SISSA

Software-based data acquisition and processing for neutron detectors at European Spallation Source—early experience from four detector designs

View the article online for updates and enhancements.

IOP ebooks™

Bringing you innovative digital publishing with leading voices to create your essential collection of books in STEM research.

Start exploring the collection - download the first chapter of every title for free.

TECHNICAL REPORT

Software-based data acquisition and processing for neutron detectors at European Spallation Source — early experience from four detector designs

M.J. Christensen,[a,1] **M. Shetty,**[a] **J. Nilsson,**[a] **A. Mukai,**[a] **R. Al Jebali,**[b,c] **A. Khaplanov,**[b] **M. Lupberger,**[f] **F. Messi,**[b,g] **D. Pfeiffer,**[b,f] **F. Piscitelli,**[b] **T. Blum,**[d] **C. Søgaard,**[d] **S. Skelboe,**[d] **R. Hall-Wilton**[b,e] **and T. Richter**[b]

[a]*European Spallation Source, Data Management and Software Centre,*
Ole Maaløes vej 3, Copenhagen N, 2200 Denmark

[b]*European Spallation Source ERIC,*
P.O. Box 176, Lund, 221 00 Sweden

[c]*School of Physics & Astronomy, Glasgow University,*
Glasgow, Scotland, G12 8QQ United Kingdom

[d]*Niels Bohr Institutet, Blegdamsvej 17, Copenhagen Ø, 2100 Denmark*

[e]*STC Research Centre, Mittuniversitetet,*
Sundvall, 851 70 Sweden

[f]*CERN, Route de Meyrin, Genève, 1211 Switzerland*

[g]*Division of Nuclear Physics, Lund University,*
Professorsgatan 1, Lund, 223 63 Sweden

E-mail: mortenchristensen@esss.se

https://doi.org/10.1088/1748-0221/13/11/T11002

2018 JINST 13 T11002

ABSTRACT: European Spallation Source (ESS) will deliver neutrons at high flux for use in diverse neutron scattering techniques. The neutron source facility and the scientific instruments will be located in Lund, and the Data Management and Software Centre (DMSC), in Copenhagen. A number of detector prototypes are being developed at ESS together with its European in-kind partners, for example: SoNDe, Multi-Grid, Multi-Blade and Gd-GEM. These are all position sensitive detectors but use different techniques for the detection of neutrons. Except for digitization of electronics readout, all neutron data is anticipated to be processed in software. This provides maximum flexibility and adaptability and allows deep inspection of the raw data for commissioning which will reduce the risk of starting up new detector technologies. But it also requires development of high performance software processing pipelines and optimized and scalable processing algorithms. This report provides a description of the ESS system architecture for the neutron data path. Special focus is on the interface between the detectors and DMSC which is based on UDP over Ethernet links. The report also describes the software architecture for detector data processing and the tools we have developed, which have proven very useful for efficient early experimentation, and can be run on a single laptop. Processing requirements for the SoNDe, Multi-Grid, Multi-Blade and Ge-GEM detectors are presented and compared to event processing rates archived so far.

Contents

1 Introduction

The European Spallation Source [1, 2] is a spallation neutron source currently being built in Lund, Sweden. ESS will initially support 15 different instruments for neutron scattering. The ESS Data Management and Software Centre (DMSC), located in Copenhagen, provides infrastructure and computational support for the acquisition, event formation, long term storage, and data reduction and analysis of the experimental data. At the heart of each instrument is a neutron detector and its associated readout system. Currently detectors as well as readout systems are in the design or implementation phase and various detector prototypes have already been produced [3–6]. ESS detectors will operate in event mode [7], meaning that for each detected neutron a (time, pixel) tuple is calculated, providing the detection timestamp (with a resolution of 100 ns or better) and position on the detector where the neutron hit. This allows for later filtering of individual events (vetoing) and flexible refinement of the energy determination as well as of the scattering vector.

ESS detector prototypes have been tested at various neutron facilities and a number of temporary data acquisition systems have been in use so far. When in operation, ESS will use a common readout system which is currently being developed [8]. We are also moving towards a common software platform for the combined activities of data acquisition and event formation. This platform consists of core software functionality common to all detectors and a detector specific plugin architecture.

The main performance indicators of the system are: the neutron rates, the data transport chain from the front-end electronic readout to the event formation system, the parsing requirements for the readout data, and the individual data processing requirements for the different detector technologies.

Good estimates of the neutron flux on the sample and the detectors have been produced by simulations [9–11], and estimates on the corresponding data rates have been made, although the precise values will depend upon engineering design decisions that are still to be made or further detector characterizations. Examples of these are: number of triggered readouts per neutron, readout data encapsulation methods, hardware data processing, etc.

The architecture for the ESS data path is described in section 1.3, and section 2 briefly describes ESS readout architecture and discusses hardware and physical abstractions such as digital and logical geometry. Parsing of readout data and event formation processing is the subject of section 3, which describes four detectors (sections 3.1–3.4), which have been subjected to early testing at neutron sources using a scaled-down version of the anticipated software infrastructure for ESS operations. The software architecture is described in section 4, where the choice and performance of UDP for detector readout is discussed in section 4.1.

Throughput rates for event processing are reported in section 5.

1.1 Instrument data rates

ESS is a spallation source, where neutrons are generated by the collision of high energy protons with a suitable target — tungsten in case of ESS. The proton source is pulsed, with a 14 Hz frequency. The neutron flux generated by this process has been simulated with MCNP from the target/moderator [12–14]. The flux is reduced throughout the neutron path by neutron transport components (neutron guides, monitors, beam ports) and instrument-specific components (collimators, choppers, sample enclosures, etc.) and are typically calculated using the Monte Carlo simulation tool

Table 1. Current estimates of data rates for selected ESS instruments at 5MW ESS source power. Global avg. rate is defined in [10]. Data Rate is the corresponding amount of data received for software processing.

Instrument	Detector	Flux on sample [n/s/cm^2]	Global avg. rate [MHz]	Data Rate [MB/s]
C-SPEC	Multi-Grid	10^8	10	80
ESTIA	Multi-Blade	10^9	500	4000
FREIA	Multi-Blade	$5 \cdot 10^8$	100	800
LoKI[1]	BAND-GEM	$\leq 10^9$	34	272
LoKI	BCS	$\leq 10^9$	37	298
NMX	Gd-GEM	$4.8 \cdot 10^8$	5	300
SKADI	SoNDe	$\leq 10^9$	37	1180
T-REX	Multi-Grid	10^8	10	80

McStas [15, 16]. The detector properties are determined by a combination of Geant4 simulations [17–19] for the initial considerations and experiments at neutron facilities once a prototype has been built.

Typically, rates are reported for neutrons hitting the sample and the detector, as shown in table 1. As mentioned, these rates are not directly convertible into data rates received by the software. Our estimates of the required processing power are, however, based on neutron rates for the detector surface assuming 100% efficiency. On the input side of software processing we use the peak instantaneous rate, measured as the highest number of neutrons received in a 1 ms time bin [10], because we must receive all data without loss. On the output side, we use average rates because event data is buffered up for transmission inside the event formation system which has the practical effect of load levelling the event rate with time.

1.2 State of the art

At most existing neutron facilities, the data acquisition is highly beam line specific, and based around the DAQ PC on the beamline, as the instrument control and DAQ PC, for example [20]. It can not be seen as an integrated system at the facility level. Part of the reason for this is to be able to continue operation of legacy systems. Additionally this is the simplest approach to DAQ.

Starting with ISIS, there have been moves to integrate the DAQ more at the facility level [21, 22]. More recently, the ILL has been investigating moving some of the data reduction and processing into the online data acquisition and control [23]. The SNS started with a DAQ system that used a in-house custom code for instrument control and data acquisition [24]. More recently, the SNS has upgraded most beam lines with a more integrated and standardised DAQ and controls system. The new DAQ is controlled by EPICS directly [25].

One particular feature of ESS instruments is that for the detector systems there is a dramatic increase in channel count. Typical DAQ systems for neutron instruments at existing facilities have

---

[1]The detector technology for LoKI has not been finally decided.
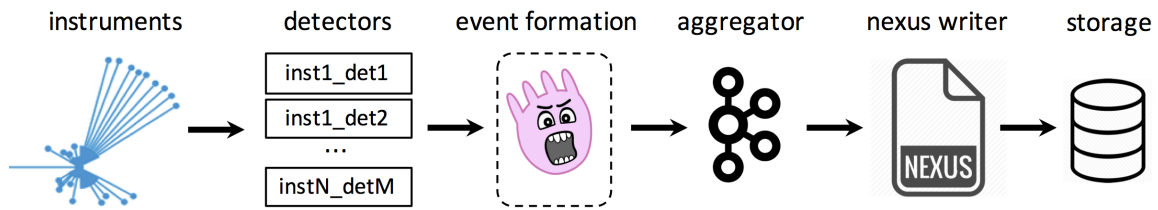
Figure 1. ESS data path.

less than 1000 electronics channels. ESS instruments will in general have > 1000 electronics channels, with some instruments being in the 10 000–100 000 electronics channels range. This means that a fully integrated DAQ system is needed. Additionally, the increase in capability for networking, processing and data handling, means that many operations and algorithms that would only have been possible to do previously either offline, after data storage, or in dedicated fast electronics, can now be done in realtime with sufficiently low latency in a single or small cluster of PCs [26, 27]. The architecture foreseen for the ESS data acquisition, described in this paper, takes advantage of this capability.

### 1.3   Architecture for the ESS data path

The system architecture for the ESS neutron data path is shown in figure 1. Every neutron scattering instrument has at least one detector: the individual detector technologies vary [28], and this is discussed in section 3, but eventually an electric signal is induced on an electrode. This signal is digitised by the readout electronics and sent via UDP to the event formation system.

The key component for the software event formation system is the Event Formation Unit (EFU), a user-space Linux application targeted to run on Intel x86-64 processors written in C++. For each ESS instrument, several EFUs will run in parallel to support the high data rates. The EFU is responsible for processing the digitised readouts and converting these into a stream of event (time, pixel) tuples.

The event tuples are serialised and sent to a scalable data aggregator/streamer providing a publish/subscribe interface. A file writer application subscribes to the neutron data stream and streams from other sources such as motor positions for collimators and sample, temperature, pressure, magnetic/electric fields, etc. This aggregated data is then written to file in a format suitable for long-term storage. From permanent storage it is then possible to perform offline data reduction and analysis [29]. The data streams are not only available to file writing, but will also be used for live data reduction by Mantid [30].

2   Detector readout

The subject of this report is mainly concerned with the detector data flowing from the readout system backend to and through the event formation system. Due to the high neutron flux delivered by ESS, the data rates will be correspondingly high. The ESS readout system conceptually consists of a detector specific front end and a generic backend as illustrated in figure 2.
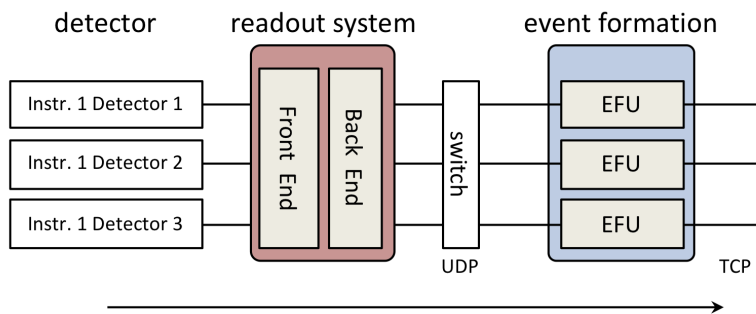
Figure 2. High level architecture of the ESS readout system.

The back end connects to the event formation system via 100 Gb/s optical Ethernet links, which provide more capacity than required for most instruments. However, for the small scale detector prototypes we typically use Gigabit Ethernet.

The ESS readout system is currently under development. Until it becomes generally available a number of different ad-hoc readout systems have been employed for testing of prototypes. The ones relevant for this report are: CAEN, mesytec, RD51 Scalable Readout System and ROSMAP-MP from Integrated Detector Electronics AS [31–34]. These are either controlled by applications supplied by the manufacturer, by custom Python scripts or GUI applications. The digitised data is transmitted as binary data over UDP in a similar way as when the instruments are in operation. None of these ad-hoc systems is currently set up to consume the ESS absolute timing information.
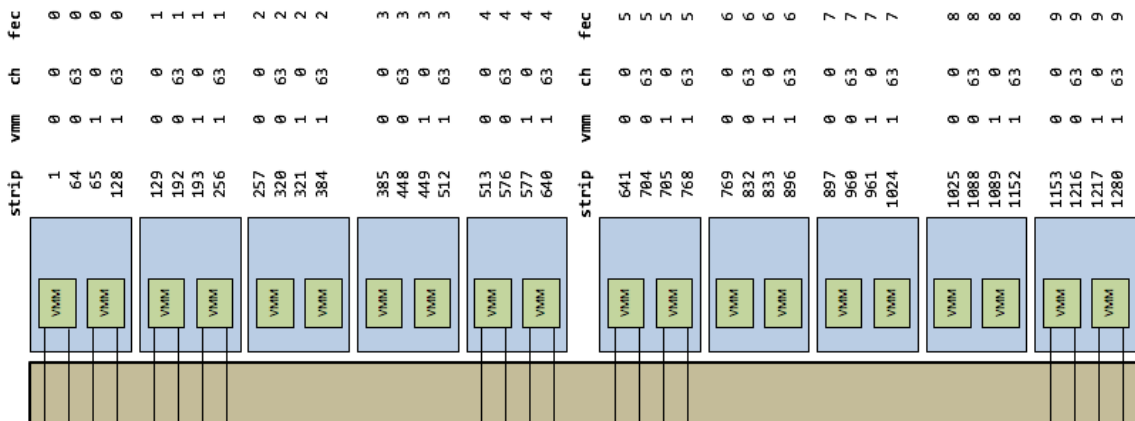
2.1 Digital geometry

While the common readout back-end deals with the connection to the event formation, different detector technologies have different electrical connections to the readout front ends. Multi-Grid for example, uses a combination of wires and grids whereas Multi-Blade uses wires and strips. Even for a specific detector technology, the different prototypes can have different sizes and therefore different number of channels. We need to combine the knowledge about the electrical wiring and how the digitisers are connected in order to know anything about where on the detector a signal was induced. We call this the digital geometry.

An example outlining the digital geometry for Gd-GEM is shown in figure 3. The x-position is a function $x(a, c, f)$ of an *asic id* ranging from 0 to 1, a *channel* from 0 to 63 and a *front end card id* from 0 to 9. For each detector pipeline, a digital geometry C++ class is created to handle this mapping. The classes are typically parametrised so they can handle multiple variants.

2.2 Logical geometry

The main end result from the event formation are event tuples. An event tuple (t, p) consists of a timestamp and a pixel_id. Due to its physical construction, the detectors are inherently pixellated and what we calculate is simply which pixel was hit by a neutron, i.e. this step does not need to know anything about the physical size or absolute compositions of the pixels. We call this the logical geometry, the principle is illustrated in figure 4. We have defined a common convention for the logical geometry for ESS instruments. The convention covers single-panel and multi-panel,

Figure 3. An example of a possible mapping of the digital geometry for x-strips for Gd-GEM. Strip 1 corresponds to asic 0, channel 0, front end card id 0 and strip 1280 to asic 1, channel 64, front end card id 9.

Figure 4. Logical geometry convention for 3D detectors. The example shows a single-panel 3D detector geometry with dimensions nx, ny, nz = (5,4,2), and pixel id's from 1 to 40.

2D and 3D detectors. For example Multi-Grid is a single panel 3D detector (which then has voxels instead of pixels, but we do not make a distinction) and Gd-GEM is a multi-panel 2D detector. In this scheme we also unambiguously define the mapping between the (x, y, z) coordinates of the (logical) positions and a unique number, called the pixel_id.

3 Four ESS detector technologies

Neutrons cannot be directly observed, but are observed as the result of a conversion event where the neutron interacts with a material with a high thermal neutron cross section for absorption. In this process the absorber material converts the neutron into charged particles or light, which can then be detected by conventional methods. For the detectors in this study the conversion materials are based on Li, Be and Gd. The detection methods for the individual detectors will be described below.

3.1 SoNDe

The Solid-state Neutron Detector (SoNDe) is based on a scintillating material that converts thermal neutrons into light which is detected by a photomultiplier tube. The detector is in an early stage of
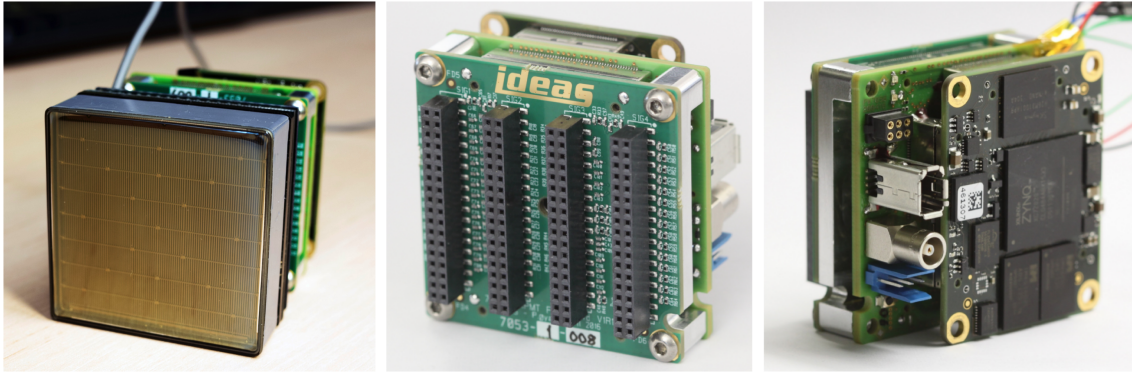
Figure 5. Single SoNDe module with PMT (left), ROSMAP module PMT interconnect side (middle), and digital electronics (right). Photos: European Spallation Source.

characterisation and is currently available as a single module demonstrator [3], shown in figure 5. It consists of a pixelated scintillator, a Hamamatsu H8500 series 8×8 MaPMT (Multi-anode Photomultiplier Tube) [35] and a SONDE/ROSMAP-MP counting chip-system to read out the MaPMT signals [36]. The chip-system consists of four ASICs each responsible for readout of 16 pixels. The final detector will consist of 400 of such modules, arranged in 100 groups of four modules in a 2 by 2 configuration. For a report on the recent progress and patent information, see [37, 38].

The ROSMAP module transmits readout data in three different operation modes as UDP data over Ethernet. The supported modes are Multi-Channel Pulse-Height Data, Single-Channel Pulse-Height Data and Trigger Time Hits over threshold Data. For early characterisation and verification it is necessary to extract the charge information for individual channels and thus support for the two "expert mode" data formats have been developed. When in operation at ESS only the event-mode format (Trigger Time) will be relevant.

3.1.1 Processing requirements

SoNDe belongs to a class of detectors requiring little data processing as the readout system already provides event data in the form of (time, asic_id and channel) values. The digital geometry only has to account for the fact that two of the readout ASICs are rotated 180 degrees compared with the others and the fact that they represent a view of the detector surface from the back, which is different from the logical geometry definition we use. For the single module demonstrator, which consists of 8×8 pixels the processing steps are

- parse the binary readout data and extract (time, asic_id, channel)

- combine asic_id and channel to a pixel_id

3.2 Multi-Grid

The Multi-Grid (MG) detector has been introduced at ILL and developed in a collaboration between ILL, ESS and Linköping University. The detector is based on thin converter films of boron-10

carbide [39, 40] arranged in layers orthogonal to the incoming neutrons. The MG detector uses a stack of grids with a number of wires running through them.

Following the neutron conversion, a signals are induced both on grids and wires, which are digitised and read out. The temporal and spatial coincidence of the signals on wires and grids is used to determine neutron positions. Signals can be induced on multiple grids, and for double neutron events also on wires. The detector geometry is three-dimensional, so our visualisation of the detector image consists of projections of the neutron counts onto the xy-, xz-, and yz-planes respectively as shown in figure 6.

3.2.1 Readout

The Multi-Grid readout system used for prototyping and demonstration detectors is based on stacked MMR readout boards supporting 128 channels, a Mesytec VMMR-8/16 VME receiver card supporting up to 16 readout links, and a SIS3153 Ethernet to VME interface card. It is self-triggered: when the Mesytec hardware registers a signal above a certain trigger-threshold, it triggers a readout of all channels with signals above a second threshold. This readout is then transmitted as UDP packets to the EFU. The binary data format is hierarchical as it supports multiple interface cards, each supporting multiple boards with up to 128 channels.

3.2.2 Processing requirements

The Mesytec UDP protocol has been partially reverse-engineered based on captured network traffic and the available documentation. The protocol parser must be able to support multiple triggers in a single packet, and to discard unused or irrelevant data fields. The data fields consist of 32 bit words each containing a command (8 bits), address/channel (12 bits) and ADC values (12 bits). The channel readouts are given in alternating order $(1, 0, 3, 2, 5, 4, \ldots)$. All channels are assigned a single common 32-bit timestamp, in units of 16 MHz ticks, by the electronics. Thus temporal clustering is performed in hardware, but no continuous global time is currently available.

The EFU then parses the channel readouts, and applies software thresholds. At this stage it discards inconsistent readouts. Channel readouts for Multi-Grid are then mapped to either a grid or a wire id. The current algorithm for the Multi-Grid event formation simply uses the maximum ADC values for grids and wires to determine the position. The processing steps thus consist of

- parse the binary Mesytec readout format to extract time, channel and ADC

- discard inconsistent readouts

- map channel to either grids or wires

- apply suppression thresholds independently for wires and grids

- check for coincidence (must involve both one wire and one grid)

- combine wire_id and grid_id to pixel_id

Figure 6. Grafana dashboard and live detector images from a recent test run with low neutron intensity at the Source Testing Facility at Lund University [41].

3.3 Multi-Blade

The Multi-Blade detector is a stack of Multi Wire Proportional Chambers operated at atmospheric pressure, with a continuous gas flow. It consists of a number of identical units, called cassettes. Each cassette holds a blade (a substrate coated with $^{10}B_4C$) and a two-dimensional readout system, which consists of a plane of wires and a plane of strips. The cassettes are arranged along a circle-arc centered on the sample, and are angled slightly with respect to the neutron beam, for improved counting rate capability and spatial resolution. The operation is based on the temporal and spatial coincidence of signals on strips and wires. Despite inherently being a three-dimensional detector, the visualisations of the detector images typically display an "unfolded" two-dimensional pixel map. For further details of the design and performance of this detector see [42–44].

3.3.1 Readout

The Multi-Blade detector prototype currently has nine cassettes, each with 32 wires and 32 strips, for a total of 576 channels. The readout is based on six CAEN V1740D digitisers, and a custom readout application based on the API and software libraries supplied by CAEN. The digitisers each have 64 channels, 32 for wires and 32 for strips. The wires and strips are connected to the digitiser via front-end electronics boards. The final detector will have 32 wires and 64 strips per cassette, and up to 50 cassettes for a total of 4800 channels.

When the CAEN readout system has detected signal above a certain (hardware) threshold it triggers an individual readout of that channel. The readout consists of a channel number, a pulse integral (QDC), a time-stamp, and digitiser id. For each trigger there will be one or more signals from both wires and strips. The readout application continuously reads from the CAEN digitiser's hardware registers using optical links and transmits the raw data over UDP to the event formation unit.

3.3.2 Processing requirements

Readouts are subject to clustering analysis, where they are matched in both time and amplitude. The maximum timespan for which channels can be said to belong to the same cluster is a configurable parameter of the algorithm. For coincidence building there can be up to 2 wires and 4 strips in a cluster, where the typical case is one wire and two strips. Following clustering, we then calculate the pixel where the neutron was detected and adds a timestamp. To summarise, the processing steps for Multi-Blade are:

- parse the UDP readout format to extract time, digitiser, channel and QDC values

- collect readouts in clusters

- map channel ids to either strips or wires

- check for coincidence (time and amplitude)

- combine wire_id and strip_id to pixel_id

It is possible to improve the spatial resolution by employing CoG (center of gravity) on strip readouts weighted by the deposited charge (QDC).

The measured amplitudes on the wires and on the strips are strongly correlated. This means that with sufficient dynamic range double neutron events, which would cause some ambiguity, might be resolved by requiring matching amplitudes [42].

The processing pipeline for Multi-Blade currently differs from the other detectors in that the code responsible for clustering and event formation runs in multiple incarnations, namely one for each cassette. This is a case where we explore the solution space for event processing. The approach has the advantage of supporting individual processing for each blade rather than having to explicitly maintain information about blade id's in the processing algorithm itself.

3.4 Gd-GEM

The NMX macromolecular diffraction instrument will use the Gd-GEM detector technology. The neutron converter is a 25 µm thin foil of gadolinium, which also serves as cathode in a gas volume (Ar/CO_2 70/30 at athmospheric pressure). After traversing the readout and the GEM foils, the neutron hits the converter where it is captured as shown in figure 7. After the neutron capture, gamma particles and conversion electrons are released into the gas volume.

The conversion electrons loose energy by ionizing the gas atoms, and create secondary electrons along their path. Due to an electric field, those secondary electrons are drifted away form the cathode to an amplification stage consisting of a stack of two or three GEM foils. Each electron generates a measurable amount of charge by an avalanche in the GEM holes, which induces a signal on a segmented anode.

This segmentation is realised by copper strips with a pitch of 400 µm. The signal on the strips is read out with a timing resolution in the order of 10 ns, such that projections of the tracks in the x–t and y–t plane can be used to combine hits in both planes (clustering) and reconstruct the neutron impact point (micro-TPC method) [45].
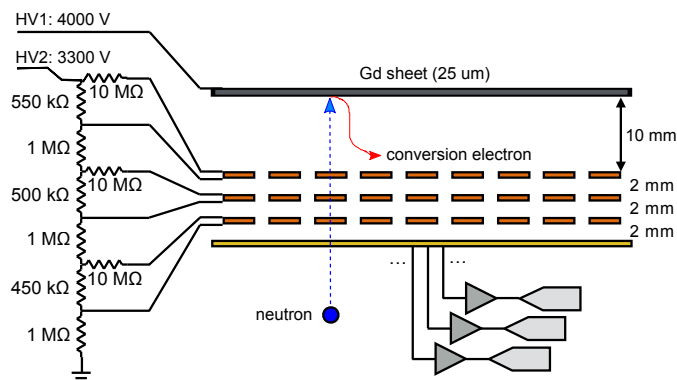
Figure 7. Schematic drawing of the Gd-GEM detector in backwards mode. From [6].
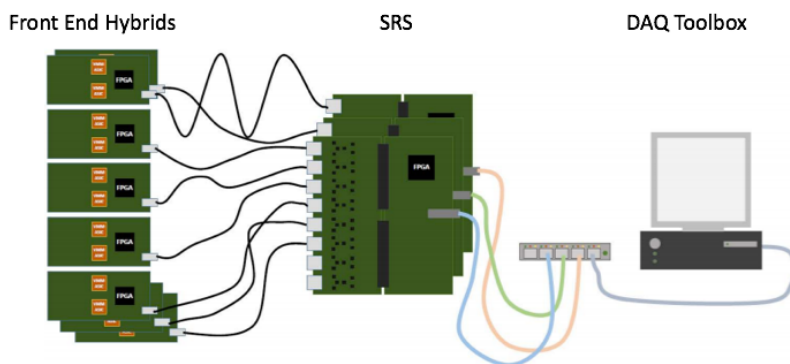
Figure 8. Figure of the Gd-GEM readout and data acquisition system, from [47].

3.4.1 Readout

The analogue signals of the strips of the Gd-GEM detector are read out by the VMM ASIC developed by Brookhaven National Laboratory for the New Small Wheel Phase 1 upgrade [46]. The VMM has been implemented in the SRS [47] at CERN and a schematic drawing of the readout chain is shown in figure 8. The so called front-end hybrids are directly mounted onto the detector. This PCB holds two VMM ASICs, each with 64 input channels connected to the anode strips with a spark protection circuit. For each hit strip where the signal surpasses a configurable threshold, the VMM outputs a 38 bit binary word, see table 2.

For the prototype a Spartan-6 FPGA on the hybrid controls the ASICs and bundles the data, that are transmitted via HDMI cables to the core of SRS, the Front-End Concentrator (FEC) card. Up to eight hybrids can currently be connected to one FEC and the data are encapsulated into UDP packages of a 1 Gb/s Ethernet connection to the readout computer [48].

The readout of the Gd-GEM detector is partitioned into 4 sectors. Each of these 4 sectors has 640 strips read out by 5 hybrids in x and y direction, resulting in a total of 5120 strips and 40 hybrids. If a signal is recorded on a detector strip, the VMM on the hybrid generates hit data for the corresponding channel. Before sending out the data via UDP, the FEC adds the VMM ID and the FEC ID to the hit data.

Table 2. Data fields from VMM3.

field	size (bits)	field	size (bits)	field	size (bits)
flag	1	threshold	1	channel	6
amplitude	10	time	8	BCID	12

With the information tuple (channel, VMM ID, FEC ID), the geometrical position of each hit can be reconstructed. A configuration file that reflects this digital geometry of the detector is loaded during the start up phase of the DAQ. The configuration can be modified for reordering, exchange or extension of physical readout components.

3.4.2 Processing requirements

The Gd-GEM detector requires the most complicated processing requirements in terms of the physical processes, the data acquisition and processing power. The steps required are

- parse the binary data from SRS readout and extract (time, channel, adc)-tuples

- queue up (time, channel, adc)-tuples until enough data for attempting clustering analysis

- perform clustering analysis — determine if coincidence occurred

- calculate neutron entry position for x and y

- convert positions to pixel_id

Some of the software related challenges for Gd-GEM are: scaling up to a full rate, detector size and for discriminating invalid tracks. A neutron event generates a track with extensions in both time and space so it is not possible to just partition the detector in regions for independent parallel processing. Several processing options for distinguishing which tracks from which position can be extracted have been described in [49]. In addition, due to the required buffering of data, memory usage and cache performance may well be a concern.

4 Event Formation Unit architecture

The EFU architecture, illustrated in figure 9, consists of a main application with common functionality for all detectors and detector-specific processing pipelines. The software is written in C++, and is built using gcc and clang compilers for Ubuntu, macOS and CentOS. CentOS is currently the target Linux distribution for ESS operations, whereas the other operating systems are used during development and implementation.

The main application handles low CPU-intensity tasks such as launch-time configuration via command-line options, run-time configuration using a TCP-based command API, application state logging and periodic reporting of run time statistics and counters.

Detector pipelines are responsible for handling realtime readout data, and must conform to a common software interface definition. The pipelines are implemented as shared libraries that are loaded and launched by the main application as POSIX threads with support for thread affinity,
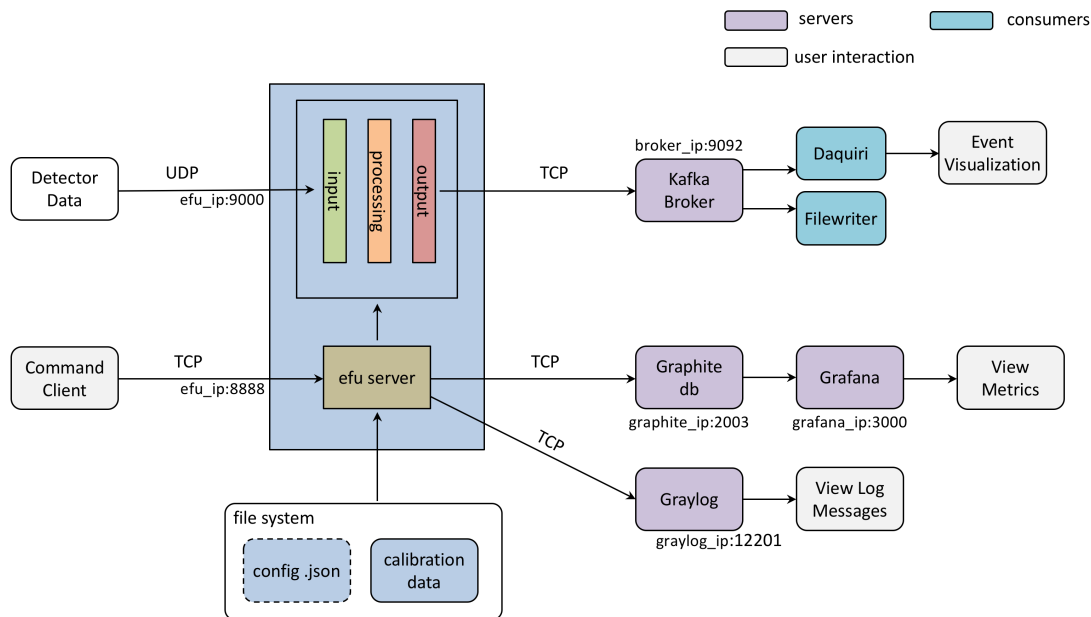
Figure 9. Event Formation Unit (EFU) architecture.

which fixes a thread onto a specific processor core. The plugin must specify at least one processing thread but apart from this no further restrictions are imposed. We have experimented with different configurations for different detectors, but currently the number of threads in a detector pipeline ranges from one to three.

When more than one thread is in use, the data is shared between the producer and the consumer thread by a circular data buffer (FIFO) which preserves the order of the arriving data. The FIFO is based on pre-allocated memory to avoid unnecessary data copying and C++ `std::atomic` primitives for resource locking. For performance benchmarking we use the `rdtsc()` instruction call, which gives a high resolution timestamp counter with low latency.

The data processing part of the detector pipelines generally consists of a tight loop with a BSD socket `recvfrom()` system call, a `parse()` function, and a `produce()` step. These processing steps can be done in a single or multiple threads, depending on specific requirements.

4.1 UDP data input

At ESS we have chosen User Datagram Protocol (UDP) for the transmission of readout data over Ethernet. Other contenders were Transmission Control Protocol (TCP), 'raw' Ethernet frames and (briefly) InfiniBand. UDP is a simple protocol for connectionless data transmission [50] without a guaranteed delivery. The alternative, TCP, guarantees the ordered delivery of data and automatically adjusts transmission to match the capability of the transmission link [51].

Despite its inherent unreliability, UDP is widely used. For example in the RD51 Scalable Readout System [33], or the CMS trigger readout [52], both using 1 Gb/s Ethernet. The ESS readout system described in section 2 is based on FPGAs with support for transmission of Ethernet packets over 100 G. Implementing TCP on these are not an option, so UDP was eventually decided.

The unreliability of UDP is widely overestimated. It is true that a UDP packet can potentially be dropped in any part of the communications chain: the sender, the receiver, intermediate systems such as routers, firewalls, switches, load balancers, etc. This makes it difficult in the general case to rely on UDP for high speed communications. However high reliability solutions can be engineered for simple network topologies such as the one anticipated for the ESS readout shown in figure 2. This is a simple network topology, where both the FPGA and the switch can deliver packets at wire rate. Thus no packet loss will occur except at the receiver.

At the receiver, two types of data loss are the main causes of performance degradation: buffer exhaustion and packet processing overhead. These are not independent as increased time spent in processing data will increase the likelihood that the receive buffers fill up. Nevertheless the effects can be reduced as we describe in the following two sections.

4.1.1 Receive buffers

The main parameters for controlling socket buffers are `rmem_max` and `wmem_max`. The former is the size of the UDP socket receive buffer, whereas the latter is the size of the UDP socket transmit buffer. To change these values from a BSD socket application use `setsockopt()`, for example

```
int buffer = 4000000;
setsockopt(s, SOL_SOCKET, SO_SNDBUF, buffer, sizeof(buffer));
setsockopt(s, SOL_SOCKET, SO_RCVBUF, buffer, sizeof(buffer));
```

In addition there is an internal queue for packet reception whose size (in packets) is named `netdev_max_backlog`, and a network interface parameter, `txqueuelen` which were also adjusted for our benchmark testing.

The default value of these parameters on Linux are not optimised for high speed data links such as 10 Gb/s Ethernet, so for the measurements presented here the following parameters were used:

```
net.core.rmem_max=12582912
net.core.wmem_max=12582912
net.core.netdev_max_backlog=5000
txqueuelen 10000
```

These values were generally adopted from previous studies [53] and guides [54].

4.1.2 Packet sizes

Packets arriving at a data acquisition system are subject to a nearly constant per-packet processing overhead. This is due to interrupt handling, context switching, checksum validations and header processing. There is an intimate relation between Ethernet link speed, packet size, packet rates and header overhead as shown in table 3. For 10 G Ethernet at up to 15 M packets per second, this processing alone can consume most of the available CPU resources.

In order to achieve maximum performance, data from the electronics readout should be bundled into jumbo frames if at all possible. Using the maximum Ethernet packet size of 9018 bytes reduces the per-packet overhead by up to a factor of 100. This does, however, come at the cost of larger latency. For example the transmission time of 64 bytes is 67 ns, whereas for 9018 it is 7230 ns. For

Table 3. Packet rates as function of packet sizes for 10 Gb/s Ethernet.

User data size [B]	1	18	82	210	466	978	1472	8972
Packet size [B]	64	64	128	256	512	1024	1518	9018
Overhead [%]	98.8	78.6	44.6	23.9	12.4	5.5	4.3	0.7
Frame rate [Mpps]	14.88	14.88	8.45	4.53	2.35	1.20	0.81	0.14

applications sensitive to latency a tradeoff must be made between low packet rates and low latency. For ESS latency is not an issue as all readout is time-stamped by hardware before transmission.

By bundling readout data into Ethernet Jumbo frames of size 9018 bytes, rather than using small Ethernet frames of size 64 bytes, the packet rate is reduced by a factor of 100 as shown in table 3. This is directly measurable as a reduced time spent in system calls and reduced number of context switching between user space and kernel.

We configured the systems with an MTU of 9000 bytes allowing user payloads up to 8972 bytes when taking into account that IP and UDP headers are also transmitted. Given the efficiency gained by using large packets, there was no need to consider InfiniBand or raw Ethernet to reduce the size of protocol headers, thus confirming the choice of UDP.

4.1.3 Performance

For an early validation of the use of UDP we ran a series of performance tests using an experimental configuration described in appendix A. The testbed consists of two hosts, one acting as a UDP data generator and the other as a UDP receiver. As mentioned the readout transmitter at ESS will be based on FPGA and not a Linux server. However this does not affect the results we measured for the receiver.

The measured performance, shown in figure 10, covers user data speed, packet error ratios and CPU load. They are time averaged over 10 second intervals while transmitting 400 GB of data at a time. There is a clear variation with packet size for all parameters with the best results obtained with packet sizes larger than 2200 bytes. The best results in terms of CPU bandwidth is obtained with a packet size of 9000 bytes.

The tests made use of packet sequence numbers allowing the determination of packet error rates (PER). Sequence numbers are not supported in the current prototype readout systems. Thus packet loss and PER numbers are not available for these. Sequence numbers will, however, be part of the ESS readout system.

The main conclusion is that with only a single cpu core used as the receiver it is possible to support 10 Gbit/s with zero packet loss using UDP. The actual performance when real data is applied will naturally change, and will need to be followed closely.

4.2 FlatBuffers/Kafka

The output of the EFU is a stream of events. We have chosen Apache Kafka [55] as the central technology for transmission, and Google FlatBuffers [56] for serialisation. Apache Kafka is an open source software project for distributed data streaming. Multiple Kafka brokers form a scalable cluster, which supports a publish-subscribe message queue pattern with configurable data persistence.
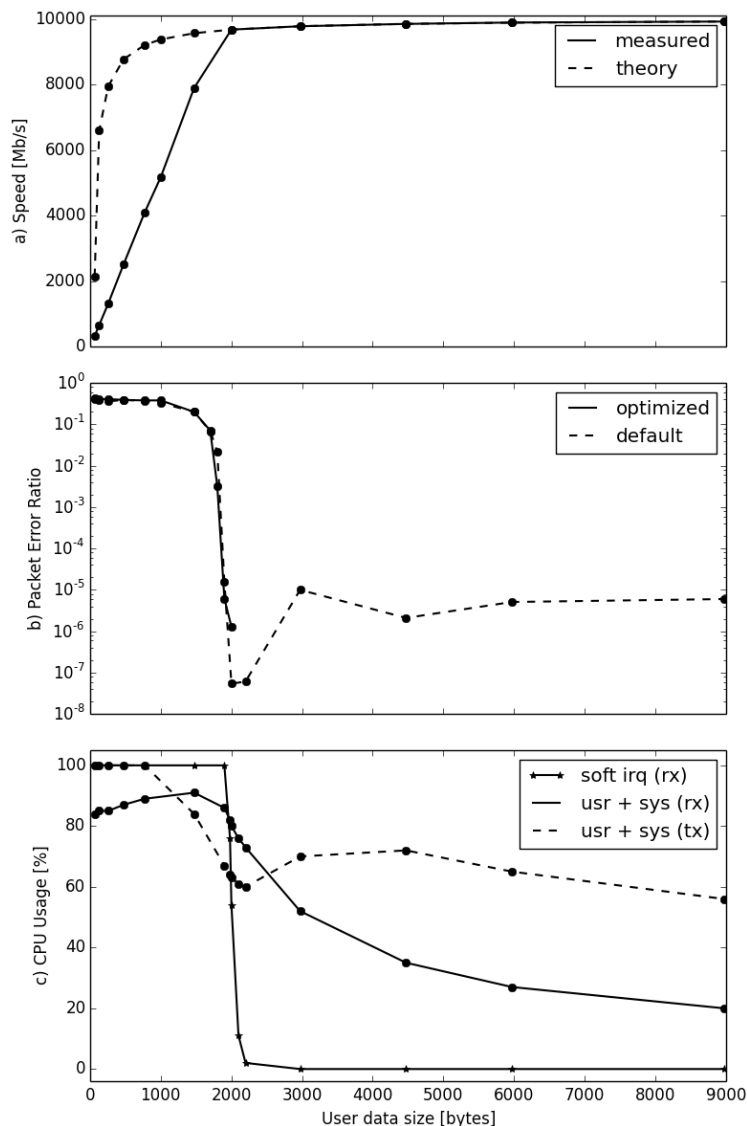
Figure 10. Performance measurements. a) User data speed. b) Packet Error Ratio. c) CPU Load. Note that for the optimized values PER is zero for user data larger than or equal to 2200 bytes (solid line).

In Kafka, producers send data to a topic in a cluster. A consumer subscribes to a topic to receive messages, either from the instant the subscription starts or from a previous offset, provided the requested data is still available in storage on the cluster, given a retention policy. Consumers may also be grouped to distribute the processing load among different processes.

Both producers and consumers can be developed using open source Kafka client libraries. The EFU uses `librdkafka` [57], which offers a C/C++ API. While Kafka offers a scalable and reliable transmission of arbitrary data, FlatBuffers provides a schema-based event serialisation method, and a mechanism for forward and backward compatibility of the schemas. Figure 11 shows the currently used schema for events.

```
include "is84_isis_events.fbs";

file_identifier "ev42";

union FacilityData { ISISData }

table EventMessage {
    source_name : string;
    message_id : ulong;
    pulse_time : ulong;
    time_of_flight : [uint];
    detector_id : [uint];
    facility_specific_data : FacilityData;
}
root_type EventMessage;
```

Figure 11. The ESS FlatBuffers event schema. The main fields of relevance are the arrays `time_of_flight` and `detector_id` (from https://github.com/ess-dmsc/streaming-data-types).

4.3 Live detector data visualisation

After writing the first prototype for event formation, it became clear that it would be beneficial to also use the EFU as a DAQ system for early detector experiments and commissioning. One of the easiest ways to validate the event processing is to visualise the detector image and other relevant data, such as channel intensities and ADC distributions.

For this reason the EFU also publishes such information via Kafka, and an application named Daquiri was written for visualising these data. Daquiri subscribes to Kafka topics, collects statistics, and provides plotting functionality, and is planned to be an integral part of the software bundle developed for ESS operations. The Daquiri GUI is based on Qt [58] and is highly configurable in terms of available plotting formats, the dashboard configuration, labels, axes, colour schemes, etc. A typical screenshot is depicted in figure 12. Daquiri is open source software [59].

4.4 Runtime stats and counters

The availability of relevant application and data metrics is essential for both early prototyping and easy monitoring while in operation, for example incoming packet rates, parsing errors, calculated events, discarded readouts, etc. The detector API provides a mechanism for the detector plugin to register a number of named 64-bit counters. These are then periodically queried by the main application and reported to a time-series server.

We have chosen Graphite as the time-series server [60] technology and use Grafana [61] for presentation. Graphite has a simple API for submission of data, which consists of a hierarchical name such as *efu.net.udp_rx*, a counter value and a UNIX timestamp. The combination Grafana/Graphite has proven to be very useful, not only for monitoring the event processing software. Scripts have been written to check Linux kernel and network card counters, as well as disk and CPU usage, all of
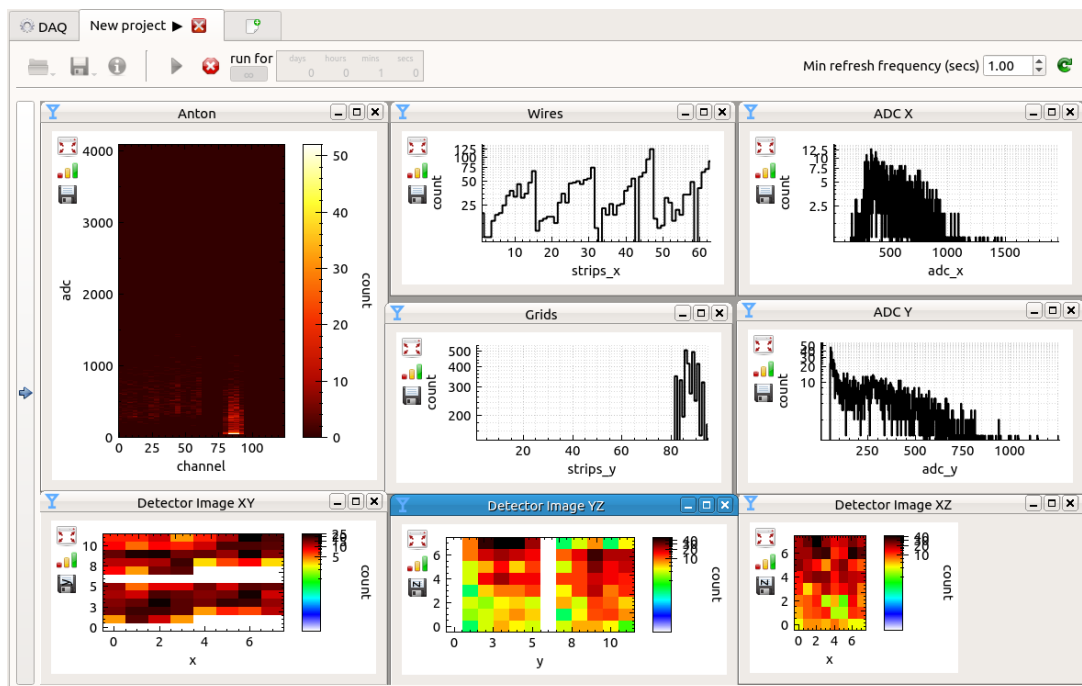
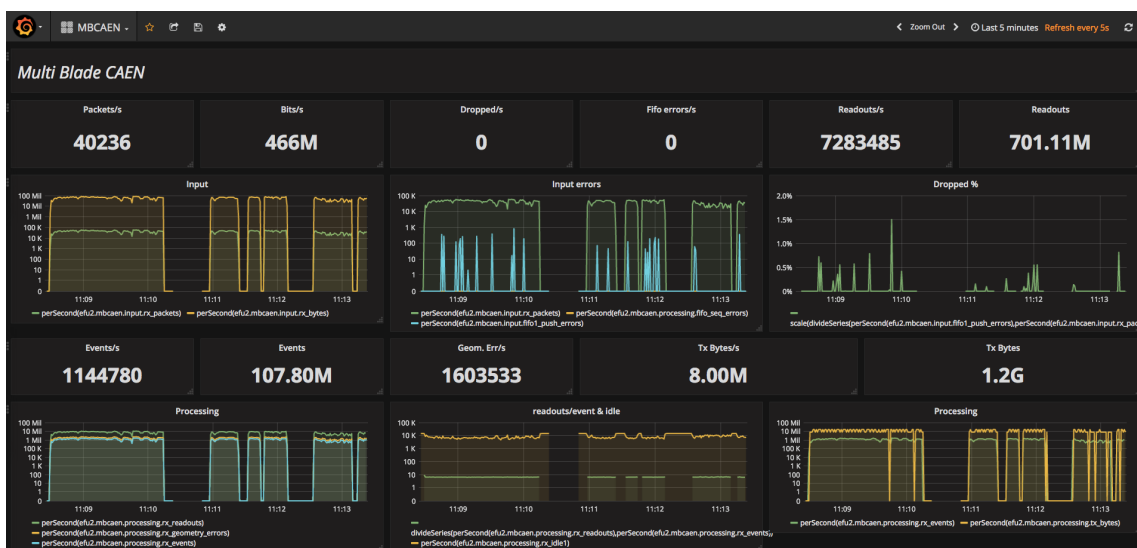**Figure 12**. The Daquiri commissioning tool.

Figure 13. Grafana dashboard used for monitoring packet and event related counters for an implementation of Multi-Blade data processing.

which are relevant when running at high data rates while simultaneously writing raw data to disk. We typically publish monotonically increasing counter values, and then use Grafana to transform these into rates. We plan to offer Grafana/Graphite for the software we are developing for ESS operations. Figure 13 shows how this currently looks like.

4.5 Trace and logging

For application logging we have chosen Graylog [62]. In the EFU, Graylog is used for low rate log messages. We use the `syslog` [63] conventions for logging levels and severities. Graylog is not currently in wide use in the infrastructure, but will be essential for monitoring the ESS data processing chain once in operations, when multiple EFUs are deployed.

During development we use a simple but effective trace system consisting of groups and masks. These currently print directly to the console, which is extremely detrimental to performance when operating at thousands of packets and millions of readouts per second. Therefore, we made the trace macros configurable at compile time, so that no overhead occurs when they are not needed. Both the log and trace system accept log messages in a `printf()`-compatible format using variable arguments.

4.6 Software development infrastructure

All the software components that are part of the data aggregation and streaming pipeline are being developed collaboratively by the partners as open source projects released under a BSD license. Git [64] is used for version control and all software is available for public scrutiny on GitHub [65]. We use Conan [66] as our C++ package manager and CMake [67] for multi-platform Makefile generation. The projects are built with gcc and clang compilers [68, 69].

A Jenkins [70] build server automatically triggers builds and runs commit stage tests each time new code is pushed to a repository: every commit on every branch for every project triggers a Jenkins build and test cycle on multiple operating systems providing rapid feedback on breaking changes. The tests that are run vary according to the application but for C++ code in general include unit tests with Google Test [71], static analysis with Cppcheck [72], and test coverage reports with gcovr [73]. We also check for code format compliance with clang-format [74], and memory management problems with Valgrind [75].

Individual methods and algorithms can be benchmarked for performance with Google Benchmark [76]. The executables generated from every successful build cycle are saved as artefacts and can thus be used for quick deployment or integration testing. Configuration of the machines in the build and test environment is done using Ansible [77], with the scripts kept under version control.

5 Event processing rates

The key metric we use for the evaluation of performance is the number of events the detector pipeline can process per second. To benchmark this, we use detector data recorded as Ethernet/UDP packets in an number of measurement campaigns. This data is then sent to the event formation system as fast as possible and the achieved rates are retrieved via Grafana.

The setup uses three servers: a macOS laptop acting as a data generator/detector readout, a Ubuntu workstation hosting the EFU and Kafka, and another Ubuntu workstation which hosts Graphite and Grafana metrics. The hardware specifications are listed in table 4. The tests were made on the latest event formation software [78]. For Gd-GEM we have implemented a performance test based on Google Benchmark, which directly targets the event processing algorithm, and is likely to present an upper bound for the event rates in a single processing thread as there is no other overhead involved.

Table 4. Machine configurations for the performance test setup.

Machine	OS	CPU	RAM
detector	macOS 10.13.3	Intel Core i7 @ 2.2 GHz	16 GB, DDR3, 1600 MHz
efu	Ubuntu 16.04	Intel Xeon E5-2620 v3 @ 2.40 GHz	64 GB, DDR4, 2133 MHz
metrics	Ubuntu 16.04	Intel Xeon E5-2620 v3 @ 2.40 GHz	64 GB, DDR4, 2133 MHz
benchmarks	Ubuntu 16.04	Intel Core i7-6700 @ 3.40 GHz	32 GB, DDR4, 2133 MHz

Table 5. Measured performance for detector pipelines.

Detector	Machine	Packet Rate [pkt/s]	Trigger Rate [readouts/s]	Event Rate [events/s]	Cores	Packet size [bytes]
Gd-GEM	benchmarks	n/a	18.6 M	500 k–1 M	1	n/a
Multi-Grid	efu	86.000	3.0 M	2.46 M	1	1137
Multi-Blade	efu	82.000	5.6 M	2.31 M	2	1494
SoNDe	efu	94.000	23.5 M	23.5 M	2	1307

Table 5 summarises the results of the performance measurements. It shows that a pipeline can support the reception and processing of around 85.000 UDP packets per second and several millions readouts per second using one or two CPU cores. The reported event rates reflect the amount of computational work that has to be performed on the data: Gd-GEM has the most complex algorithm, Multi-Grid and Multi-Blade have medium complexity, and SoNDe requires the least processing.

The large uncertainty for Gd-GEM comes from the fact that for this detector technology neutron events gives rise to a range of readouts of up to 20 strip hits for both x- and y- strips. None of the readout systems supported Jumbo frames at the time of these experiments, so the additional benefit of using large packets is not reflected in the reported rates.

Taking into account that a mid-range server can have two CPU sockets, each having 8 cores/16 hyper threads we can naively scale these numbers to the very high event rates required at ESS by parallelisation. For example, by employing a small number (5–10) of servers, each dedicated to processing data from a fraction of the detector surface, we expect to scale the rates by more than an order of magnitude.

6 Conclusion

The previous sections have given an overview of the ESS software architecture for event processing in general and as implemented in four detector designs specifically. We have discussed the technology choices made and the toolchain used for software development. Finally we presented recent event processing rates for four detectors which will be used in ESS instruments.

We have shown an architecture that can be scaled up to deal with the high neutron rate ESS will deliver. Without having spent much time on optimisation of the code so far we have achieved high event processing rates of the order of 1 to 25 M events per second, and have shown how this can be scaled to much higher event rates using commodity hardware. The detectors discussed in this paper represent the range of the expected processing requirements foreseen at ESS, from simple to quite
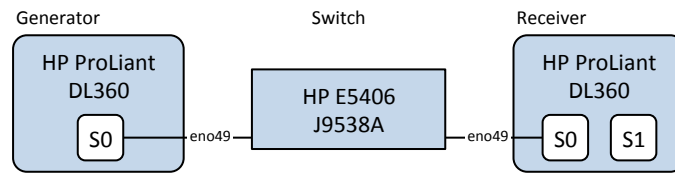
Figure 14. Experimental setup.

complex. The toolchain does not wait to become operational after 2021 where ESS is expecting see first beam on target, but is in actual use for data acquisition as the detector development continues.

Most detectors are constructed by the tiling of identical and independent units. Scaling the processing up for these are easy as we can employ multiple event formation units running in parallel.

Not all scalability problems have been solved yet, however. Future work will focus on scaling the Gd-GEM processing as it is markedly more complicated than the other detectors. For example a simple partitioning of the detector surface may not be feasible, because the charge tracks from a single neutron conversion can easily cross partition borders. Collaboration on this topic has already started. Work will also be done on deploying multiple processing pipelines on a multi-core CPU, where typically resource sharing problems, such as memory and network bottlenecks, will become more pronounced than observed so far.

Acknowledgments

A UDP performance testbed

For the UDP performance testing we used a two server setup shown in figure 14. The hosts are HPE ProLiant DL360 Gen9 servers connected to a 10 Gb/s Ethernet switch using short (2 m) single mode fibre cables. The switch is a HP E5406 switch equipped with a J9538A 8-port SFP+ module. The server specifications are shown in table 6. Except for processor internals the servers are equipped with identical hardware.

The data generator is a small C++ program using BSD socket, specifically the `sendto()` system call for transmission of UDP data. The data receiver is based on the EFU whose architecture is described The system, named the Event Formation Unit (EFU), supports loadable processing pipelines. A special UDP 'instrument' pipeline was created for the purpose of these tests. Both the generator and receiver uses `setsockopt()` to adjust transmit and receive buffer sizes. Sequence

– 21 –

Table 6. Hardware components for the testbed.

Motherboard	HPE ProLiant DL360 Gen9
Processor type (receiver)	Two 10-core Intel Xeon E5-2650v3 CPU @ 2.30 GHz
Processor type (generator)	One 6-core Intel Xeon E5-2620v3 CPU @ 2.40 GHz
RAM	64 GB (DDR4) — 4 × 16 GB DIMM — 2133 MHz
NIC	dual port Broadcom NetXtreme II BCM57810 10 Gigabit Ethernet
Hard Disk	Internal SSD drive (120 GB) for local installation of CentOS 7.1.1503
Linux kernel	3.10.0-229.7.2.el7.x86_64

numbers are embedded in the user payload by the transmitter allowing the receiver to detect packet loss and hence to calculate packet error ratios. Both the transmitting and receiving applications were locked to a specific processor core using the `taskset` command and `pthread_setaffinity_np()` function. The measured user payload data-rates were calculated using a combination of fast timestamp counters and microsecond counters from the C++ chrono class. Care was taken not to run other programs that might adversely affect the experiments. CPU usages were calculated from the `/proc/stat` pseudofile as also used in [53].

B Source code

The software for this project is released under a BSD license and is freely available on GitHub [78]. To build the exact versions of the programs used for the UDP performance experiments, complete the steps below. To build and start the producer:

```
> git clone https://github.com/ess-dmsc/event-formation-unit
> git checkout 547b3e9
> cd event-formation-unit/udp
> make
> taskset -c coreid ./udptx -i ipaddress
```

to build and start the receiver:

```
> git clone https://github.com/ess-dmsc/event-formation-unit
> git checkout 547b3e9
> mkdir build
> cd build
> cmake ..
> make
> ./efu2 -d udp -c coreid
```

References

[1] *European Spallation Source ERIC*, (2018) https://europeanspallationsource.se/.

[2] S. Peggs et al., *ESS Technical Design Report*, European Spallation Source ESS AB, Lund Sweden (2013) [ESS-2013-001] [ISBN: 978–91–980173–2–8] and online pdf version at https://europeanspallationsource.se/sites/default/files/downloads/2017/09/TDR_online_ver_all.pdf.

[3] S. Jaksch et al., *Cumulative Reports of the SoNDe Project July 2017*, `arXiv:1707.08679`.

[4] M. Anastasopoulos et al., *Multi-Grid Detector for Neutron Spectroscopy: Results Obtained on Time-of-Flight Spectrometer CNCS*, 2017 *JINST* **12** P04030 [`arXiv:1703.03626`].

[5] F. Piscitelli et al., *The Multi-Blade Boron-10-based Neutron Detector for high intensity Neutron Reflectometry at ESS*, 2017 *JINST* **12** P03013 [`arXiv:1701.07623`].

[6] D. Pfeiffer et al., *First Measurements with New High-Resolution Gadolinium-GEM Neutron Detectors*, 2016 *JINST* **11** P05011 [`arXiv:1510.02365`].

[7] T. Gahl et al., *Hardware Aspects, Modularity and Integration of an Event Mode Data Acquisition and Instrument Control for the European Spallation Source (ESS)*, `arXiv:1507.01838`.

[8] S. Kolya et al., *4.1: Integration plan for detector readout*, BrightnESS (2015).

[9] K. Kanaki et al., *Simulation tools for detector and instrument design*, in press [*Physica B* (2018)].

[10] I. Stefanescu et al., *Neutron detectors for the ESS diffractometers*, 2017 *JINST* **12** P01019 [`arXiv:1607.02324`].

[11] K. Kanaki et al., *Detector rates for the Small Angle Neutron Scattering instruments at the European Spallation Source*, 2018 *JINST* **13** P07016 [`arXiv:1805.12334`].

[12] C.J. Werner et al., *MCNP Version 6.2. Release Notes*, LA-UR-18-20808, Los Alamos National Laboratory (2018).

[13] J. Armstrong et al., *MCNP® User's Manual. Code Version 6.2*, LA-UR-17-29981, C.J. Werner ed., Los Alamos National Laboratory (2017).

[14] L. Zanini, *The neutron moderators for the European Spallation Source*, *J. Phys. Conf. Ser.* **1021** (2018) 012066.

[15] K. Lefmann and K. Nielsen, *McStas, a General Software Package for Neutron Ray-tracing Simulations*, *Neutron News* **10** (1999) 20.

[16] P. Willendrup, E. Farhi, E. Knudsen, U. Filges and K. Lefmann, *McStas: past, present and future*, *J. Neutron Res.* **17** (2014) 35.

[17] GEANT4 collaboration, S. Agostinelli et al., *GEANT4: A Simulation toolkit*, *Nucl. Instrum. Meth.* **A 506** (2003) 250.

[18] J. Allison et al., *Geant4 developments and applications*, *IEEE Trans. Nucl. Sci.* **53** (2006) 270.

[19] J. Allison et al., *Recent developments in Geant4*, *Nucl. Instrum. Meth.* **A 835** (2016) 186.

[20] S.A. Kulikov and V.I. Prikhodko, *New generation of data acquisition and data storage systems of the IBR-2 reactor spectrometers complex*, *Phys. Part. Nucl.* **47** (2016) 702.

[21] W.C.A. Pulford, *Future Strategy for Computing at ISIS*, in proceedings of the *ICANS-Xl International collaboration on Advanced Neutron Sources*, KEK, Tsukuba, Japan, 22–26 October 1990.

[22] F.A. Akeroyd, S.I. Campbell and C.M. Moreton-Smith, *The New ISIS Instrument Control System*, in proceedings of the *NOBUGS 2002 Conference: New Opportunities for Better User Group Software*, Gaithersburg, Maryland, U.S.A., 4–6 November 2002.

[23] P. Mutti et al., *Real-Time Data Reduction Integrated into Instrument Control Software (2015)*, in proceedings of the *15th International Conference on Accelerator and Large Experimental Physics Control Systems*, Melbourne, Australia, 17–23 October 2015, https://doi.org/10.18429/JACoW-ICALEPCS2015-THHB3O02.

[24] H. Kleines et al., *Design of the Control and Data Acquisition System of the Neutron Spin Echo Spectrometer at the Spallation Neutron Source*, in proceedings of the *2007 International Conference on Accelerator and Large Experimental Control Systems (ICALEPCS 2007)*, Knoxville, Tennessee, U.S.A., 15–19 October 2007.

[25] S.M. Hartman, *SNS Instrument Data Acquisition and Controls*, in proceedings of the *14th International Conference on Accelerator & Large Experimental Physics Control Systems (ICALEPCS 2013)*, San Francisco, CA, U.S.A., 6–11 October 2013.

[26] G. Bauer et al., *The data-acquisition system of the CMS experiment at the LHC*, *J. Phys. Conf. Ser.* **331** (2011) 022021.

[27] C. Youngman et al., *The design and performance of the ZEUS global tracking trigger*, *Nucl. Instrum. Meth.* **A 580** (2007) 1257.

[28] O. Kirstein et al., *Neutron Position Sensitive Detectors for the ESS*, `PoS(Vertex2014)029` (2014) [`arXiv:1411.6194`].

[29] A.H.C. Mukai et al., *Status of the development of the experiment data acquisition pipeline for the European Spallation Source*, in proceedings of the *16th International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS 2017)*, Barcelona, Spain, 8–13 October 2017, JACoW Publishing (2017).

[30] J. Taylor et al., *Mantid: Manipulation and Analysis Toolkit for Instrument Data*, (2013) http://doi.org/doi:10.5286/software/mantid.

[31] *CAEN*, (2018) http://www.caen.it/.

[32] *mesytec*, (2018) https://www.mesytec.com/.

[33] S. Martoiu, H. Muller and J. Toledo, *Front-end electronics for the Scalable Readout System of RD51*, in proceedings of the *IEEE Nuclear Science Symposium Conference Record*, Valencia, Spain, 23–29 October 2011, pp. 2036–2038 [https://doi.org/10.1109/NSSMIC.2011.6154414].

[34] Integrated Detector Electronics AS, (2018) http://ideas.no/.

[35] Hamamatsu, *Hamamatsu H8500 MaPMT*, (2011) and online pdf version at https://www.hamamatsu.com/resources/pdf/etd/H8500_H10966_TPMH1327E.pdf.

[36] Integrated Detector Electronics AS, *ROSMAP-MP*, (2018) http://ideas.no/products/rosmap-mp/.

[37] S. Jaksch, *Recent Developments of the SoNDe High-Flux Detector Project*, in proceedings of the *International Conference on Neutron Optics (NOP2017)*, Nara, Japan, 5–8 July 2017.

[38] S. Jaksch et al., *Scintillation detector with a high count rate*, PCT/EP2015/074200 (2015).

[39] C. Höglund et al., *B_4C thin films for neutron detection*, *J. Appl. Phys.* **111** (2012) 104908.

[40] C. Höglund et al., *Stability of $^{10}B_4C$ thin films under neutron radiation*, *Radiat. Phys. Chem.* **113** (2015) 14.

[41] F. Messi et al., *The neutron tagging facility at Lund University*, `arXiv:1711.10286`.

[42] F. Piscitelli et al., *Characterization of the Multi-Blade 10B-based detector at the CRISP reflectometer at ISIS for neutron reflectometry at ESS*, 2018 *JINST* **13** P05009 [`arXiv:1803.09589`].

[43] F. Piscitelli et al., *The Multi-Blade Boron-10-based Neutron Detector for high intensity Neutron Reflectometry at ESS*, 2017 *JINST* **12** P03013 [`arXiv:1701.07623`].

[44] G. Mauri et al., *Neutron reflectometry with the Multi-Blade ^{10}B-based detector*, *Proc. Roy. Soc. Lond.* **A 474** (2018) 20180266 [`arXiv:1804.03962`].

[45] D. Pfeiffer et al., *The μTPC method: improving the position resolution of neutron detectors based on MPGDs*, 2015 *JINST* **10** P04004 [arXiv:1501.05022].

[46] CERN, *VMM3, an ASIC for Micropattern Detectors*, (2017) and online pdf version at https://indico.cern.ch/event/581417/contributions/2556695/attachments/1462787/2259956/MPGD2017_VMM3.pdf.

[47] M. Lupberger et al., *Implementation of the VMM ASIC in the Scalable Readout System*, *Nucl. Instrum. Meth.* **A 903** (2018) 91.

[48] M. Lupberger et al., *4.9: Detector electronics chain*, BrightnESS (2017).

[49] M. Shetty, M.J. Christensen, S. Skelboe, T. Richter and S. Board, *5.2: Report processing choices for detector types*, BrightnESS (2017).

[50] J. Postel, *User Datagram Protocol*, IETF (1980) and online at https://tools.ietf.org/html/rfc768.

[51] J. Postel, *Transmission Control Protocol*, IETF (1981) and online at https://tools.ietf.org/html/rfc793.

[52] R. Frazier, G. Iles, D. Newbold and A. Rose, *Software and firmware for controlling CMS trigger and readout hardware via gigabit Ethernet*, *Phys. Procedia* **37** (2012) 1892.

[53] M. Bencivenni et al., *Performance of* 10 *Gigabit Ethernet Using Commodity Hardware*, *IEEE Trans. Nucl. Sci.* **57** (2010) 630.

[54] packagecloud, *Monitoring and Tuning the Linux Networking Stack: Receiving Data*, (2016) https://blog.packagecloud.io/eng/2016/06/22/monitoring-tuning-linux-networking-stack-receiving-data/#special-thanks.

[55] Apache Kafka, (2018) https://kafka.apache.org/.

[56] Google FlatBuffers, (2018) https://github.com/google/flatbuffers.

[57] M. Edenhill, *librdkafka — the Apache Kafka C/C++ client library*, (2018) https://github.com/edenhill/librdkafka.

[58] *Qt*, (2018) https://www.qt.io/.

[59] *Daquiri*, (2018) https://github.com/ess-dmsc/daquiri.

[60] *Graphite*, (2018) https://graphiteapp.org/.

[61] Grafana Lab, *Grafana*, (2018) https://grafana.com/.

[62] *Graylog*, (2018) https://www.graylog.org/.

[63] C. Lonvick, *The BSD syslog Protocol*, IETF (2001) and online at https://tools.ietf.org/html/rfc3164.

[64] *GIT — distributed version control system*, (2018) https://git-scm.com/.

[65] *DMSC on Github*, (2018) https://github.com/ess-dmsc.

[66] CONAN, *C/C++ Package Manager*, (2018) https://conan.io/.

[67] CMake, *CMake — cross-platform tools for building software*, (2018) https://cmake.org/.

[68] *GCC, the GNU Compiler Collection*, (2018) https://gcc.gnu.org/.

[69] *Clang: a C language family frontend for LLVM*, (2018) https://clang.llvm.org/.

[70] *Jenkins*, (2018) https://jenkins.io/.

[71] *Google Test — C++ test framework*, (2018) https://github.com/google/googletest.

[72] *Cppcheck — A tool for static C/C++ code analysis*, (2018) http://cppcheck.sourceforge.net/.

[73] *Gcovr — gcov based code coverage report*, (2018) https://gcovr.com/.

[74] The Clang Team, *ClangFormat*, (2018) https://clang.llvm.org/docs/ClangFormat.html.

[75] *Valgrind — an instrumentation framework for building dynamic analysis tools*, (2018) http://valgrind.org/.

[76] *Google Benchmark — A library to support the benchmarking of functions*, (2018) https://github.com/google/benchmark.

[77] *ANSIBLE — managing complex deployments*, (2018) https://www.ansible.com/.

[78] *Event Formation Unit source code*, (2018) https://github.com/ess-dmsc/event-formation-unit.