# New Software-Based Readout Driver for the ATLAS Experiment

Serguei Kolos, Gordon Crone, and William P. Vazquez

*Abstract*—**To maintain sensitivity to new physics in the coming years of Large Hadron Collider (LHC) operations, A Toroidal LHC ApparatuS (ATLAS) collaboration has been working on upgrading a portion of the front-end (FE) electronics and replacing some parts of the detector with new devices that can operate under the much harsher background conditions of future LHC runs. The legacy FE of the ATLAS detector sent data to the data acquisition (DAQ) system via the so-called Read Out Drivers (RODs) custom-made VMEbus boards devoted to data processing, configuration, and control. The data were then received by the Read Out System (ROS), which was responsible for buffering them during the High-Level Trigger (HLT) processing. From Run 3 onward, all new trigger and detector systems will be read out using new components, replacing the combination of the ROD and the ROS. This new path will feature an application called the Software Read Out Driver (SW ROD), which will run on a commodity server receiving FE data via the Front-End Link eXchange (FELIX) system. The SW ROD will perform event fragment building and buffering as well as serving the data on request to the HLT. The SW ROD application has been designed as a highly customizable high-performance framework providing support for detector-specific event building and data processing algorithms. The implementation that will be used for Run 3 of the LHC is capable of building event fragments at a rate of 100 kHz from an input stream consisting of up to 120 MHz of individual data packets. This document will cover the design and the implementation of the SW ROD application and will present the results of performance measurements executed on the server models selected to host SW ROD applications during Run 3.**

*Index Terms*—**Data acquisition (DAQ), data collection, data transfer, object-oriented programming.**

## I. Introduction

As PART of the preparation for Large Hadron Collider (LHC) Run 3, which will start early in 2022, A Toroidal LHC ApparatuS (ATLAS) [1] collaboration has upgraded some parts of the detector with new components, able to operate under the much harsher background conditions expected as the LHC reaches higher instantaneous luminosity. The new detector and trigger systems will use modern front-end (FE) electronics that require an updated readout system. During the first two LHC runs, the trigger and data acquisition (TDAQ) system [2] received data from the detector FE electronics via custom-made VMEbus boards called Read Out Drivers (RODs) [3]. The data from RODs were received by the Read Out System (ROS) [2], which was responsible for buffering and serving them to the High-Level Trigger (HLT) [2]. From Run 3 onward, the TDAQ system will use a new facility called the Software Read Out Driver (SW ROD) for all new trigger and detector components. The SW ROD will receive FE data via the FE Link eXchange (FELIX) system [4] and perform event fragment building and buffering as well as serving data on request to the HLT.

## II. FELIX System Overview

FELIX is a new generic detector readout system that can receive data from detector FE electronics via (among others) the versatile radiation hard optical link architecture [5] developed at Conseil Européen pour la Recherche Nucléaire (CERN). FELIX can be used to receive data via either Giga-Bit Transceiver (GBT) [6] or the in-house designed FULL mode protocol. FELIX uses a custom peripheral component interconnect express (PCIe) card that receives data via optical links and passes them to the memory of a commodity computer via the PCIe bus. FELIX also provides a software application that can forward these data to a number of subscribers via a commercial network. To maximize performance, the software uses RDMA-capable protocols. RDMA stands for Remote Direct Memory Access, a technology that makes it possible to put data directly into the main memory of another computer without involving the CPU, network software stack, or operating system kernel of that computer. FELIX implements a custom network communication layer called NetIO [7] on top of the RDMA over Converged Ethernet (RoCE) protocol that is supported by many modern network cards. NetIO provides a C application programming interface (API) that can be used by a software application to receive data from the FELIX system. A FELIX card can be operated in two modes using the respective protocols:

1) GBT Mode: With the GBT protocol, a physical input link can be subdivided into a number of logical sublinks (known as E-Links), which can pass information from separate pieces of FE electronics. For Run 3, the maximum number of E-Links for a single FELIX card is limited to 192, which in this case are equally spread over 24 GBT links.
2) FULL mode: This mode has no logical subdivision of links and uses an in-house designed protocol for higher

Manuscript received October 28, 2020; revised January 28, 2021 and March 2, 2021; accepted May 23, 2021. Date of publication June 8, 2021; date of current version August 16, 2021.

Serguei Kolos is with the Department of Physics and Astronomy, University of California, Irvine, CA 92697 USA.

Gordon Crone is with the Department of Physics and Astronomy, University College London, London WC1E 6BT, U.K.

William P. Vazquez is with Royal Holloway, University of London, London TW20 0EX, U.K.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TNS.2021.3083987.

Digital Object Identifier 10.1109/TNS.2021.3083987

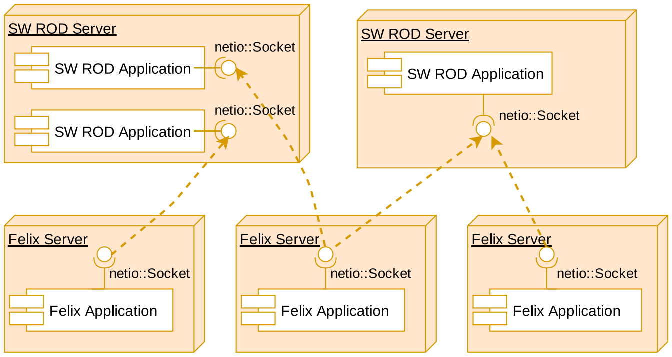

Fig. 1. SW ROD deployment diagram. An SW ROD application is fully flexible with respect to its location and connectivity. It can be freely moved from one computer to another and can receive data from an arbitrary number of FELIX applications.

bandwidth. For Run 3, this mode can be used to send data either via 12 links at full occupancy with the speed of 9.6 Gbps or via 24 links with 50% occupancy (4.8 Gbps).

## III. SW ROD System Architecture

The SW ROD facility is envisaged to be implemented as software running on a set of commodity computers. Given that a single computer can serve only a limited amount of input data, and to scale to the size of the new ATLAS readout, that for Run 3 will receive data from about a hundred FELIX input cards, the software had to be designed in a way that allowed it to be distributed over an arbitrary number of computers. In the current design, this is achieved by splitting the input data channels between a number of software processes, which are referred to as SW ROD applications as shown in Fig. 1.

Each instance of the SW ROD application can run on a separate computer, but it originates from the same binary executable. This executable implements a highly customizable high-performance framework, rendering support for detector-specific event building and data processing algorithms provided in the form of shared libraries (a.k.a. binary plugins). This way different instances of the SW ROD application can be specialized at run time by providing different configurations, and each configuration defines a set of plugins to be used as well as their configuration parameters.

## IV. SW ROD Application Functional Requirements

The RODs, used by the legacy readout system to receive and process data from the ATLAS detector FE, were developed independently by every subdetector. As such, they perform subdetector-specific data processing and event building based on the signals received from the ATLAS Central Trigger Processor (CTP) [8]. As the FELIX system does not perform any data processing or event aggregation, but merely provides data routing between detector FE and the data acquisition (DAQ) system, the task of data aggregation and processing has to be fulfilled by the SW ROD application before transferring data to the HLT farm. The SW ROD is expected to be used not only for normal physics data taking but also for various auxiliary subdetector-specific activities, such as commissioning, calibration, monitoring, and debugging. During these activities, data will undergo specific processing and may need to be transferred to a different destination than the HLT farm. To meet such requirements, the SW ROD application has been designed as a framework that supports a high degree of customization by making it possible to load subdetector-specific event building and data processing algorithms at run time, which can be further configured by subdetectors with respect to their specific needs.

## V. SW ROD Application High-Level Design

The SW ROD application is split internally into a number of independent components. Each of them exposes a simple interface that defines how other components can interact with it. There are three main components defined by the SW ROD application architecture that can be accessed via the respective interfaces:

1) DataInput interface: It abstracts a source of input data to shield other components of the SW ROD application from any changes in the network input protocol. In addition, it also makes it possible to use another data source, for example, internal data generators, for testing and debugging.
2) ROBFragmentBuilder interface: It abstracts implementations of data aggregation algorithms, which may be needed to facilitate the implementation of different data-handling strategies as required by subdetectors and should be able to support different FELIX operation modes. This approach scales well with a number of data aggregation algorithms, as adding a new algorithm does not require modification of the existing ones. It also offers the possibility to support auxiliary data aggregation strategies to be used for calibration or monitoring without affecting the basic procedures used for normal physics data taking. An implementation of this interface is responsible for aggregating data chunks from individual E-Links received via the DataInput interface into event fragments according to the given configuration. Such a configuration defines the set of event fragments to be produced as well as a list of input links for each fragment.
3) ROBFragmentConsumer interface: It abstracts any kind of processing that can be applied to fully aggregated event fragments. Multiple implementations of this interface can be used simultaneously in the same SW ROD application, in which case they will be organized into a singly linked list. Each consumer in this list will have to forward event fragments to the next one after finishing its specific processing step. For example, as shown in Fig. 2, one implementation of this interface can apply a custom subdetector-specific processing procedure to the event fragments before passing them to another consumer that is used to transfer these fragments to the HLT farm.

With such a design, data-handling is done by the implementations of the SW ROD application interfaces, while the application itself merely loads and instantiates the corresponding implementation classes in accordance with a given

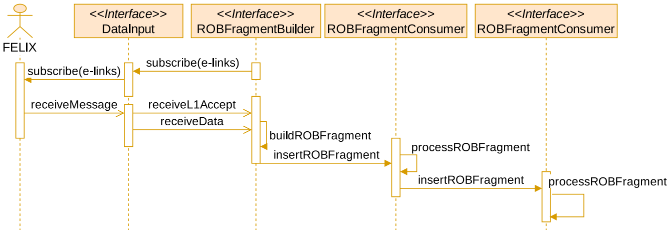

Fig. 2. Typical interactions between SW ROD application components for a normal data-taking activity. The DataInput component subscribes to FELIX and passes received data to the ROBFragmentBuilder, which aggregates data into event fragments and transfers them to the first ROBFragmentConsumer in the chain. This chain can contain an arbitrary number of consumers, which pass event fragments from one another along the chain.

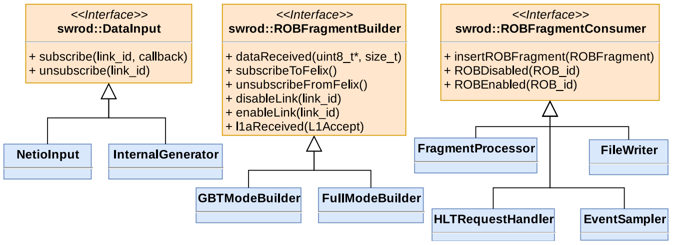

Fig. 3. Default SW ROD interface implementations, which are provided by the TDAQ software release. A new custom implementation of any SW ROD interface can be added to the SW ROD application dynamically as a plug-in.

configuration and links the instantiated objects in the order defined by this configuration.

## VI. SW ROD Default Component Implementations

A shared library that contains default implementations for all three interfaces is supplied along with the SW ROD application. This library contains all classes shown in Fig. 3.

### A. DataInput Interface Implementations

1) The NetioInput class is responsible for receiving data from the FELIX system using the NetIO Socket interface for a given set of E-Links and passing these data to the fragment builder via the ROBFragmentBuilder interface.
2) The InternalDataGenerator can generate FELIX-like data chunks of a given size for a configurable number of E-Links. This class is used for debugging and for unit test implementation.

### B. ROBFragmentBuilder Interface Implementations

The library provides two implementations of the ROBFragmentBuilder interface, which can be used to receive data from the FELIX system in either GBT or FULL mode. The algorithms implement a specific data aggregation strategy in a generic way that is independent of the format of the incoming data chunks. As this format is detector-specific, this feature was implemented by allowing detectors to supply two custom procedures as parameters for these algorithms:

1) Trigger Information Extraction procedure: This is a function that extracts the Level 1 Trigger identifiers from a given data chunk. These identifiers are used to assign data chunks to a particular event fragment and to align data with the Trigger information received from the CTP.
2) Data Integrity Checking procedure: This function is intended to be used if there is a suspicion that input data chunks could be corrupted or a sequence of data packets for a particular input link is broken. This function is assumed to know the location of the checksum value in a given data packet format and the Cyclic Redundancy Check CRC) algorithm that was used to calculate that value.

In most cases, detector developers have to define only these functions and reuse the data aggregation strategies provided by the carefully optimized and extensively tested default ROBFragmentBuilder interface implementations. On the other hand, if another event fragment aggregation strategy is required for a particular subdetector, a new algorithm can be implemented and plugged in to the SW ROD application as is done for default implementations. This does not affect the existing components of the SW ROD application and is completely transparent for the application itself.

### C. ROBFragmentConsumer Interface Implementations

1) The FragmentProcessor class was developed to simplify implementation of the common task, required by many subdetectors, of applying custom detector-specific postprocessing to all event fragments produced by the given SW ROD application. This class provides a workbench to execute detector-specific code on every event fragment that is passed to this consumer. This code should perform the necessary modifications to the event fragment payload but should keep the structure of the fragment untouched. The code can be provided in the form of a function, which has to be implemented by the corresponding detector experts and be given to the SW ROD application in the form of a shared library that will be loaded at runtime.
2) The HLTRequestHandler class is responsible for buffering event fragments and serving them to the HLT farm on request. It keeps the event fragments until informed by the HLT that they are no longer needed. Event fragments are indexed by their Level 1 Trigger identifier and stored in an internal buffer until a clear request has been received from the HLT. On receipt of a clear request, all the event fragments with the identifiers provided by this request will be removed from the index and their allocated memory freed.
3) The FileWriter class implements a consumer that simply writes all received event fragments to a file on disk. The files created by the FileWriter will be in the standard ATLAS data file format [9] with all event fragments prepended by the ATLAS full event header, which makes such files compatible with standard ATLAS event processing and analysis applications. This functionality is useful for testing, commissioning, calibration, and other auxiliary activities which are performed by detectors beyond normal data taking.
4) The EventSampler implements event selection for online monitoring. An instance of this class can be optionally added to the list of an SW ROD application consumers

TABLE I

FELIX CARD OUTPUT RATES FOR RUN 3

| Mode | Chunk Size (B) | Chunk Rate per Link (kHz) | Links per FELIX Card | Chunk Rate per Card (MHz) | Data Rate per Card (GB/s) |
|---|---|---|---|---|---|
| GBT | 40 | 100 | 192 | 19.2 | 0.77 |
| FULL | 5000 | 100 | 12 (24) | 1.2 (2.4) | 6 |

to select a subset of aggregated event fragments for the purpose of online monitoring. This class passes selected events to the TDAQ Event Monitoring service [10] that transfers them to the applications responsible for data quality assessment.

## VII. SW ROD APPLICATION PERFORMANCE REQUIREMENTS

In Run 3, the SW ROD has to be able to operate at an input rate of 100 kHz, matching the ATLAS Level 1 Trigger accept rate. The number of input links and the overall data rates are defined by the output produced by the FELIX system. Table I summarizes these numbers for a single FELIX card.

An important goal of the SW ROD is to handle as many input data links as possible to reduce the total system price by minimizing the number of computers to be used to run SW ROD applications. While in FULL mode this number is essentially defined by the input data rate requirements, and in GBT mode the data rate produced by a single FELIX card is much lower and the number of input links that can be served by a single SW ROD computer is mostly driven by the performance of the GBT data aggregation algorithm. A dedicated study has been performed to estimate the maximum number of input links that can be handled by the GBT event fragment aggregation algorithm executed on a single SW ROD computer. The results of this study will be presented in Section VIII.

## VIII. GBT MODE EVENT FRAGMENT BUILDING ALGORITHM OPTIMIZATION

Due to power consumption and heat dissipation issues, the clock frequency of a modern CPU depends on the number of cores, and the product of these parameters gives a similar value for any CPU in the same price range. This value can be used to make a rough estimate of the full computing power a particular CPU can offer. It should be noted that a modern CPU is capable of executing more than one operation per cycle, but in practice this is difficult to achieve for complex code, and normally a one-to-one ratio between cycles and operations is considered satisfactory. Taking 2.5 GHz as an average CPU frequency for a CPU that has 10 cores, one can assess the total number of CPU operations per second provided by an averagely priced CPU to be on the order of $2.5 \cdot 10^{10}$.

Given that the rate of data chunks from a single FELIX card in GBT mode is about 20 MHz, a simple division shows that such a CPU can provide 1250 operations for a single data chunk, which corresponds to about 0.5 ms. If one wants to maximize the number of FELIX cards that can be handled by the same SW ROD computer, this budget has to be divided further accordingly. It should be taken into account that every chunk has to be aligned, by means of the Level 1 Trigger identifier that this chunk contains, with the other ones for the purpose of event fragment aggregation. This requires the extraction of L1ID from each data chunk and finding an appropriate event buffer to which to copy the chunk. Given the size of GBT data packets, this is more efficient than using a zero-copy memory management technique. Moreover, the computational resources are spread over all cores of the given CPU, which means that to use them in an efficient way the software has to be designed to use multiple threads with a high degree of parallelism. This essentially precludes the use of high-level design patterns, like producer-consumer, to pass data between threads as this would incur too much performance overhead for thread synchronization.

The solution that was implemented for the GBT event fragment aggregation algorithm to minimize the rate of interactions between threads was to combine both data reading and event fragment aggregation into the same thread. To achieve that the total number of input E-Links is split among a configurable number of worker threads, with each thread reading data chunks from the given subset of E-Links and aggregating them into a subfragment of a given event. When a subfragment is ready, the worker thread passes it to the final fragment building stage via the `tbb::concurrent_map` container [11], which uses L1ID as key. This approach makes it possible to split the algorithm into two stages:

1) The processing of individual data chunks is done in parallel by multiple concurrent threads at the O(10) MHz rate.
2) The final event fragment assembly that requires synchronization between threads is done at the rate of 100 kHz only.

The degree of parallelism provided by this algorithm can be estimated using a formula that is based on Amdahl's law [12]

$$S(n) = \frac{1}{(1-P) + \frac{P}{n}}. \tag{1}$$

Equation (1) defines how the speedup $S(n)$ of an algorithm executed by a given number of threads $n$ depends on the parallel fraction of this algorithm $P$. Given that we know the processing rates of the parallel and nonparallel fractions of the GBT event fragment aggregation algorithm, we can express $P$ as (2)

$$P = 1 - C\text{FA} \times \frac{10^5}{10^7} = 1 - 0.01 \times C\text{FA}. \tag{2}$$

Here, $C\text{FA}$ is the relative cost of the final subfragment assembly operation with respect to the cost to handle a single data chunk. This equation shows that if the relative cost of the final assembly operation is less than 100, then the algorithm should give some performance gain, but to scale well this number should be less than 50. Using this, (1) can be transformed to

$$S(n) = \frac{n}{0.01 \times C\text{FA}(n-1) + 1}. \tag{3}$$

This equation defines how the speedup of the GBT algorithm depends on the relative cost of the final assembly operation. Finally, inverting (3) yields (4), which will be used in the next chapter to assess $C\mathrm{FA}$ for the current algorithm implementation using the empirical values for $S(n)$ obtained from performance measurements

$$C\mathrm{FA} = \frac{\left(\frac{n}{S(n)} - 1\right)}{0.01 \times (n-1)}. \tag{4}$$

## IX. Performance Measurements

### A. Testbed Configuration

Event building algorithm performance measurements were performed on a testbed that replicates the same hardware configuration that will be used by the readout system during Run 3:

1) SW ROD application running on a computer with a dual-socket motherboard with 2 Intel(R) Xeon(R) Gold 5218 CPUs and 96 GB of DDR4-2667 RAM. Each CPU has 16 physical cores with a base frequency of 2.3 GHz.
2) Input data for the tests generated by a FELIX card software emulation application running on another computer with an Intel Xeon E5-1660 v4 CPU with 3.2 GHz base frequency and equipped with 32 GB DDR4 2667 MHz memory.
3) Both computers were equipped with Mellanox ConnectX-5 100 GbE network adapters, which were connected via Juniper QFX5200-32C switch. Data were sent to the SW ROD application via the FELIX NetIO protocol.

### B. Network Throughput Test

To assess the overhead of the RoCE protocol, the network throughput was measured using a simple bandwidth test utility from the Mellanox OFED-4.7 software package with a default packet size of 65K bytes and maximum transmission unit (MTU) value of 1500 bytes. The receiving application was started on the SW ROD computer with the following command:

```
# ib_send_bw -F -n 100000
```

The client (sending) application was started on FELIX computer with the Internet Protocol (IP) address of the SW ROD host:

```
# ib_send_bw -F -n 100000 192.168.100.1
```

Both applications reported an average rate of 91.3 Gb/s that stayed almost constant throughout the test, with marginal variations of less than a fraction of 1 Gb/s.

### C. GBT Mode Tests

The aim of these tests was to study how the GBT event fragment building algorithm scales with the number of input E-Links and the number of threads used to handle input data. To this end, three series of tests were performed with the SW ROD application using one, two, and three threads, respectively, to receive and aggregate data chunks from every group of 192 input links, which corresponds to the input from a single FELIX card. The software that generated data for these tests simulated input from up to 6 FELIX cards, which represents 6% of the total number of FELIX cards for Run 3 and about 1% for Run 4. The total number of E-Links for the tests increased gradually from 192 to 1152. The size of the generated data chunks was set to 40 bytes. The results of these tests are shown in Fig. 4.

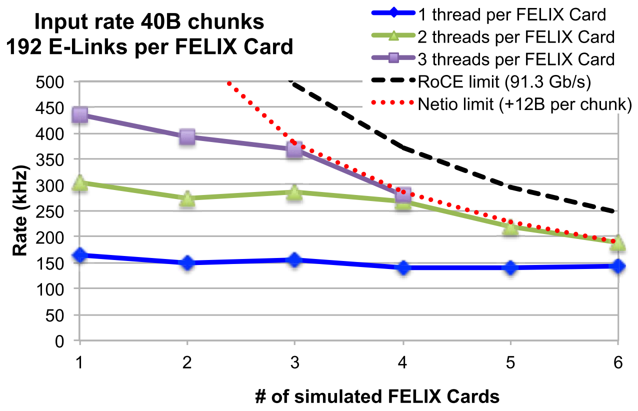

Fig. 4. GBT event fragment aggregation algorithm performance test results. The algorithm scales well with the number of worker threads used for aggregating data from a FELIX card and with the number of cards. For the given hardware configuration, the algorithm performance is limited by the 100 GbE input network bandwidth.

TABLE II
ESTIMATE OF THE PARALLEL FRACTION OF GBT ALGORITHM

| N of threads | Average $S(n)$ | $C\mathrm{FA}$ | $P$ |
|---|---|---|---|
| 2 | 1.86 | 7.5 | 0.925 |
| 3 | 2.65 | 6.6 | 0.934 |

These results show that the GBT event fragment aggregation algorithm implementation scales well in both dimensions: with the number of worker threads aggregating data from a given number of E-Links and with the number of such aggregation operations running concurrently in the scope of the same SW ROD application.

The dotted line shows the maximum theoretical input rate that can be obtained with the given hardware configuration, which is limited by the available network bandwidth. This line represents (5)

$$L = \frac{91.3 \times 10^9}{192 \times F \times (40 \times 8 + 12 \times 8)} \tag{5}$$

where $L$ is the input rate, $F$ is the number of simulated FELIX cards, $40 \times 8$ is the size of the data chunk in bits, $12 \times 8$ is the size of the NetIO protocol overhead per chunk in bits as well, and $91.3 \cdot 10^9$ is the maximum bandwidth that can be achieved with using the RoCE protocol in Gb/s. This line demonstrates that the last three results of the tests with two reading threads and all but the first two results for the test series with three reading threads were limited by the network bandwidth. The dashed line shows the maximum rate that could be achieved if the NetIO protocol overhead was equal to zero. It indicates that the input rate could potentially be improved by reducing the NetIO overhead.

Using the results which were not limited by network bandwidth, one can calculate the speedup $S(n)$ and parallel fraction $P$ of the GBT event fragment assembly algorithm and then use these numbers with (4) to compute an estimate of the $C\mathrm{FA}$ coefficient. The results of these calculations are shown in Table II.

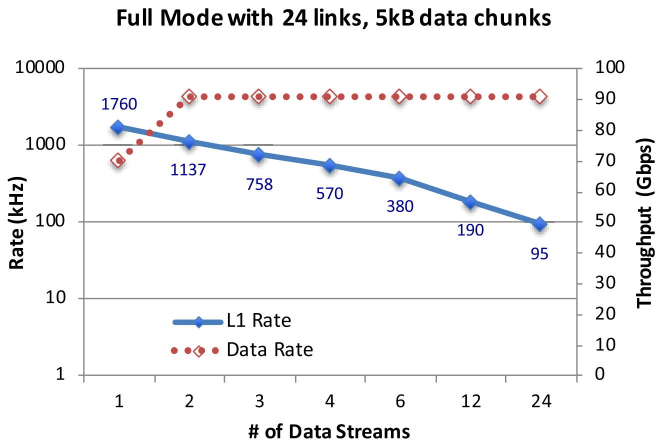

Fig. 5. FULL mode data handling algorithm performance test results. The algorithm scales well with the number of input data streams, except the extreme case of using a single input stream only. In this case, the algorithm performance is CPU-limited, but this is insignificant as in this case simulated data rate exceeded by an order of magnitude a FULL mode link bandwidth.

### D. FULL Mode Tests

In FULL mode, larger sized event fragments are sent to FELIX over fewer (higher bandwidth) links, with no data aggregation required in the SW ROD. The number of links needing to be serviced at this increased bandwidth can vary from 1 to 24 in the extreme case. In FULL mode, several links can be grouped together to be used to send fragments corresponding to different Level 1 trigger events in a round-robin pattern from the same piece of detector FE electronics if more bandwidth than a single link can provide is required.

Tests have been performed to study the behavior of the SW ROD's FULL mode data-handling algorithm relative to the number of input link groups, which need to be serviced independently. Each group of links was used to send an independent stream of data and, inside a group, event fragments were sent over the given links using the round-robin pattern. For these tests, the size of the generated packets was set to 5 kB, the maximum expected packet size for Run 3, and the number of independent streams of data generated by the FELIX software simulator varied from 1 to 24. For each configuration, the average input rate per data stream was measured. The results of these tests are shown in Fig. 5.

The results demonstrate the excellent scalability of the FULL mode data-handling algorithm with respect to the number of input links. In all test series, except one input rate was limited by the network bandwidth. The only exception is the configuration with all 24 input links used for the same data stream. In this test, the input rate went up to 1.76 MHz, which saturated the CPU cores used by the SW ROD application's reading threads. No further study has yet been done for this scenario as the rate that was achieved is far in excess of the 100-kHz input data rate requirement for Run 3.

### E. Scalability Toward Run 4 Requirements

For Run 4, which is planned to start in 2027, the LHC will undergo the High Luminosity Upgrade [13] that will significantly increase instantaneous luminosity and the number of particle interactions per bunch crossing. The high luminosity LHC (HL-LHC) baseline parameters with a peak luminosity of 5 to 7 $\times 10^{-34}$ cm$^{-2}$ s$^{-1}$ and an average number of *pp* interactions per bunch crossing 140 to 200 will bring new challenging requirements for the ATLAS TDAQ system, which will have to receive data from trigger and detector electronics at an input rate of 1 MHz. As the readout system for Run 4 will be based on FELIX, it is useful to study the limits of the current readout implementation, such that they can be addressed in the new TDAQ architecture. For this reason, a series of tests were performed with the GBT event fragment aggregation algorithm to reveal the maximum number of input E-Links and chunk size configurations at which 1 MHz input can be sustained. For these tests, the SW ROD application used the default GBT event fragment aggregation algorithm that assembles data from all the given input E-Links to a single fragment. Two rounds of tests were performed. In the first one, the number of input E-Links varied from 24 to 192 and the input data chunk sizes were set to 40 and 80 bytes for different test series. For the second round of tests, the number of input E-Links was fixed to be 48 and the data chunk size varied from 40 to 240 bytes. The algorithm used six reading threads for both rounds of tests.

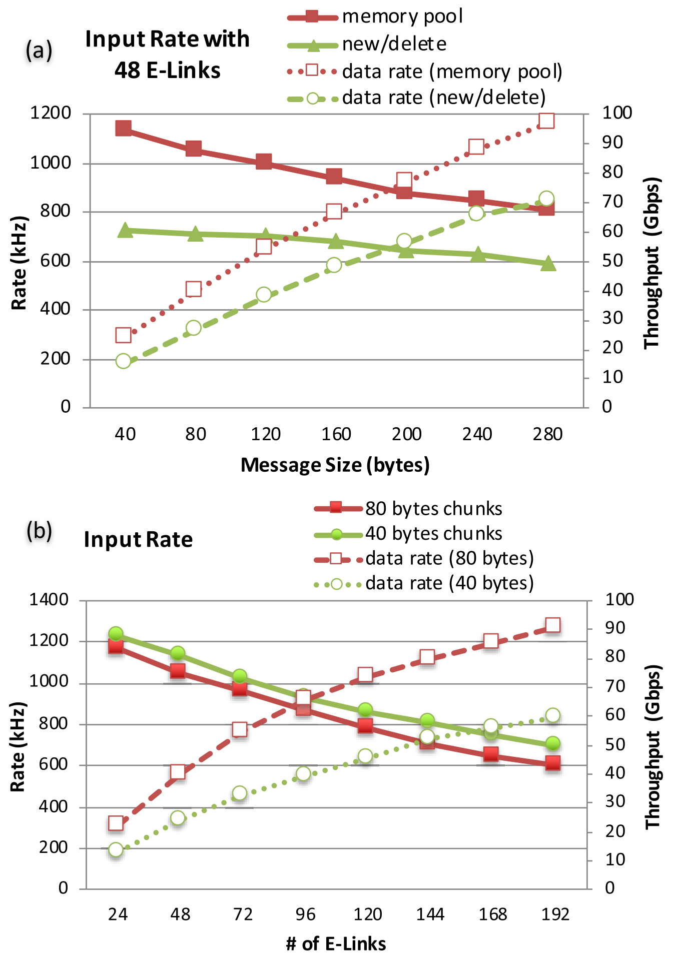

Fig. 6. SW ROD maximum input rate as a function of data chunk size (a) and number of input E-Links (b). In both tests, the results are CPU-limited and were improved by replacing the standard memory management *new* and *delete* operations with a custom memory pool. (a) Difference between the two implementations. (b) Results which were obtained using memory pool modification with two different chunk sizes.

Fig. 6(a) shows two data series which were obtained with the same test configurations but using different versions of the SW ROD application. The first test series revealed a bottleneck in the SW ROD application that was caused by the standard new and delete memory management operations. This was not a problem for previous tests, where these operations were taking place at a rate of about 100 kHz, but when the input rate was increased to 1 MHz the memory management overhead became prominent. A quick solution was put in place by replacing the new and delete operations with a custom memory pool implementation that preallocates a large number of memory blocks and keeps a list of free blocks in a `tbb::concurrent_queue` container [11], which made the use of this memory pool by multiple concurrent threads possible. This improved the input rate of the SW ROD application by almost 50% and made it possible to reach a rate above 1 MHz with some configurations. The same implementation was used for the second round of tests, for which the results are shown in Fig. 6(b).

## X. Conclusion

A mixture of the legacy ROD-based and the new FELIX-based readout will be used by the ATLAS TDAQ system for LHC Run 3. The SW ROD is a new component of the ATLAS DAQ system that was developed to receive data from FELIX. The SW ROD implements a high-performance customizable framework that supports custom input data formats and different event fragment aggregation strategies as required by the new ATLAS detector and trigger components. The SW ROD fully satisfies the performance and functional requirements which have been defined by ATLAS for Run 3. The default GBT event fragment aggregation algorithm makes it possible to handle data input from up to 6 FELIX cards at the rate of about 150 kHz, giving enough safety margin to reliably sustain 100-kHz rate for Run 3. A single SW ROD server can also handle data coming from 12 FULL mode links of a FELIX card at the rate of 190 kHz, which is almost twice the Run 3 input rate requirement. Further optimization could be achieved by reducing the overhead of the FELIX communication protocol. A study of how the Run 4 performance requirements can be met is ongoing and has already revealed some very promising results.

## References

[1] T. A. Collaboration *et al.*, "The ATLAS experiment at the CERN large hadron collider," *J. Instrum.*, vol. 3, no. 08, Aug. 2008, Art. no. S08003, doi: 10.1088/1748-0221/3/08/S08003.

[2] ATLAS TDAQ Collaboration, "The ATLAS data acquisition and high level trigger system," *J. Instrum.*, vol. 11, no. 6, Jun. 2016, Art. no. P06008, doi: 10.1088/1748-0221/11/06/P06008.

[3] A. Gabrielli, "Commissioning of ROD boards for the entire ATLAS pixel detector," *J. Instrum.*, vol. 13, no. 9, Sep. 2018, Art. no. T09009, doi: 10.1088/1748-0221/13/09/t09009.

[4] S. Ryu, "FELIX: The new detector readout system for the ATLAS experiment," *J. Phys., Conf. Ser.*, vol. 898, Oct. 2017, Art. no. 032057, doi: 10.1088/1742-6596/898/3/032057.

[5] F. Vasey *et al.*, "The versatile link common project: Feasibility report," *J. Instrum.*, vol. 7, no. 1, Jan. 2012, Art. no. C01075, doi: 10.1088/1748-0221/7/01/c01075.

[6] P. Moreira *et al.*, "The GBT project," in *Proc. Topical Workshop Electron. Phys.*, Sep. 2009, pp. 342–346. [Online]. Available: https://cds.cern.ch/record/1235836

[7] J. Schumacher, C. Plessl, and W. Vandelli, "High-throughput and low-latency network communication with NetIO," *J. Phys., Conf. Ser.*, vol. 898, Oct. 2017, Art. no. 082003, doi: 10.1088/1742-6596/898/8/082003.

[8] G. Anders *et al.*, "The upgrade of the ATLAS level-1 central trigger processor," *J. Instrum.*, vol. 8, no. 1, Jan. 2013, Art. no. C01049, doi: 10.1088/1748-0221/8/01/c01049.

[9] C. P. Bee *et al.*, "The raw event format in the ATLAS trigger/DAQ," CERN, Geneva, Switzerland, Tech. Rep. ATL-DAQ-98-129, Feb. 2016. [Online]. Available: https://cds.cern.ch/record/683741

[10] I. Scholtes, S. Kolos, and P. F. Zema, "The ATLAS event monitoring service—Peer-to-peer data distribution in high-energy physics," *IEEE Trans. Nucl. Sci.*, vol. 55, no. 3, pp. 1610–1620, Jun. 2008, doi: 10.1109/TNS.2008.924057.

[11] A. D. Robison, "Intel threading building blocks (TBB)," in *Encyclopedia of Parallel Computing*, D. A. Padua, Ed. Boston, MA, USA: Springer, 2011, pp. 955–964.

[12] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proc. Spring Joint Comput. Conf. AFIPS (Spring)*, vol. 30, Apr. 1967, pp. 483–485.

[13] C. Bernius, o. behalf of the ALICE, and C. Collaborations, "HL-LHC prospects from ATLAS and CMS," *J. Phys., Conf. Ser.*, vol. 1271, Jul. 2019, Art. no. 012004, doi: 10.1088/1742-6596/1271/1/012004.