120 MiB L3
DRAM	DDR5 1.0 TB, 4800 MT/s
NIC	2 x Intel E810-CQDA2 (1 per socket)
Drives	6 x 7.68 TB U.3 NVMe drives Samsung 980 Pro (3 per socket)
OS, DPDK	Alma Linux 9.3, Linux kernel 5.4, DPDK 22.11

Table V: Overview of the number of threads of each components and the assigned CPU cores to be used to handle 2 CRP detector elements. CPU cores are assigned with their corresponding Hyper-thread (HT) core included. The last column indicates the maximum CPU utilization percentage of the assigned CPU cores during the test. The latency buffers' capacity is configured to pre-allocate memory for ∼10 seconds' worth of data, which is ∼196 GB in total.

Component	Number of threads	CPU cores assigned	Maximum CPU core utilization (%)
Data reception (Packet processors)	8	4 phys. and 4 HT	∼48.2
Data processing (TPG)	96	10 phys. and 10 HT	∼55.8
Supernova Burst (Recording)	96	8 phys. and 8 HT	∼52.6

The system wide resources utilization of CPU and memory bandwidth are shown in Figure7. It shows how the symmetric topology results in an equal balancing between the two sockets: this is expected since each socket has an identical workload. On each socket the overall readout workload uses ∼19% of CPU resources and ∼37 GB/s memory bandwidth. When the Supernova Burst recording is enabled for over 100 seconds, the CPU utilization peaks at ∼32% and the memory bandwidth utilization increases with the expected 10 GB/s, resulting with ∼47 GB/s.

C. Future work

There are several remaining tests and configurations that are under evaluation in order to establish the optimal readout subsystem implementation for the DUNE experiment, considering many factors such as power efficiency, flexibility, modularity for fault tolerance, and cost.

1) Scale-up to 400 Gbps: The tests with the demonstrator server whowed that there is substantial headroom available in terms of processing capabilities. Therefore, reading out four detector components with a single readout unit is being considered. This requires two 200 Gbps capable network interfaces in order to demonstrate the data reception and trigger primitive generation of 400 Gbps input data streams with a single server.

2) Bare minimum requirements: The readout servers will be located in a deep underground facility with a strict power budget, therefore it is important to find the right balance between available resources and their power requirements. Power draw measurements are ongoing in order to find the

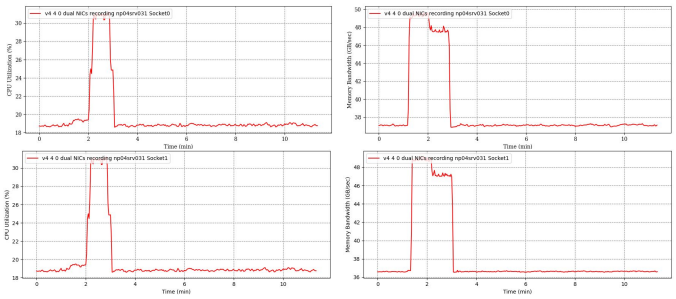

Figure 7: Overall resources utilization on the scale-up demonstrator server's two CPU sockets. The upper row corresponds to the first socket and the lower row to the second socket. The first column shows the CPU utilization of the node and the second column the memory bandwidth utilization. Data is received from the two detector components (∼200 Gbps) and buffered for 10 seconds on their corresponding NUMA node. Trigger Primitive Generation finds the hits in the data frames, forms and sends aggregated sets downstream. The peaks in the plots are highlighting the activation of the Supernova Burst recording, which continuously persists the full data stream to the NVMe drives.

optimal CPU and memory requirements and the right concentration strategy of how many detector components will be read out by a single readout unit.

3) Asymmetric topology: The integration with an AMD server highlighted that scaling up the system for this platform requires allocating readout components differently due to specific hardware features and constraints. Work is ongoing to compare the resource utilization behavior of different placement strategies and to decide on the proposed topology.

VII. Conclusions

The Ethernet readout is successfully integrated into the DUNE DAQ system and is used in standard operations for the ProtoDUNE detector prototypes at the Neutrino Platform at CERN. The full readout feature set and requirements were validated and demonstrated using multiple generations of CPU servers.

The introduction of Ethernet as the detector readout technology allowed to focus the effort on software and tuning of servers and NICs, instead of custom hardware and protocols development and testing. Thanks to the overall readout subsystem optimization it was possible to demonstrate that ∼5-years old servers are capable of successfully implementing the full readout functionality for one detector unit (100 Gbps) and that a more recent server can be used to readout two detector units (200 Gbps).

Scalability studies and further performance evaluation with different hardware components and topologies are ongoing in order to finalize the readout units' technical specification, in order to launch the DAQ procurement for the first far detector next year.

References

[1] *Dune homepage*. [Online]. Available: https://dunescience.org.

[2] A. Abed Abud, C. Batchelor, K. Biery *et al.*, 'Trigger and data acquisition (tdaq) system design,' DUNE DAQ Project, Tech. Rep., Jan. 2023. [Online]. Available: https://edms.cern.ch/document/2812882 (visited on 25/07/2023).

[3] R. Sipos, 'Dune daq readout final design review,' Tech. Rep. [Online]. Available: https://edms.cern.ch/ui/file/2826457/1/DUNE-DAQ-FDR-Readout.pdf.

[4] *Design of a request/response buffering application for i/o intensive workloads*. [Online]. Available: https://iopscience.iop.org/article/10.1088/1742-6596/2438/1/012025 (visited on 05/05/2024).

[5] *Rutherford appleton laboratory*. [Online]. Available: https://www.ukri.org/who-we-are/stfc/facilities/rutherford-appleton-laboratory/ (visited on 05/05/2024).

[6] *10g/25g high speed ethernet subsystem product guide*. [Online]. Available: https://www.xilinx.com/content/dam/xilinx/support/documents/ip_documentation/xxv_ethernet/v4_1/pg210-25g-ethernet.pdf (visited on 05/05/2024).

[7] *Dpdk homepage*. [Online]. Available: https://dpdk.org.

[8] *Pmd driver*. [Online]. Available: https://doc.dpdk.org/guides/prog_guide/poll_mode_drv.html (visited on 05/05/2024).

[9] *Vfio driver*. [Online]. Available: https://docs.kernel.org/driver-api/vfio.html.

[10] *Hit finding algorithms for the dune experiment using single instructions multiple data parallel processing*. [Online]. Available: https://indico.tlabs.ac.za/event/112/contributions/2813/attachments/1186/1610/Adam_Abed_Abud_TPG_TIPP_230906.pdf (visited on 05/05/2024).

[11] *Understanding i/o direct cache access performance for end host networking*. [Online]. Available: https://dl.acm.org/doi/10.1145/3508042 (visited on 05/05/2024).

[12] *Intel data direct i/o technology*. [Online]. Available: https://www.intel.com/content/www/us/en/io/data-direct-i-o-technology.html (visited on 05/05/2024).

[13] *Dpdk performance reports*. [Online]. Available: https://core.dpdk.org/perf-reports/ (visited on 05/05/2024).

[14] *Intel performance counter monitor*. [Online]. Available: https://github.com/intel/pcm (visited on 05/05/2024).

[15] *Intel vtune profiler*. [Online]. Available: https://www.intel.com/content/www/us/en/developer/tools/oneapi/vtune-profiler.html (visited on 05/05/2024).

[16] *Amd uprof*. [Online]. Available: https://www.amd.com/de/developer/uprof.html (visited on 05/05/2024).

[17] *Redhat - control groups*. [Online]. Available: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/resource_management_guide/chap-introduction_to_control_groups (visited on 05/05/2024).