# The ATLAS TDAQ DataCollection Software

**Inauguraldissertation**
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

**Christian Häberli**
von Münchenbuchsee BE und Zürich ZH

Leiter der Arbeit: Prof. Dr. K. Pretzl
Laboratorium für Hochenergiephysik

# The ATLAS TDAQ DataCollection Software

**Inauguraldissertation**

der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

**Christian Häberli**
von Münchenbuchsee BE und Zürich ZH

Leiter der Arbeit: Prof. Dr. K. Pretzl

Laboratorium für Hochenergiephysik

Von der Philosophisch-naturwissenschaftlichen Fakultät angenommen.

Bern, den 18. Dezember 2003 Der Dekan

Prof. Dr. G. Jäger

## Abstract

The Large Hadron Collider, which is currently under construction at CERN near Geneva, will collide protons with a center-of-mass energy of 14 TeV. This high energy offers the possibility to discover particles with masses on the TeV scale. Bunches of $1.15 \times 10^{11}$ protons will cross at a rate of 40 MHz. 23 proton-proton collisions will happen at every bunch-crossing, which results in a total proton-proton interaction rate of almost one GHz. The biggest part of these interactions do not contain new physics but mostly QCD background. Therefore the detectors dedicated to discovery physics, such as ATLAS, need to select the $\sim 100$ bunch-crossings with the biggest discovery potential out of the $40 \times 10^6$ bunch-crossings per second.
In case of the ATLAS experiment this reduction will be achieved on a three level trigger system. The first level trigger runs on custom hardware, the two higher trigger levels run as software algorithms on farms of hundreds of commodity PCs. The second level trigger will run at a rate of up to 100 kHz on a subset of the event data (16 kB in average) and the third level trigger will run at a rate of around 3 kHz on the full event data (1.2 MB in average).
The DataCollection subsystem is a part of the data acquisition system and has to provide the event data for the second level trigger (1.6 GB/s), for the third level trigger (3.6 GB/s) and in case of selected events to mass storage ($\sim$ 120 MB/s). The event data is obtained from 144 or 1600 data sources (depending on an open choice on the architecture of the data acquisition system).
This thesis explains the infrastructure which is needed to fulfill the demanding task of the DataCollection subsystem, focusing on the software aspects. In addition, performance studies on the DataCollection software and studies on the open choice in the architecture are presented.

# Contents

# Chapter 1

# Introduction

The ATLAS (A Toroidal LHC ApparatuS) experiment [WEB01] is a multi-purpose detector for the Large Hadron Collider [WEB02]. Both are currently under construction at the European Laboratory for Particle Physics (CERN) near Geneva, Switzerland.
In the year 2007 the Large Hadron Collider (LHC) at CERN will collide protons at a center of mass energy of 14 TeV. This opens new experimental possibilities on the TeV energy scale. It will allow to validate theoretical predictions such as the Higgs-mechanism or Supersymmetry, by looking for the existence of the respective Higgs and SUSY particles. The LHC will accelerate protons in bunches of $1.5 \times 10^{11}$. The bunches will cross at four collision points at a rate of 40 MHz. On average 23 proton-proton interactions will occur per bunch-crossing.
At one of the four collision points the ATLAS experiment [WEB01] is being constructed. The experiment is designed as a multi-purpose experiment to exploit the full discovery potential of the LHC. One of the main challenges of the experiment (on the level of the detector, the trigger system and the offline data analysis) is to handle the enormous rate of proton-proton interactions ($\sim 10^9$ per second). Given the fact that the biggest part of those interactions are background and do not contain new physics, ATLAS wants to cope with this challenge by applying a three-level trigger system, consisting of the Level 1 trigger, the Level 2 trigger and the Event-filter. The trigger system will select $\sim$ 100 bunch-crossings per second with the biggest discovery potential. The Level 1 trigger runs on custom-built hardware, whereas the Level 2 trigger and the Event-filter are implemented as software processes runs on farms of hundreds of commodity PCs. The algorithms run on parts (Level 2) or on the full (Event-filter) event data. The Level 2 runs at the Level 1 accept rate of up to 100 kHz (technical limitation of the Level 1 trigger) and needs to be supplied with 16 kB in average per event (data rate of 1.6 GB/s). The

Event-filter runs at the Level 2 accept rate of 3 kHz and needs to be supplied with full event of the size of 1.2 MB (3.6 GB/s data rate).
After an event is accepted by Level 1, its data is passed to the data acquisition system (DAQ). The DAQ system handles all data movements, until the event is eventually written to mass storage [ATL03]. Two components of the Dataflow system provide this part of the DAQ functionality: the Read-out Subsystem (ROS) and the DataCollection subsystem. The ROS houses Read-out Buffers (ROBs), which receive data fragments of Level 1 accepted events via the Read-out Links (ROLs) and stores them until each event is either rejected by Level 2 or fully built and ready to be sent to the Event-filter. It is an open architectural choice how the ROS will be designed in the final DAQ system for ATLAS.
The task of the DataCollection subsystem is

- to transport and to make available parts of the events (Regions of Interest) to the Level 2 trigger by obtaining the data from the ROS
- to build the Level 2 accepted events completely by obtaining the event data from the ROS, and to make those events accessible to the Event-filter
- to make Event-filter accepted events available to mass storage

The DataCollection subsystem will obtain the event data from 144 - 1600 data sources (depending on the ROS architecture) and deliver it to several hundred destinations.
In order to fulfill these tasks network infrastructure connecting the data sources and the destinations and a CPU/software infrastructure steering the flow of data is needed. As link technology Gigabit Ethernet has been chosen. As CPU/software infrastructure software applications running on commodity PCs driven by the Linux operating system were implemented: *the ATLAS TDAQ DataCollection Software* [WEB03].
This thesis will first give a short overview of the discovery potential of the LHC, of the ATLAS physics goals and of the detector layout. After a short discussion of the trigger system the DataCollection subsystem will be described in detail. A short chapter will be dedicated to the Gigabit Ethernet network topology. The architecture of the DataCollection software and its components will be explained. Based on this the discussion on the design of the Event-building (EB) applications will be described. Finally a detailed performance study on the EB applications and the EB system will be presented, together with a comparative study on the performance of the EB assuming two different options of the ROS architecture.

The reader of this thesis should be aware of the very frequent use of acronyms. These acronyms and other specific expressions throughout the text are spelled out in the glossary at the end.

# Chapter 2

# The Large Hadron Collider

The Large Hadron Collider (LHC) is a proton-proton accelerator, which is currently under construction at CERN in Geneva and will be operational in the year 2007 [WEB02].
The LHC will collide protons at a center of mass energy of 14 TeV. The luminosity will be $0.12 \times 10^{34}$ cm$^{-2}$s$^{-1}$ at startup (low luminosity) and will increase to the design value of $1 \times 10^{34}$ cm$^{-2}$s$^{-1}$ (high luminosity). The high luminosity corresponds to a proton-proton collision rate of $1 \times 10^{9}$ per second, which can be compared with the current rate at Tevatron[1] of $2.1 \times 10^{7}$ per second at a peak luminosity of $4.2 \times 10^{31}$ cm$^{-2}$s$^{-1}$ [WEB05].
At high luminosity the protons will be grouped in bunches of $1.15 \times 10^{11}$, which will cross at a rate of $4 \times 10^{7}$. This means that on average 23 proton-proton interactions will occur per bunch crossing. This effect is called pile-up and is a challenge for the experiments installed at LHC, because they detect about 23 events within the same bunch-crossing.
Fig. 2.1 shows the cross sections of pp-interactions [DEN90]: QCD processes are dominating. Events containing new physics are very rare: under the assumption that the Higgs-Boson has a mass of 150 GeV, one proton-proton interaction out of $10^{10}$ creates this particle. This fact explains why the LHC needs to provide such a high luminosity. In addition this fact forces the experiments, which are constructed at LHC and are dedicated to discover new physics, to apply a restrictive but efficient online event selection (Trigger). This is crucial in order not to store an enormous amount of events without discovery potential, which would lead to the storage of non-manageable and non-analyzable data volumes.

[1] A proton anti-proton collider operating at Fermilab [WEB04].

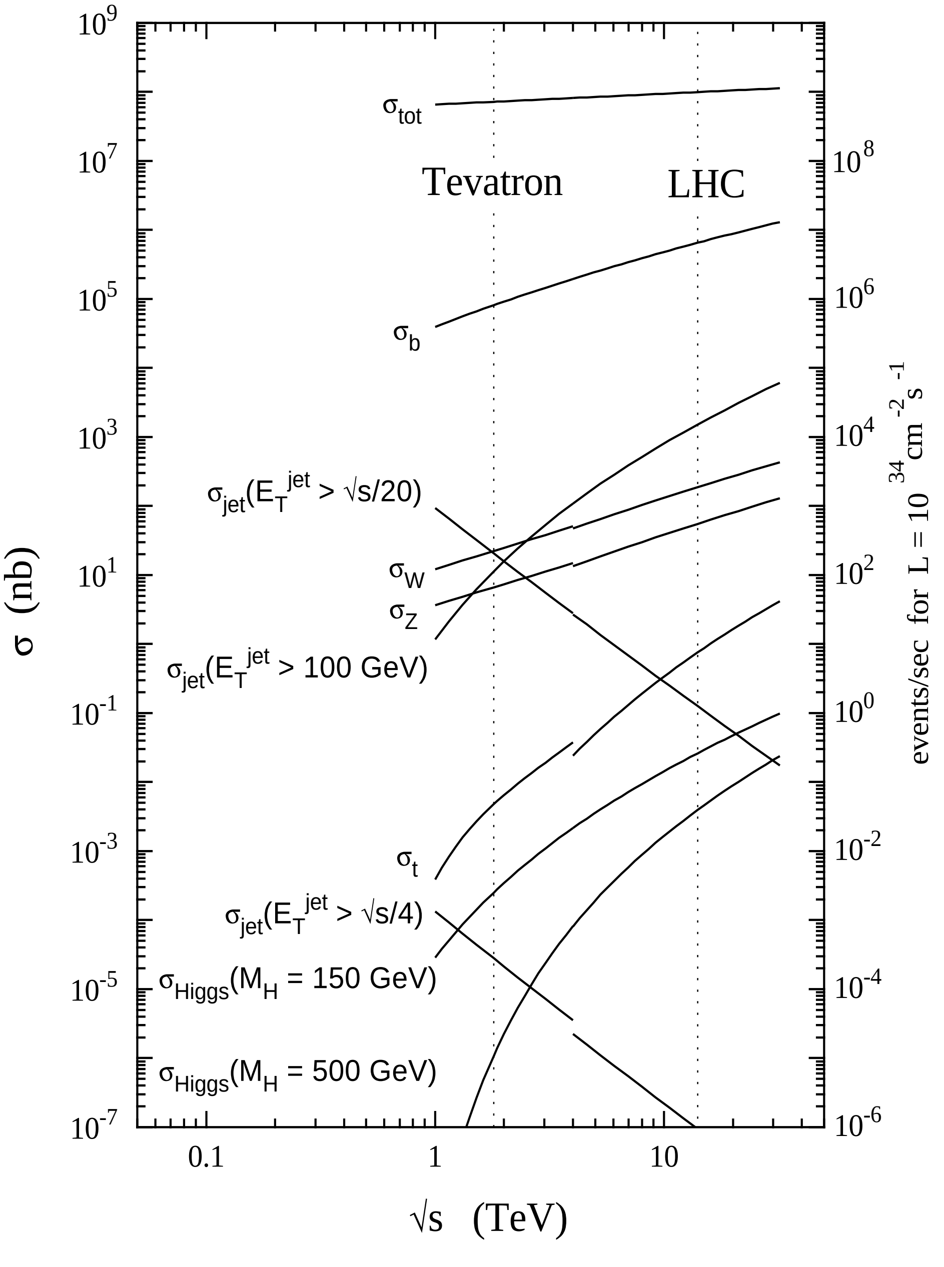


Figure 2.1: The cross sections of proton-proton interactions. The gap of the cross sections between the Tevatron energies and the LHC energies are caused by slight differences in the cross-sections of p$\bar{p}$ and pp collisions

# Chapter 3

# The ATLAS Physics Goals

In the list below, the main physics goals of ATLAS are mentioned with a brief description of the theoretical motivations behind them [WEB06].

- Higgs discovery: the existence of the Higgs Boson is motivated by the electro-weak symmetry breaking. The fundamental question is why the W and Z bosons are massive, while photons are massless. The Higgs mechanism, which postulates the existence of the Higgs boson, is an explanation for this phenomenon.

- SUSY discovery: Super-symmetry is a theoretical extension of the Standard Model. For every elementary particle $p$ with spin $s$ it postulates a partner particle $\tilde{p}$ with spin $s - \frac{1}{2}$, a fermion to every boson and a boson to every fermion. This extension of the Standard Model is mainly motivated by two reasons: the first motivation is, that without the existence of SUSY particles the Higgs mass would diverge. The second motivation is, that the coupling constants for the electroweak, the strong and the gravitational forces do not converge on high energy scales without the existence of super-symmetric particles.

- Collecting samples of B-hadron decays. Studying the decays of the $B^0$ allows to determine the angles of the unitarity triangle, describing the CP-violation. The $B^0$ production is not an exclusive feature of the LHC, B-factories are currently operational e.g. at SLAC. However, due to the high luminosity of the accelerator LHC, a multi-purpose detector could collect a huge amount of B-decay samples. In addition some heavy B-mesons e.g. $B_S^0$ are exclusively produced at LHC and Tevatron.

Due to these physics motivations, the multi-purpose detector should be designed in such a way, that it is sensitive to the decay products of the Higgs,

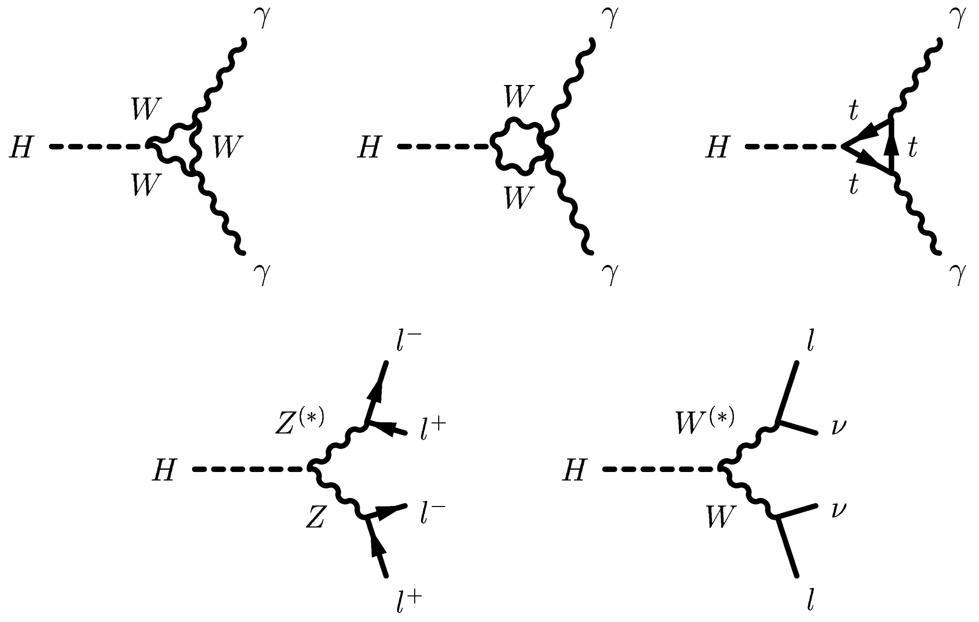


Figure 3.1: Feynman diagrams for decay channels of the Standard Model Higgs boson. The lepton $l$ is either an electron, muon or tau. The observation of hadronic decays of the Higgs boson in the ATLAS detector is extremely challenging due to the irreducible QCD background

SUSY particles and B-hadrons.

## 3.1 Higgs Discovery

The Feynman diagrams in figure 3.1 show the most promising decay channels of the Higgs particle, which may be produced via various mechanisms at LHC proton-proton collisions [DEN90].

Studying these five decay channels, one can derive a big part of the requirements on the ATLAS detector. The branching ratios of the different channels depend on the Higgs mass [ATL99].

- In order to detect the three $H \to \gamma\gamma$ decay channels, electromagnetic calorimetry is needed. The electromagnetic calorimeter must have a high resolution, in order to be able to distinguish the electromagnetic showers caused by the Higgs decay photons from the showers caused by $\pi^0 \to \gamma\gamma$ decays (QCD background).

- The fourth diagram represents various leptonic decay channels, e.g. $H \to eeee$, $H \to \mu\mu ee$ and $H \to \mu\mu\mu\mu$. In order to detect the first channel, excellent electromagnetic calorimetry is needed again. In addition, track identification is needed to be able to distinguish electromagnetic showers caused by electrons from those caused by photons. As in the two other channels decay products are muons, muon spectroscopy is required.

- The last diagram shows neutrinos among the decay products: $H \to e\nu_e e\nu_e$, $H \to \mu\nu_\mu e\nu_e$ and $H \to \mu\nu_\mu\mu\nu_\mu$. The neutrinos are not seen by the detector, but they carry unobserved energy. Therefore a good overall calorimetry is needed (electromagnetic and hadronic) to identify missing transverse energy $E_T$.

## 3.2 SUSY Discovery

Two possible SUSY decays are drawn in Fig. 3.2. These decays are possible under the assumption that the R-parity[1] is conserved, which means that if super-symmetric particles decay, the lightest super-symmetric particle is always produced at the end of the decay chain. Under the assumption that the lightest super-symmetric particle is a neutralino $\chi_1^0$, it will carry out energy, leaving the detector without interacting. So the detection of missing $E_T$ will be the most important signature for the discovery of SUSY particles. This means that a hermetic calorimetry is the key to the discovery of SUSY particles.

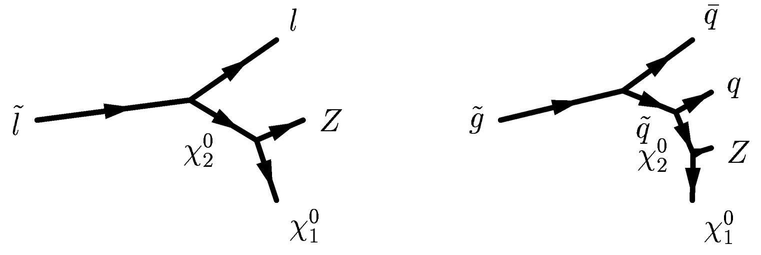


Figure 3.2: Two possible SUSY decays in ATLAS in case of R-parity conservation: a simple slepton decay into a lepton, a Z and the lightest super-symmetric particle $\chi_1^0$ and a cascade decay of a gluino into quarks (jets), a Z and a $\chi_1^0$

[1]R-parity is a parity operation on the number of SUSY particles. An interaction involving an odd number of SUSY particles always has to result in an odd number of SUSY particles, an interaction involving an even number of SUSY particles always has to result in a even number of SUSY particles (0 included).

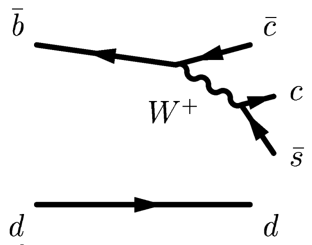


Figure 3.3: $B^0$ decaying into a $J/\Psi$ and a $K_S^0$

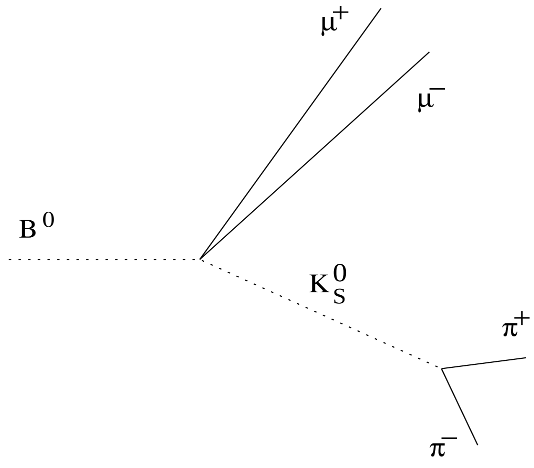


Figure 3.4: Tracks of a $B^0$ decay to $J/\psi \to \mu^+\mu^-$ and $K_S^0 \to \pi^+\pi^-$

## 3.3 B-Physics

In Fig. 3.3 an example of the decay of a $B_d^0$ is shown. The decay pattern in the detector is drawn in Fig. 3.4. In order to be sensitive to such a decay pattern and to be able to identify the flavor of the B-meson at production time, track reconstruction close to the beam-pipe is crucial. This is one of the reasons why an inner tracking with vertex detection is required.

## 3.4 Detector Requirements

The incomplete set of examples above and other physics goals described in [ATL99] lead to the following requirements on the layout of ATLAS as a multi-purpose experiment for the LHC:

- Very good electromagnetic calorimetry for electron and photon identification and measurements, complemented by full-coverage hadronic calorimetry for accurate jet and missing transverse energy measurements.

- High-precision muon momentum measurements, with the capability to guarantee accurate measurements at the highest luminosity using the external muon spectrometer alone.

- Efficient tracking at high luminosity for high $p_T$ lepton-momentum measurements, electron-photon identification, $\tau$-lepton and heavy flavor identification (e.g. b jet tagging), and full event reconstruction capability at low luminosity.

- Almost full azimuthal angle detection coverage and a large polar angle coverage

- Triggering and measurement of particles at low $p_T$ threshold, providing high efficiencies for most physics processes at LHC.

# Chapter 4

# The ATLAS Detector

The layout of the ATLAS detector (Fig. 4.1) tries to fulfill the requirements which were explained in the previous chapter. The main reference to the ATLAS detector layout is [ATL99]. The detector can be split into the magnet system and three detector subsystems:

- the inner detector (tracker)
- the calorimeter (electromagnetic and hadronic)
- the muon spectrometer

## 4.1 The Magnet System

The magnet system consists of three superconducting components: the central solenoid, the end cap toroids and the barrel toroid. The central solenoid provides the magnetic field for the inner detector, with a central field of 2.0 T and a peak field of 2.6 T. The solenoid is placed inside the calorimeters due to cost constraints. However, it is problematic to put the magnet inside the calorimeter because the introduced material may compromise the calorimeter performance. This is the reason why the central solenoid and the LAr calorimeter are housed in the same vacuum vessel, which allows two vacuum walls to be saved. The barrel toroid provides the magnetic field for the muon spectrometer in the barrel region, with a peak field of 3.9 T. It consists of three toroids of eight coils each. The coils are housed in individual cryostats and therefore the support structure of the cryostat has to take up the forces between the toroids. The end cap toroids provide the magnetic field for the muon spectrometers in the end cap region with a peak field of 4.1 T. They are housed in one single cryostat per end cap and therefore the cold support

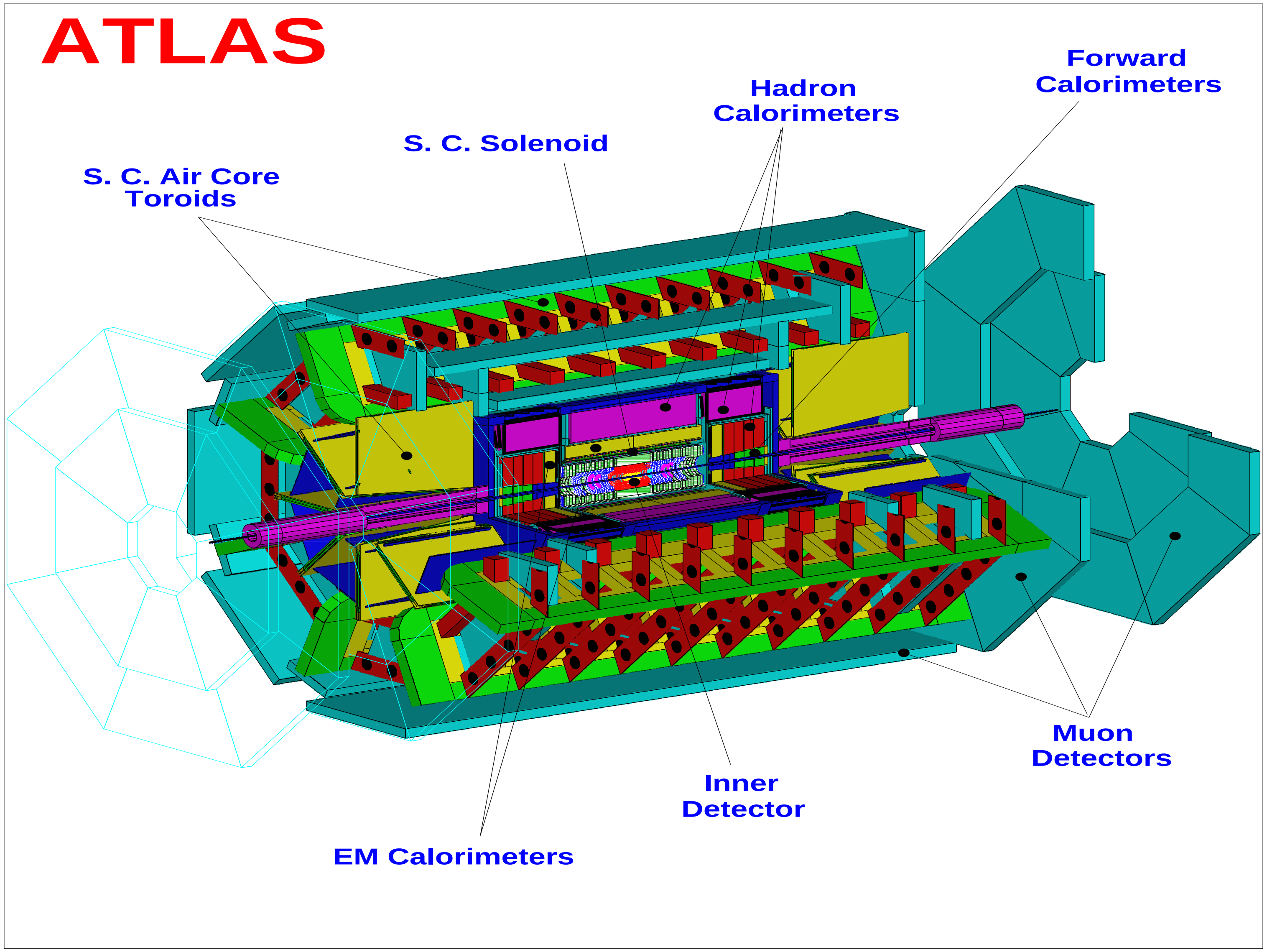


Figure 4.1: The ATLAS detector

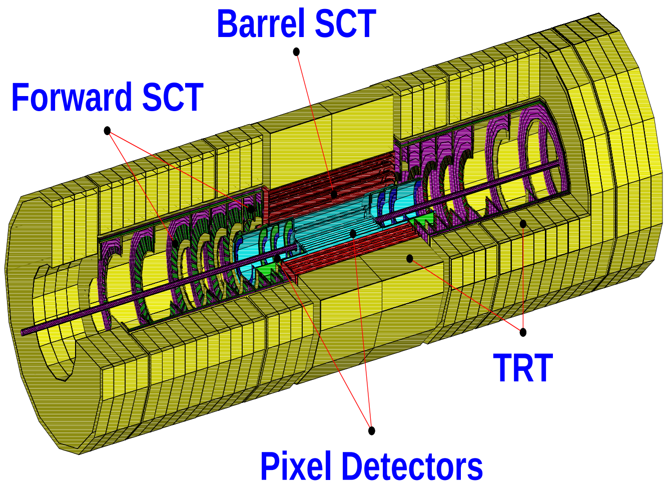


Figure 4.2: The ATLAS inner detector

structure has to take up the forces between the eight coils which are assembled radially and symmetrically around the beam axis. The magnet system is cooled with the forced flow of 4.5 K helium.

## 4.2 The Inner Detector

The purpose of the inner detector is tracking for lepton-momentum measurements, electron-photon identification and $\tau$ and heavy flavor identification e.g. b-tagging. A schematic view of the inner detector is drawn in Fig. 4.2.

The resolution of the inner-detector must be very high close to the vertex area and can be a bit less precise at larger radii. The constraints are that as

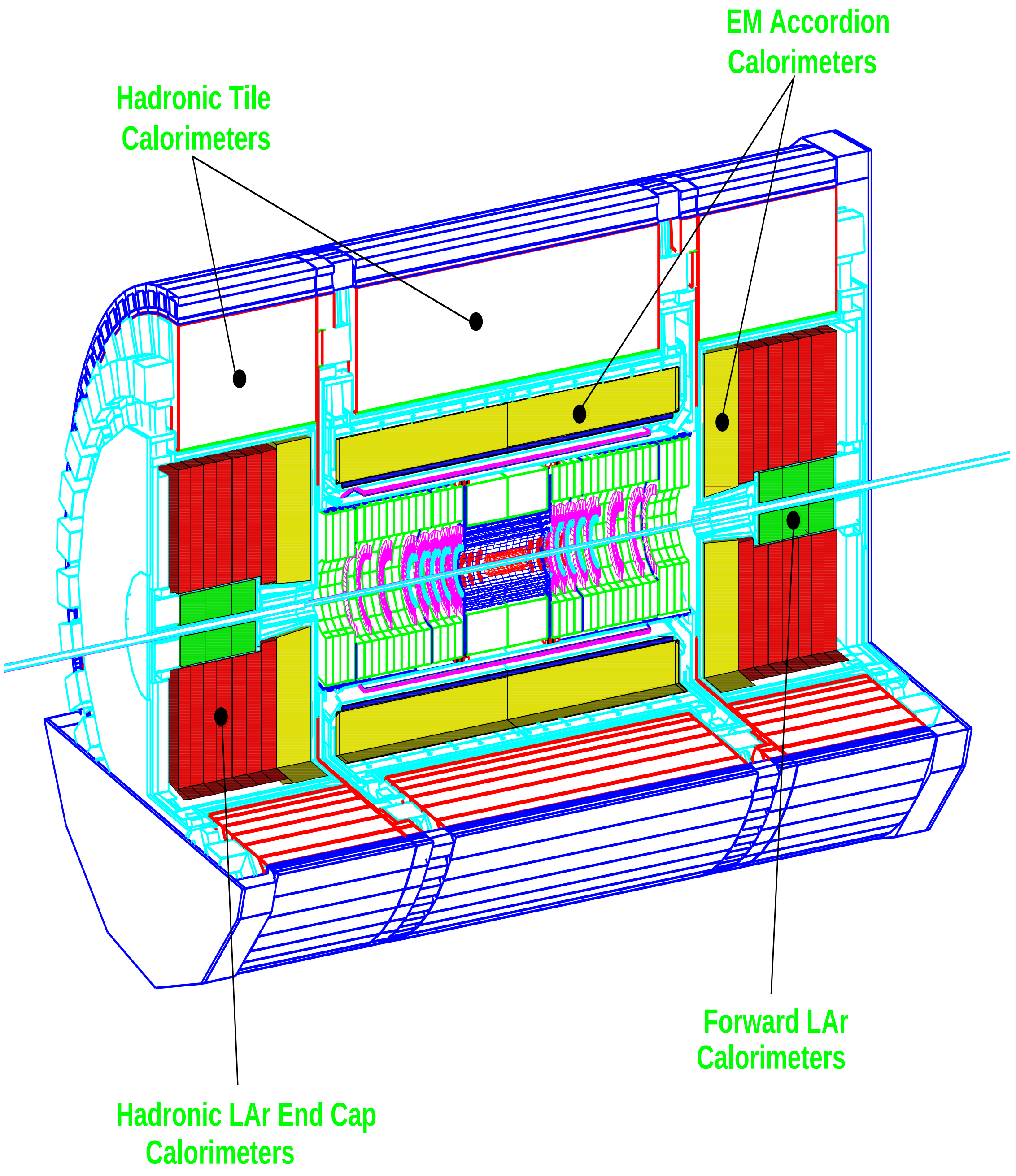
little material as possible should be put in front of the calorimeter and that the inner tracker must cope with a very harsh radiation environment. ATLAS has chosen a combination of high resolution detectors at the inner radii and continuous tracking elements at the outer radii. Close to the vertex region three layers of a semiconductor pixel detector with high granularity are installed to identify displaced vertices originating from decays of short-lived particles like B-mesons and taus. Around the pixel detector the semiconductor tracker consisting of silicon micro-strips will be installed. It provides four precision space points per track which allows track finding and momentum measurements. The third layer of the inner detector is the straw tube tracker (TRT). It provides continuous track following with 36 points per track, while introducing only a small amount of material into the detector. In addition, the TRT improves the electron identification of the detector by the detection of transition-radiation photons in the xeon gas of the tubes.
The resolution achieved by the pixel detector, as measured in the ATLAS standard coordinate system[1], is 12 $\mu$m in R$\phi$ and 66$\mu$m in z, respectively 77 $\mu$m in R for the end-cap disks. The SCT provides a resolution of 16 $\mu$m in R$\phi$ and 580 $\mu$m in z respectively in R for the end-cap wheels. The TRT will achieve a resolution of 170 $\mu$m.

## 4.3 The Calorimeters

The purpose of the calorimeter is the measurement of hadronic and electromagnetic showers and the measurement of missing transverse energy $E_T$, especially for SUSY discovery. The calorimeter system consists of an inner electromagnetic and an outer hadronic calorimeter (see Fig. 4.3).
The electromagnetic calorimeter is built with accordion-shaped lead absorbers and liquid Argon (LAr) scintillating material in the barrel region and in the end-cap region. The calorimeter consists of three layers with different granularities: the innermost layer is called preshower detector and designed for enhancing the particle identification ($\gamma/\pi^0$, $e/\pi$ separation). In the low pseudo-rapidity[2] region $|\eta| < 2.5$ its granularity is $\Delta\eta \times \Delta\phi = 0.003 \times 0.01$. The middle and outside layers provide a coarser granularity only ($0.025 \times 0.25$ and $0.05 \times 0.025$), but fulfill the requirements for electromagnetic jet reconstruction and missing energy measurements. In the high pseudo-rapidity region $|\eta| > 2.5$ the electromagnetic calorimeter consists of two layers only,

[1]R is radius in the plane perpendicular to the beam axis; $\phi$ is the azimuthal angle around the beam axis; z is the beam axis

[2]The pseudo-rapidity is defined as $\eta = -\ln \tan \frac{\theta}{2}$, where $\theta$ is the polar angle from the z direction



EM Accordion Calorimeters
Hadronic Tile Calorimeters
Forward LAr Calorimeters
Hadronic LAr End Cap Calorimeters

Figure 4.3: The ATLAS calorimeter system

both providing a resolution of $0.1 \times 0.1$.
The hadronic calorimeter is split into different subsystems applying different technologies. In the barrel region a tile calorimeter will be installed with iron absorbers and plastic scintillator as active material. The hadronic calorimeter in the end-cap will be based on copper absorbers and LAr as active material. A particularly challenging detector is the forward calorimeter close to the beam pipe, due to the high radiation level in this region. The absorber material in the first of the three FCAL sections is copper, in the other two sections tungsten. LAr serves as sensitive medium in the whole FCAL.
The thickness of the hadron calorimeter is an important parameter, because the calorimeter should contain the hadronic shower well in order to provide an adequate missing $E_T$ measurement and to avoid punch-through to the muon system. A thickness of 11 $X_0$ (hadronic interaction lengths) was chosen to meet the containment requirements.

## 4.4 The Muon Spectrometer

The purpose of the muon spectrometer is the high-precision muon momentum measurement and to provide fast trigger information on muon tracks for the Level 1 trigger. Therefore the muon spectrometer is split into four parts: in a trigger and a detection system, each in the barrel and end-cap region (see Fig. 4.4). The spectrometry is based on the magnetic deflection of muon tracks. The necessary magnetic field is provided by the barrel toroid which provides a field which is mostly orthogonal to the muon tracks.
In the barrel region the muon spectrometer is designed as three cylinders, called stations. In the inner-most station 2x4 sensitive layers of Cathode Strip Chambers (CSC) will be installed, the two outer stations are based on 2x3 layers of Monitored Drift Tubes (MDT). CSCs are MWPCs with segmentation and read-out of the cathode strips. The trigger system consists of Resistive Plate Chambers (RPCs). The chambers are mounted on both sides of the inner MDT station and inside the outer MDT station. The end-cap spectrometer applies four disks of MDTs as trackers and Thin Gap Chambers (TGC) as trigger elements. TGCs are similar to CSCs, but the anode wire pitch is larger than the cathode-anode distance.

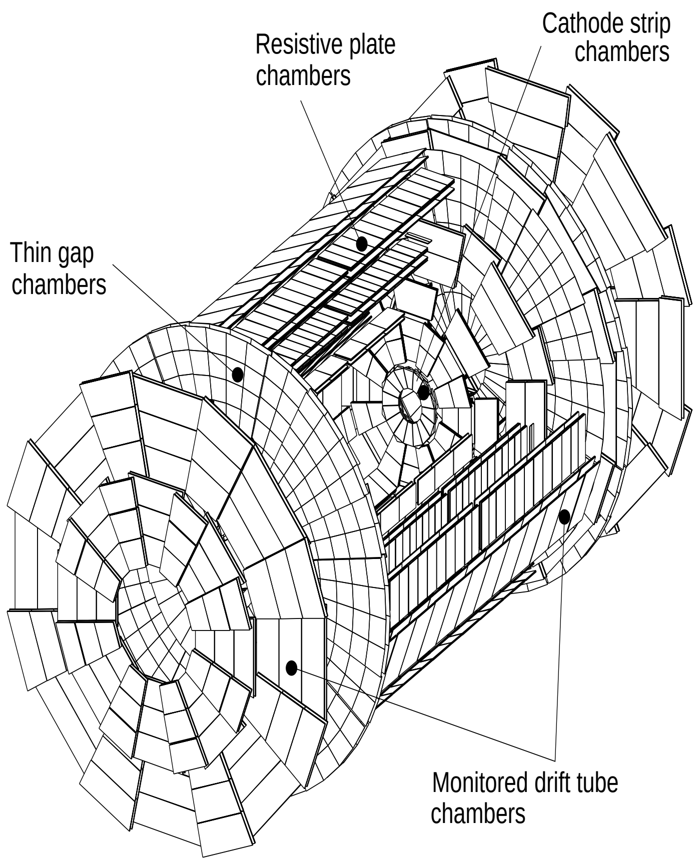


Figure 4.4: The ATLAS muon spectrometer

# Chapter 5

# The ATLAS Trigger and Data Acquisition System

The ATLAS Trigger and Data Acquisition System (TDAQ) can be divided in two tightly coupled parts: the Trigger system and the Data Acquisition (DAQ). The Trigger system selects events at three levels: Level 1, Level 2 and Event-filter. With this strategy the ATLAS experiment plans to handle the enormous background of non-interesting physics related to minimum bias and QCD related processes, which occur with every bunch crossing at LHC (see chapter 2). The interaction rate of $\sim 10^9$ per second (the LHC bunch crossing rate is 4x$10^7$ per second, but about 23 proton-proton interactions happen per bunch crossing) is cut down by the three trigger levels to $\sim 10^2$ per second, selecting those interactions which are most promising to detect new physics.
The DAQ system must provide event data to the Level 2 trigger and the Event-filter and must make the data available to the mass storage system. Therefore it is mainly defined by the needs of the trigger system.

## 5.1 Level 1 Trigger

The Level 1 Trigger runs at the bunch crossing rate of 40 MHz and has to take its decisions in less than 2.5 $\mu$s. It bases its decision on the muon and the calorimeter system of the ATLAS detector. Simple but fast selection algorithms are executed by custom-built hardware. The calorimeter trigger processor can identify various $p_T$ thresholds of electromagnetic clusters, jets and missing transverse energy. The muon trigger identifies $p_T$ thresholds of muons. For both the muon and the calorimeter trigger only coarse information is available. The central trigger processor compares these results with a global trigger menu. The Level 1 trigger does not only provide a binary decision: the calorimeter and muon trigger processors communicate to the Region of Interest builder (RoIB) those detector elements which contain interesting signatures. The RoIB collects this information and forwards it to the Level 2 trigger.

## 5.2 Level 2 Trigger

The Level 2 trigger generally takes its decision based on subsets of the event data, the so-called Regions of Interest (RoIs). It receives the description of the RoIs from the RoIB. Based on this information, the Level 2 is able to request the subset of the event data it needs to take its decision. In a first step Level 2 verifies the Level 1 decision, by using the fine-grained information (event data) of the calorimeters and the muon spectrometer. If this confirmation succeeds, more complex processing may take place such as shower shape analysis, track finding and track matching. For every subsequent step Level 2 can request the necessary event data e.g. from the inner detector. As the complete event data is accessible for Level 2, even a full subdetector scan may happen on this trigger level in extreme cases. The Level 2 trigger is implemented as software algorithms running on farms of custom PCs (a few hundred) running Linux. The algorithms will be built upon the same software framework as the offline reconstruction in order to allow consistency checks.

## 5.3 Event-Filter

After a positive Level 2 decision, the Event-filter receives completely built events and therefore can base its decision on the full event information. Like Level 2, the Event-filter is implemented as software algorithms running on farms of custom PCs (many hundred) and, also like Level 2, the algorithms will be built upon the same framework as the offline reconstruction.

## 5.4 Trigger Rates

The architecture of the trigger system and trigger rates on the different levels define the performance requirements on the DAQ system as the transport infrastructure. The data supply for the Level 2 trigger must work at Level 1 accept rate and the data supply for the Event-filter must work at the Level 2 accept rate. The working values are an accept rate of 100 kHz for Level 1

(technical limitation of the Level 1 system) and an accept rate on Level 2 of 3 kHz. These rates are estimates and still under study, but they are sufficiently accurate for the initial design of the DAQ.
However, in later phases of the experiment the trigger strategy may change, either due to new physics needs or due to an unsatisfactory Level 2 efficiency for certain physics processes. With a different trigger strategy the trigger rates may change significantly: the Level 2 trigger may become more open and the Event-filter rate may increase by a large factor.

## 5.5 The Data Acquisition System

The architecture of the DAQ system is to a large extent defined by the trigger scheme. The DAQ needs to provide event data for the Level 2 Trigger, the Event-filter and finally for mass-storage. An overview of the whole TDAQ system is drawn in Fig. 5.1.

Event data is pushed into the pipeline memories of the on-detector electronics at the LHC bunch-crossing rate. When the Level 1 accepts a bunch-crossing, the data is pushed to the Read-out Drivers (ROD), which are off-detector. Then the data is sent via the Read-out Links (ROLs) to the Read-out Buffers (ROBs), which are housed by the Read-out Subsystem (ROS). Simultaneously the RoIB is processing the Level 1 information and sending a RoI description to the Level 2 Supervisor.
The Level 2 Supervisor assigns this event to a processor in one of the Level 2 farms, a Level 2 Processing Unit. Via the switching network the Level 2 Processing Unit requests and receives the RoI data it needs for the processing. It takes a decision and reports it to the Level 2 Supervisor. None of the raw event data is output from the Level 2 Processing Unit.
The Level 2 Supervisor forwards the decision to the Dataflow Manager. In case of a Level 2 reject, the Data-flow Manager issues a clear to all ROSs via the switching network. In case of a Level 2 accept, the Dataflow Manager assigns the event to an Event-builder node. The Event-builder node asks for the data fragment of the assigned event from all ROS, builds the event and sends it on request to an Event-filter farm. In addition is notifies the Data-flow Manager that the event is fully built, which then can be cleared from the ROS.
In case the event is rejected by the Event-filter, it is deleted. In case it is accepted, it is sent to the Event-storage node, which makes it available to the mass storage system.
The dashed Boxes in Fig 5.1 contain all the elements of the Dataflow system. The DataCollection subsystem comprises all the elements of the Dataflow

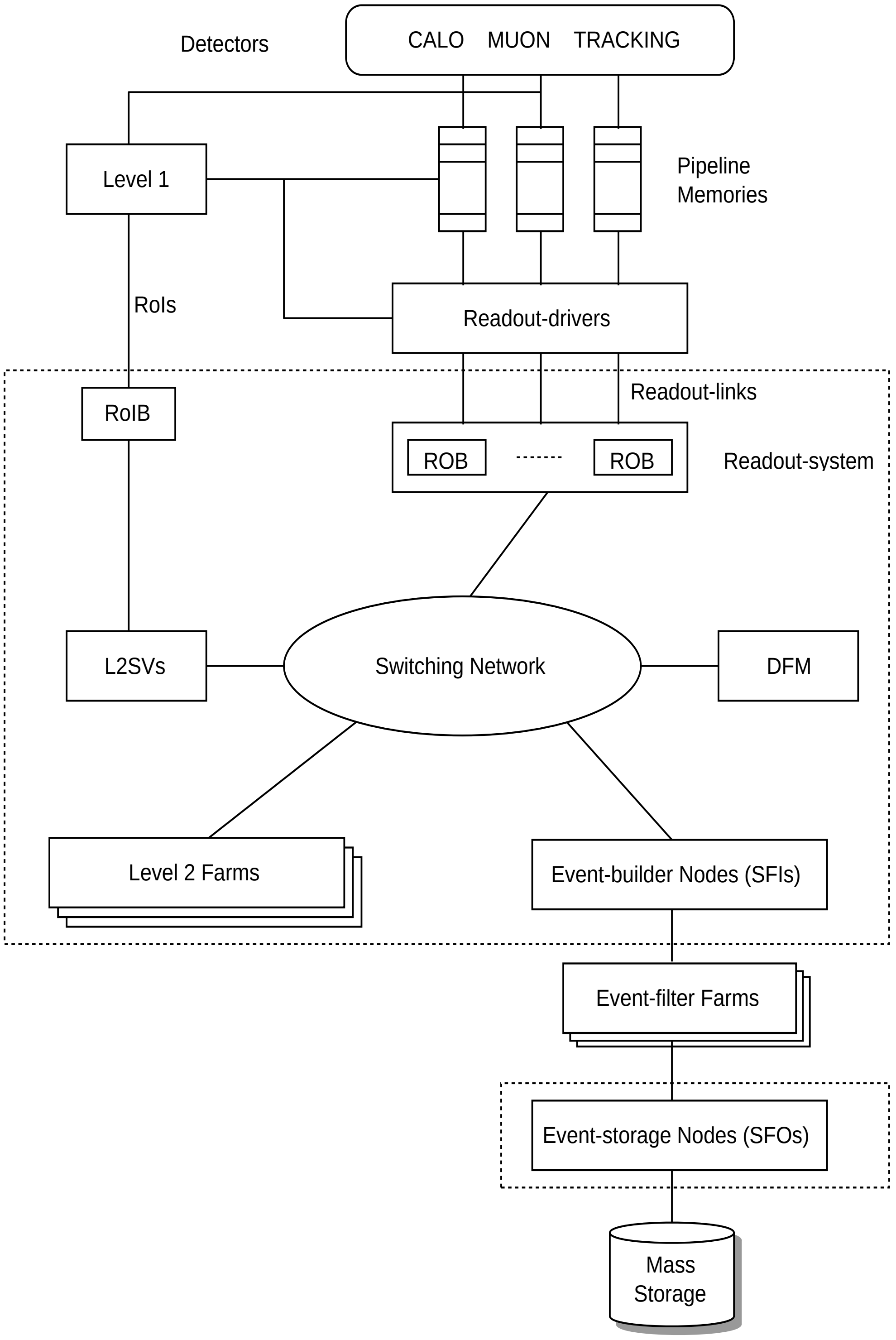


Figure 5.1: The architecture of the ATLAS TDAQ system (the acronyms can be found in the glossary). All the elements inside the dashed boxes are parts of the Dataflow system.

system except the ROS and the hardware part of the RoIB. The mandate of DataCollection is

- to transport and to make available parts of the events (Regions of Interest) to the Level 2 trigger
- to build the Level 2 accepted events completely and to make those events accessible to the Event-filter
- to make Event-filter accepted events available to mass storage

## 5.6 Requirements on Data Collection

Based on the trigger rates in the early phase of ATLAS, there are two kinds of performance requirements DataCollection has to fulfill. The first are rate requirements (RoI collection rate and EB rate). The second kind are bandwidth requirements (total Level 2 input and total event builder throughput).
The Level 1 rate is 100 kHz and the average RoI size is 16 kB. Therefore the total input that Level 2 DataCollection must provide is 1.6 GB/s. As the events are stored in the ROBs during the Level 2 processing, none of the raw event data is output from Level 2.
The EB rate is 3 kHz and the average event size is 1.2 MB. Therefore the total throughput of the Event Builder is 3.6 GB/s.
An overview on the system parameters is given in the second chapter of [ATL03].

# Chapter 6

# The DataCollection Subsystem

As explained in the previous chapter, the DataCollection subsystem transports regions-of-interest (RoI) data from the ROS to the Level 2 trigger and full events from the ROS to the Event-filter. In addition it makes Event-filter accepted events available to mass storage. This functionality is distributed among six software applications running on commodity PCs operated by Linux and interconnected with a Gigabit Ethernet local area network (LAN). A common approach to design and implementation was chosen, which leads to the design and implementation of a common DataCollection software framework, providing a suite of common services (e.g. Message-passing and Application control). The design and the implementation of the DataCollection framework is based on C++ and the standard template library (STL) [JOS99]. The DataCollection applications are multi-threaded and built on top of the common framework.

## 6.1 DataCollection Applications

The DataCollection functionality is distributed among six software applications: the Level 2 supervisor (L2SV), the Level 2 processing unit (L2PU), the pseudo ROS (pROS), the Dataflow manager (DFM), the Subfarm-input (SFI) and the Subfarm-output (SFO).

### 6.1.1 Level 2 Supervisor

The L2SV receives a Level 1 decision containing geometry information of RoIs from the RoIB. It distributes events to be processed within a part of the Level 2 farm consisting of many L2PUs by assigning events to L2PUs chosen according to a load-balancing algorithm. After processing at the L2PU the L2SV receives the Level 2 decisions, which it forwards to the DFM. The role of the L2SV is described more precisely in [DC-09]. There will be $\sim$ 10 L2SVs in the TDAQ system.

### 6.1.2 Level 2 Processing Unit

The L2PU receives a Level 1 decision from the L2SV. It requests the RoI data it needs for processing from multiple ROSs. After the Level 2 decision is taken, it reports it to the L2SV. For accepted events it additionally sends a more detailed Level 2 result to the pROS.

The DataCollection framework provides the basic services needed by the Level 2 algorithms. The decision-taking Level 2 algorithms are beyond the scope of the DataCollection software, but the L2PU builds the basis for them. A more precise description of the interface between the Level 2 algorithms and the DataCollection subsystem is given below and in [DC-19]. There will be in the order of 200 L2PUs in the TDAQ system.

### 6.1.3 Pseudo ROS

The pROS is the logical interface between the Level 2 and the Event-filter. It receives detailed Level 2 results from the L2PUs and takes part in the EB as a common ROS. It ensures that the detailed Level 2 result becomes a part of the fully built event and therefore is accessible for the algorithms running in the Event-filter. The detailed requirements and the design of the pROS are described in [DC-34] and [DC-41]. There will be one pROS in the TDAQ system.

### 6.1.4 Dataflow Manager

The DFM is responsible for the load balancing and the bookkeeping for the EB. It receives the Level 2 decisions from the L2SV and forwards the Level 2 rejects to all instances of the ROS to ensure the deletion of these events. For every Level 2 accept the DFM assigns a Subfarm-input (SFI) for EB. As soon as an event is fully built, the DFM is notified by the SFI in charge. It then deletes the event from the ROS. The task of the DFM is described more precisely in [DC-10]. There will be one DFM in the TDAQ system.

### 6.1.5 Subfarm-input

The SFI fulfills the major part of the EB functionality and therefore could also be called the Event-builder node. It gets an event assigned by the DFM and requests the event fragments from all instances of the ROS. As soon as all fragments of a given event have arrived at the SFI, it notifies the DFM. The fully built event is stored and made accessible to the Event-filter. As soon as the transfer of an event to the Event-filter is completed, it is deleted from the SFI memory. The role of the SFI in the system is more precisely described in [DC-16]. Limited by the Gigabit Ethernet bandwidth, there will be 25 SFIs per kHz of EB rate (every SFI contributing 40 Hz), so 75 instances will be needed to reach the initial 3 kHz EB rate.

### 6.1.6 Subfarm-output

The SFO receives events accepted by the Event-filter. It buffers them in files held on a local disk. These files are then available to be transfered to the mass storage system. The requirements on the SFO are specified in [DC-38]. There will be in the order of 20 SFOs in the TDAQ system.

## 6.2 Interaction between the components

The interactions between the components of the DataCollection are exclusively transmitted via the network. The whole of these interactions is called message flow and described in [DC-12].

The message flow is shown in Fig. 6.1 and explained in the text below following the numbering scheme in the Figure. The convention follows the sequence of interactions in the running system, which is shown in Fig. 6.2. The indicated message rates are calculated for the whole system and not for single instances of the applications.

1. L2SV $\rightarrow$ L2PU: A Level 1 result containing the RoI information is sent to a L2PU. This information allows the Level 2 processing on an event to start. This message occurs at the Level 1 accept rate of 100 kHz.

2. L2PU $\rightarrow$ ROS: Requests for the RoI data, which is needed by the running Level 2 algorithm, are sent to the corresponding ROSs. This message occurs at a rate of 800 or 1600 kHz, depending on an open architectural choice (see section 6.3.1).

3. ROS $\rightarrow$ L2PU: The RoI data (fragments of the full event data) is sent back to the L2PU. The rate of this message is the same as the rate of

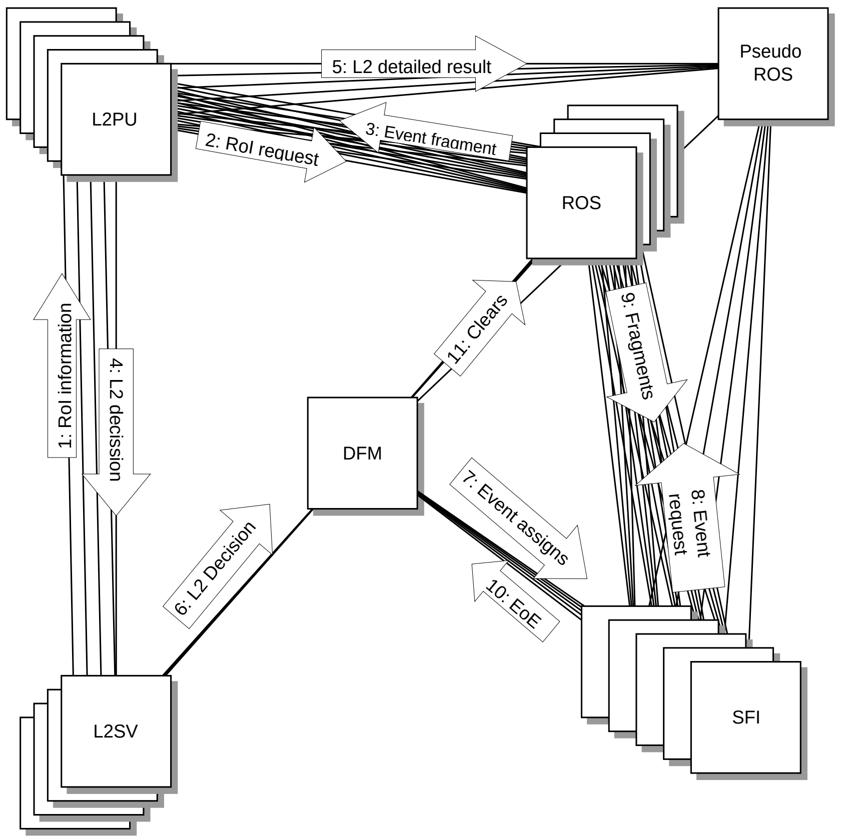


Figure 6.1: Message flow. This figure should be mapped on Fig. 5.1 showing the TDAQ architecture. The Event-filter and the SFO is missing in this figure, because these components do not take part in the message flow, but communicate via a specific Event-filter input/output protocol [DC-35].

the RoI requests.

4. L2PU → L2SV: The Level 2 decision including additional information like the Level 1 trigger type and the Level 2 trigger type is sent to the L2SV. This message occurs at the Level 1 accept rate of 100 kHz as well.

5. L2PU → pROS: The detailed Level 2 result, containing information on how the decision was derived, is sent to the pROS. This interaction happens at the Level 2 accept rate of around 3 kHz.

6. L2SV → DFM: The L2SV collects a group of Level 2 decisions and forwards them to the DFM. This grouping strategy is applied to reduce the message rate at the level of the applications and in the network.

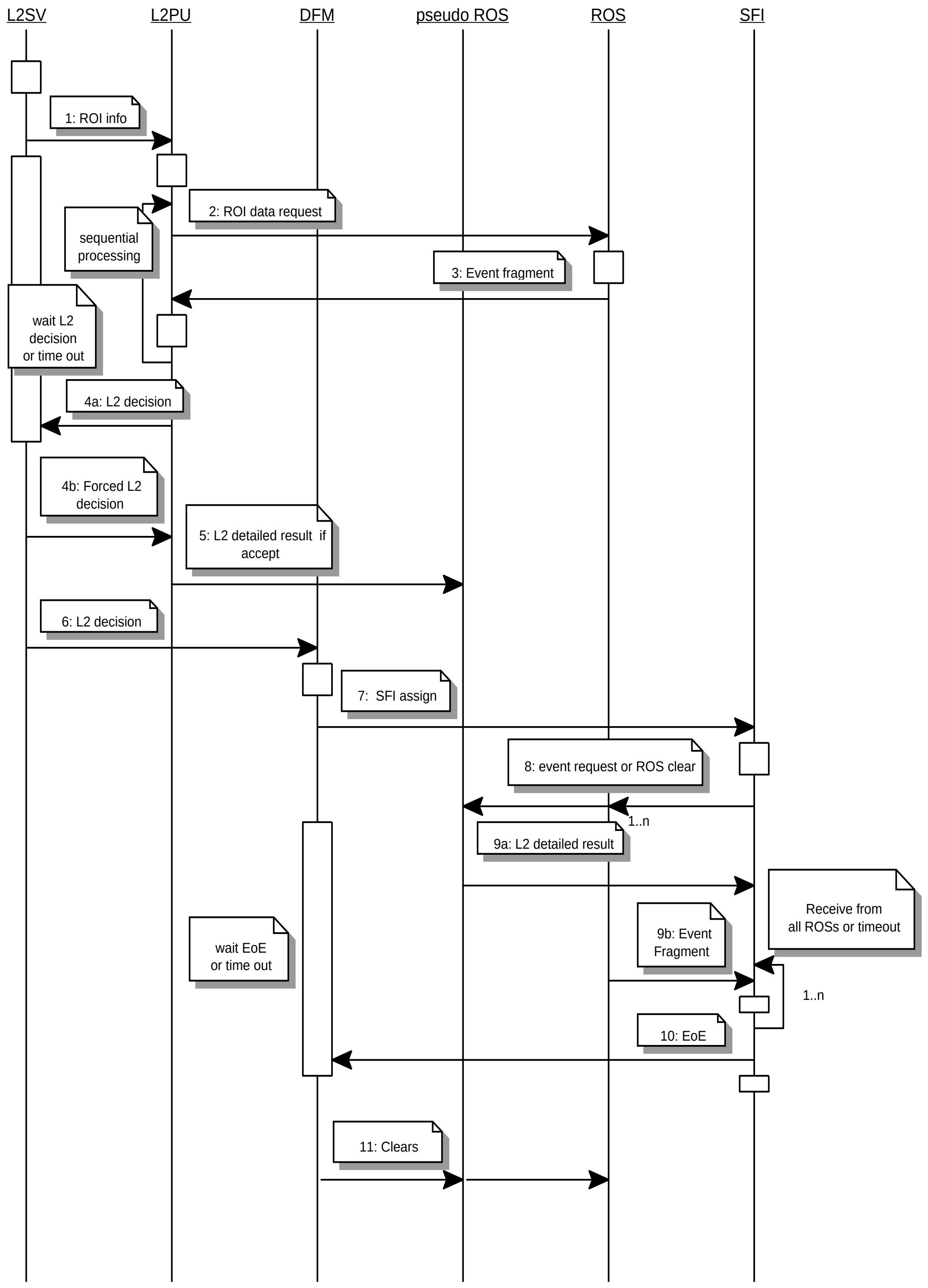


Figure 6.2: Sequence of the message flow. L2 stands for Level 2.

With an assumed grouping of around 300 decisions, the rate of this message is 330 Hz.

7. DFM → SFI: The assignment of an event to be built is sent to an SFI. This message occurs at the Level 2 accept rate of 3 kHz.

8. SFI → ROS(/pROS): Requests for the event data are sent to the ROSs (and the pROS). This message occurs at a rate of 432 or 4800 kHz, depending on an open architectural choice (see section 6.3.1).

9. ROS(/pROS) → SFI: Event data flows into the SFI. The rate of this message is the same as the rate of the requests.

10. SFI → DFM: An "End of Event" (EoE) message is sent to the DFM after an event is completely built. This message-rate is the same as the Level 2 accept rate of 3 kHz.

11. DFM → ROS: A message containing the identifiers of the events to be deleted is sent the ROS (in case of a negative Level 2 decision or after the EoE message arrived). For this interaction the strategy of grouping to reduce the message rate is applied, as for the communication between the L2SV and the DFM. With a grouping of 300 the rate is reduced to 330 Hz.

Note that the message flow between the ROS and the DataCollection involves event data messages and that the traffic pattern is request-reply: for every data request message issued by a L2PU or an SFI and sent to a ROS, an event fragment message travels in the opposite direction through the network. This traffic pattern is an important paradigm in the DataCollection architecture as it allows the management of the traffic flowing through the network with a very high granularity. This issue is called traffic shaping and is of major importance and one of the main achievements of the author's EB studies. It will be discussed in section 9.3.

## 6.3 Interfaces

The DataCollection subsystem interfaces to four neighboring systems:

- the Read-out Subsystem (ROS), where it obtains the event data from
- the High Level Trigger infrastructure in the L2PU, which requests RoI data

- the Event-filter, which takes full events from the SFI and returns a reduced number of them to the SFO
- the Online Software (OnlineSW), which provides infrastructure for run control, system monitoring, event monitoring, configuration databases and other common services of the ATLAS TDAQ system.

### 6.3.1 Read-out Subystem

The ROS receives detector data via the Read-out Link (ROL) and buffers it in the ROBs. It makes the buffered data available for the RoI collection (for the Level 2 trigger) and for EB. In case an event is rejected by Level 2 or after it is fully built if it is accepted by Level 2, the ROS is allowed to release it from the ROBs, following a clear message issued by DataCollection. The role of the ROS in the TDAQ system is more precisely described in [CRA02]. The ROS has been and still is the subject of a controversial discussion in the ATLAS TDAQ community, as there are two main possibilities how the ROLs (via the ROBs) could be connected to the DataCollection subsystem (for RoI collection and EB):

- Bus-based ROS: A few ROBs (12 is the current base-line assumption for the TDR [ATL03]) are plugged on a PCI Bus of an industrial PC and the data can be sent to DataCollection via the Network interface card (NIC) of this PC. This approach requires complex and performant software running on the PC. A schematic view on the bus-based ROS is shown in Fig. 6.3.
- Switch-based ROS: The ROBs can be read out directly via Fast or Gigabit Ethernet switching network. The role of the ROS PC is restricted to powering the ROB and to configuration and monitoring. A schematic view on the switch-based ROS is shown in Fig. 6.4.

In the bus-based ROS approach 12 ROLs are concentrated in one ROS. This means that 12 fragments of each event arrive at a ROS. In the case of EB, the ROS merges these arriving fragments to a bigger ROS fragment and sends it via the Gigabit-Ethernet network to the requesting SFI. This means that the event is already partially built on the level of the ROS, this is called *staged Event-building* or *concentration*. This stage in the EB helps the SFI performance as the ROS takes a part of the workload for EB. On the other hand, this solution has serious drawbacks: all the data for EB (and for Level 2 as well) needs to be moved via the PCI bus, all the transactions need to be managed by the CPU and finally the data has to leave the ROS PC

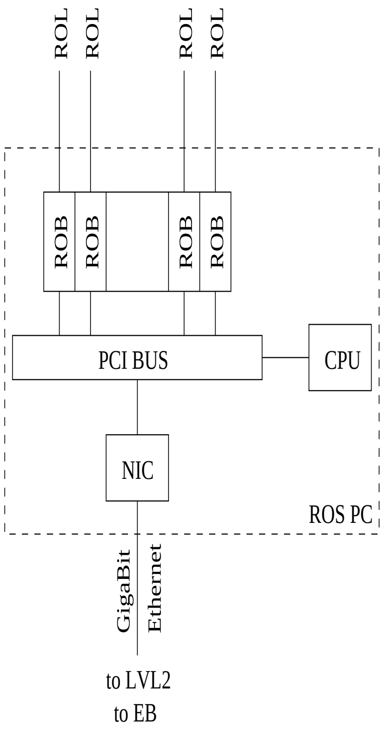


Figure 6.3: Schematic view on the bus-based ROS

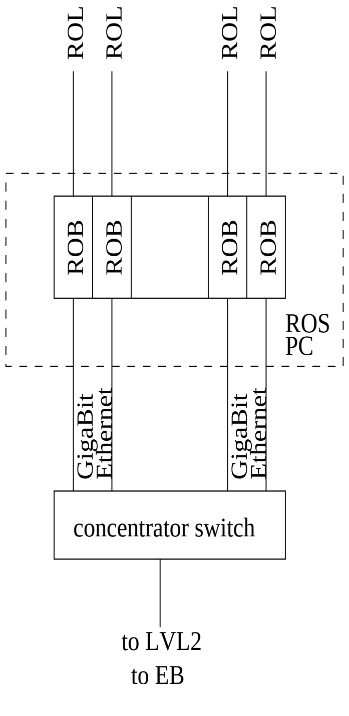


Figure 6.4: Schematic view on the switch-based ROS

via two NICs (one for RoI collection, one for EB) to the Gigabit Ethernet network. If one wanted to increase the EB rate beyond $\sim 3$ kHz, one of these three elements would become a bottleneck. As every single event has to pass every ROS, such a bottleneck would be very serious and prevent any further upgrade of the ATLAS data taking, triggering and physics capabilities. This is the major reason why the Bern group is proposing a switch-based ROS architecture, where every ROL is connected to a Gigabit-Ethernet link via the ROB, thus avoiding the possible bottlenecks in the ROS PC. This solution means that there is no staging in the EB and the SFI has to collect 12 times more fragments from 12 times more data sources, which obviously leads to more CPU occupancy and could potentially lead to a scalability problem (reading event data from 1600 different sources!).
To provide a basis for the decision switch-based vs. bus-based, our Bern group launched a measurement program dedicated to address the questions:

- Can the SFI perform building events out of 1600 data fragments coming from 1600 data sources?
- Does staged EB really help the SFI to perform better?
- In which scenario will we have a more advantageous traffic pattern in the network? Is it easier to route many small or fewer big messages through the network?
- In which scenario can we control the traffic in the network better?
- Which scenario shows a better scalability[1] to large system scales?

A large part of the measurements presented in chapter 10 are dedicated to answer the questions which are raised above and to prove that the choice of a switch-based ROS architecture is feasible and viable. If one can demonstrate, that the SFI performance and the EB system performance in a switch-based ROS scenario is the same as in the bus-based scenario or at least comparable, the switch-based ROS is a viable solution to the bottleneck problems mentioned above.
In the case of the L2PU the question of performance is much less relevant, as the L2PU will in average ask for two ROB fragments per ROS only and not for full ROS fragments. Therefore a L2PU doing RoI collection shows practically the same performance for both scenarios.
For both the L2PU and the SFI the mechanics for obtaining event data are

[1]As the ATLAS TDAQ system is large, consisting of many hundred PCs, scalability in various areas is one of the major concerns. One of these areas is the scalability of RoI collection and EB.

the same: each application requests event data from ROS PCs or from the ROBs (depending on the ROS scenario), sending a request message and receiving event fragments. So in any case the traffic pattern of RoI collection and EB is request-reply.

### 6.3.2 Level 2 High-Level Trigger

The interface between DataCollection and the Level 2 High-level Trigger (HLT) is located inside the L2PU application [DC-19]. The same software process runs the Level 2 trigger algorithms and the RoI collection. The interface between the selection software and the DataCollection is provided by the HLT infrastructure group. The interface has to deal with the following conversions:

- Level 1 result: the Level 1 result including a description of the RoI arrives via the network and is dispatched by DataCollection. This information is converted by the Byte-stream converter[2] to an Event-data model[3] object and is forwarded to the selection software.

- Data request: based on the RoI description, contained in the Level 1 result, the physics selection software requests data. The selection software is aware of the detector description and therefore knows the identifiers of those ROBs it needs data from. The DataCollection software can then resolve the corresponding network address and request the data from the ROS, which stores and makes available the event data while the Level 2 trigger is processing. The same mechanism takes place in the case of subsequent data requests of the algorithm.

- Event data: The incoming event data is dispatched by the DataCollection software layer. It is forwarded to the HLT infrastructure as a byte-stream. Before the selection software can access the information contained in the byte-stream, it has to be translated by the so-called byte-stream converter into Event-data model objects.

- Level 2 result: after the selection software has taken the Level 2 decision, a detailed record about the processing steps and the decision criteria applied is passed to the HLT infrastructure. It is converted

[2]The L2PU receives the event data from the network as a byte-stream. In order to make the event data meaningful for the trigger algorithms a conversion to objects according to the Event-data model has to take place.

[3]The Event-data model defines objects, which were identified in the detector, such as showers in the calorimeters or clusters or tracks in the inner detector.

from an object to a byte-stream. The detailed Level 2 result is then forwarded by DataCollection via the network to the pROS application. A briefer Level 2 Result, containing the decision, the Level 1 trigger type and the Level 2 trigger type only, is sent by DataCollection to the Level 2 supervisor.

### 6.3.3 Event-Filter

The interface between the SFI and the Event-filter and between the Event-filter and the SFO is the Event-filter input/output protocol (EFIO) [DC-35]. It is based on the TCP protocol (see appendix B) and designed to transfer big data blocks (1.2 MB event size) at a low rate (in the order of 40 Hz). The fully built events are sent as byte-streams from the SFIs to the Event-filter and, in case an event is accepted by the Event-filter, from the Event-filter to the SFO.

### 6.3.4 Online Software

The OnlineSW (OnlineSW) is the software used to configure, control and monitor the TDAQ system [WEB07]. The DataCollection has to interface to the OnlineSW for all services it relies on. These services are run control, configuration database, system monitoring, event monitoring and message reporting. These interfaces to OnlineSW are described more precisely in chapter 8.

# Chapter 7

# The DataCollection Network

## 7.1 Network Topology

The different DataCollection applications and the ROSs need to be interconnected with a network to exchange messages between each other. Given the performance requirements on DataCollection, the requirements on maintainability and cost constraints, Gigabit Ethernet is the only technology which is currently being investigated, after an thorough evaluation of various technologies (e.g. Fiber Channel, ATM and SCI). The Gigabit Ethernet network has to be set up in such a topology that no connection becomes a bottleneck. In case there is a bottleneck in the topology and more data tries to flow through a connection than this connection can absorb, buffers in the network switches will fill up and overflow. Therefore the switch must drop data packets, which are then lost. This problem is referred to as congestion in the switch.
In the switch-based ROS scenario, there is one kind of main data producer in the network and two kinds of main data consumer. The data producers are the ROBs (housed in ROSs, but connected to the network), which send out data fragments on request. The data consumers are the Level 2 processing unit and the SFI, both applications request the data individually. The fundamental difference between the L2PU and the SFI is, that the SFI is fully dedicated to EB and therefore should be able to drive the capability of a Gigabit-Ethernet link, whereas the L2PU spends most of its CPU resources processing trigger algorithms and the collection of the RoI data uses only a small fraction of it.
The easiest approach would be to connect all the data sources and all the data absorbers to one big central switch. However, this approach is not feasible as switches with the requested number of ports (more than 2000) do not exist.

Therefore additional switching layers need to be foreseen in the network, the so-called concentrator and deconcentrator switches: one can connect a few ROBs (e.g. 16) to a concentrator switch and connect it via an up-link to a central switch. From the other end one can connect L2PUs indirectly to the central switch via deconcentrator switches, because the required input bandwidth into a L2PU of $\sim 8$ MB/s is far below the Gigabit-Ethernet bandwidth. This approach is not valid for the SFI, because, as mentioned above, an SFI should drive the capability of one Gigabit Ethernet link. Therefore already connecting two SFIs to a deconcentrator switch would make the link between the deconcentrator switch and the central switch a bottleneck. This means that the SFIs will need to be connected directly to a central switch.
In order to reduce the size of the central switch, individual central switches for RoI collection and for EB may be installed. This means that both the Level 2 and the EB switch need to be connected to each ROB concentrator switch. This scenario (shown in Fig. 7.1) is the most probable in the beginning of the experiment [DC-59]. In later stages of the experiment, additional EB switches could be added to the system allowing the EB rate to increase above 3 kHz (see below).

Another implementation option could be to use both central switches for mixed traffic and not to separate RoI collection and EB traffic. RoI collection and EB traffic have different patterns: EB requests data from all data sources in an almost equally distributed traffic pattern, whereas RoI collection will ask for a lot of data from certain data sources, but will never or rarely touch other data sources. Therefore mixing the non-uniform RoI collection and the uniform EB traffic would allow to load the switches in a more balanced way. With the proposed "almost bottleneck-free" architecture [BEC03b], the system can grow continuously by adding more central switches with either L2PUs attached to them (again via deconcentrator switches) or SFIs. Especially adding more central switches for the EB and adding more SFIs will be an interesting option if physics needs require a bigger Event-filter capacity and therefore the EB rate needs to be increased.
Even with a very carefully designed network topology, switch buffers may fill up due to short-time traffic fluctuations (short-time congestion). Therefore it is important to control and restrict the number of messages in the network with a fine granularity to avoid buffer overflows and reduce message loss. The restriction of the total number of Ethernet frames in the network to an upper limit is implemented on the application level, e.g. in the SFI. Such strategies to control the traffic in the network are called traffic shaping and will be discussed in section 9.3.

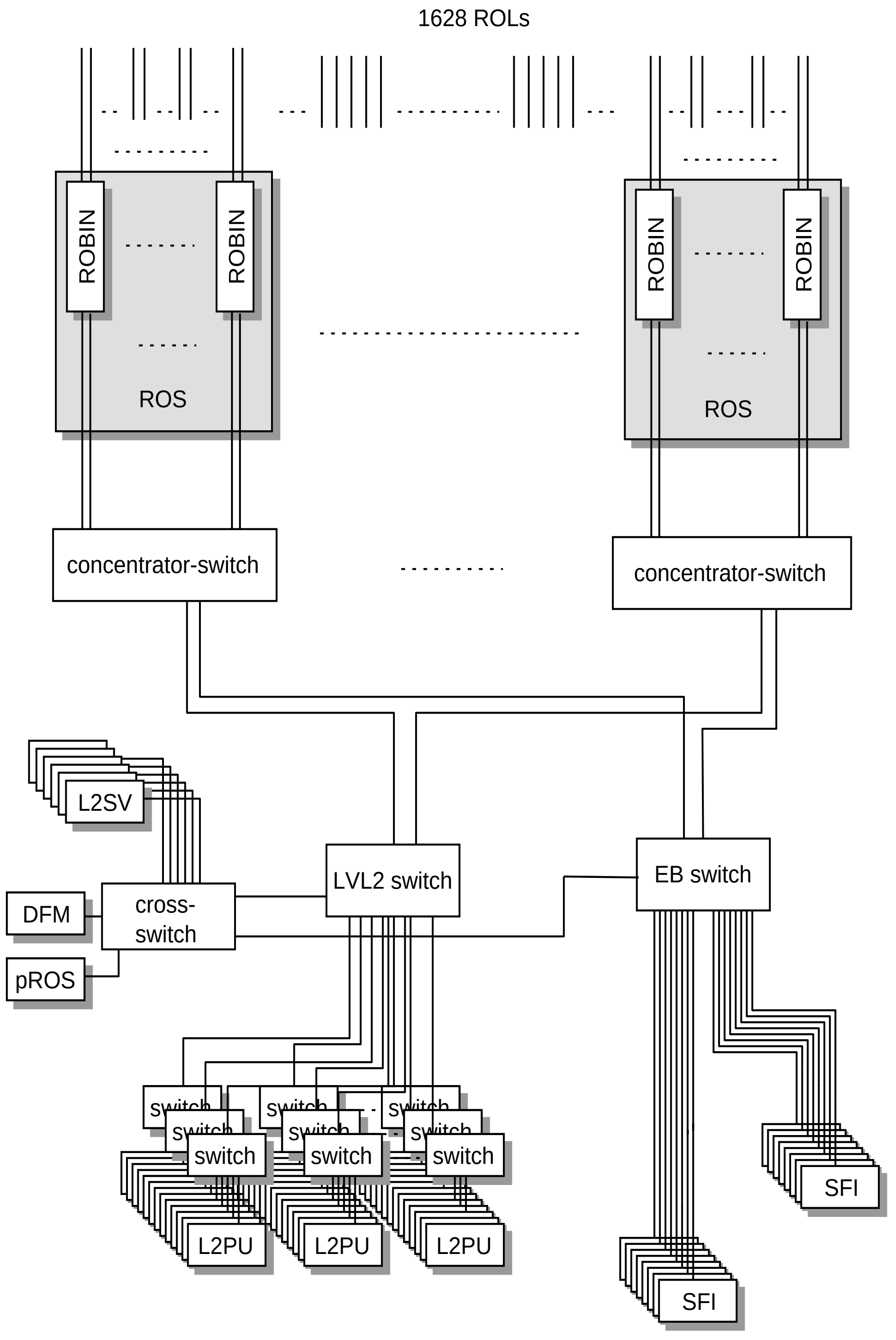


Figure 7.1: The topology of the DataCollection network at startup of ATLAS in the switch-based ROS scenario.

## 7.2 The Link Technology: Gigabit Ethernet

The link technology of the DataCollection network is exclusively based on Gigabit Ethernet [SIN97].
Gigabit Ethernet has the following key specifications, which are important for the DataCollection system:

- Line speed: 1000 MBits per second
- Maximum data fragment size (frame size): 1518 Bytes
- Copper or fiber cables as physical connections
- Full duplex: sending and receiving of data do not interfere
- Flow control: an overloaded receiver can send an "XOFF" message to the sender to stop the transmission of more data. The receiver resumes with an "XON" message.

A parameter which plays a key role in all DataCollection performance studies (see chapter 10) is the maximum data fragment size or frame size of Ethernet. As specified above, its maximum size is 1518 Bytes, but due to necessary header information in the front of every message, the effective payload per frame available for DataCollection is $\sim$ 1460 bytes. This means that independent of the message size the data sources send out, the receivers (L2PU and SFI) will always get the data in pieces of 1460 Bytes and need to merge these fragments to a full message. This fact is of particular interest for EB, because the size of the data fragments sent via the ROLs to the ROBs are for most of the ATLAS sub-detectors of the size of $\sim$ 1000 Bytes (see chapter 2 of [ATL03]).

# Chapter 8

# The DataCollection Software Framework

The framework provides an ensemble of common functionalities to the DataCollection applications. These services include Message-passing, Application Control, Error Reporting, Configuration Database, System Monitoring, OS Abstraction Layer, Event-formatting and Time-stamping. A part of these framework packages (e.g. Application Control) needs to interface to the common TDAQ control software services, provided by the OnlineSW ensemble. A global view on the DataCollection software framework is given in [DC-01], [DC-43] and [HAE03].
As the DataCollection framework is designed and implemented in an object oriented language (C++), specific terms like *class*, *object* or *inheritance* will occur in this chapter. These terms are explained in appendix D.

## 8.1 Message-Passing

The Message-passing layer is responsible for the transfer of event data and control messages in between DataCollection components and between DataCollection components and the ROS [DC-08], [DC-13]. The control messages ensure the proper movement of the data. The DataCollection Message-passing imposes no structure on the data to be exchanged, except a four-byte alignment of the byte-stream. It allows the transfer of data blocks of up to 64 kB size with a best-efforts guarantee. For efficiency reasons this layer does no re-transmission or acknowledgment of data. This choice has allowed the API[1] to be implemented over a range of technologies (see appendix B) without im-

[1]Application programming interface: the interface between different modules of software

posing an unnecessary overhead or the duplication of existing functionality, e.g. in the case of TCP/IP. The API supports the sending of both unicast[2] and multicast[3] messages. The latter may be emulated to the DataCollection application code in case the underlying protocol does not support multicast, e.g. for TCP/IP.

The architecture of the Message-passing layer is shown in Fig. 8.1. The Port class is the central interface for sending data. Every message, which is sent or received, is handled by a Port object. All user data has to be part of a Buffer object to enable it to be sent to or received from a Port. The Buffer interface allows for addition of user-defined memory locations without involving copying. The Provider class is an internal interface from which different protocol and technology-specific implementations inherit. It creates and holds the Port objects. Multiple Provider objects can be active at any given time allowing the concurrent use of different protocols. To date providers implementing TCP/IP, UDP/IP and raw Ethernet exist. The design is open for more protocols to be added at any future time.

As the Message-passing layer itself does not guarantee the reliable delivery of messages via the network (except if running a reliable underlying network protocol e.g. TCP/IP), the DataCollection application code has to implement strategies to avoid message loss and to recover from eventual packet loss. We have mainly observed message loss in input buffers of switches and in the Linux kernel input buffer. A large part of this loss can be avoided by limiting the number of outstanding requests in a request-reply traffic pattern. This traffic shaping will be extensively discussed in section 9.3.
More information about the DataCollection Message-passing can be found in [DC-08], [DC-13] and [DC-21].

## 8.2 Application Control

The ATLAS TDAQ system consists of applications running distributed among many hundreds of computers. Therefore a hardware[4] and software infrastructure is needed to control those applications, which is called run control. The run control interface is responsible for translating state transition commands (e.g. start or stop), issued by a run controller, into commands internal to a DataCollection application. This run control interface is called application control and is inherent in every DataCollection application. It controls activities which are implemented by the application developer and which

[2]Sending a message to one single network destination
[3]Sending a message to multiple network destinations
[4]A control network beside e.g. the DataCollection network

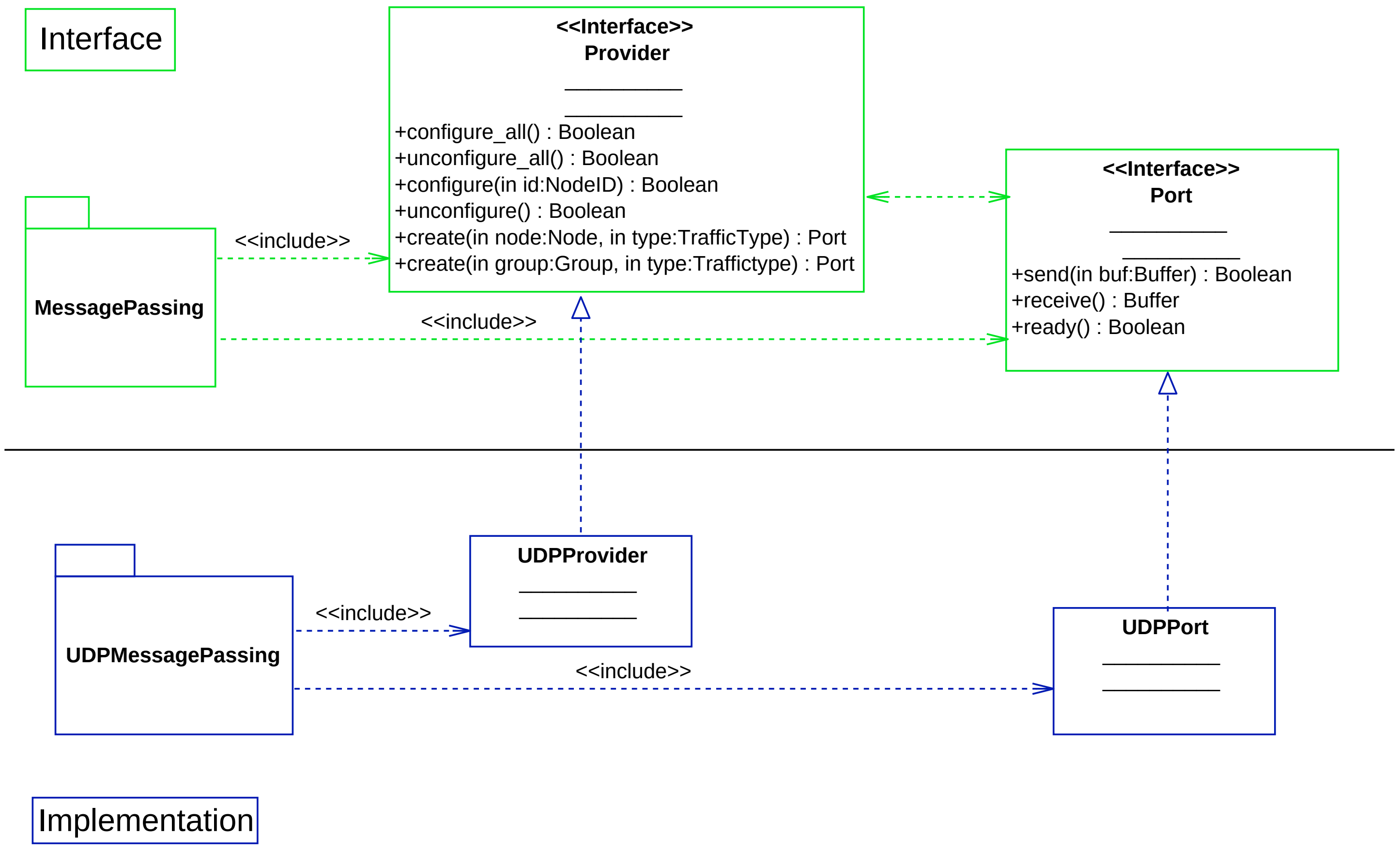

Interface
<<Interface>>
Provider
+configure_all() : Boolean
+unconfigure_all() : Boolean
+configure(in id:NodeID) : Boolean
+unconfigure() : Boolean
+create(in node:Node, in type:TrafficType) : Port
+create(in group:Group, in type:Traffictype) : Port
<<Interface>>
Port
+send(in buf:Buffer) : Boolean
+receive() : Buffer
+ready() : Boolean
<<include>>
MessagePassing
<<include>>
UDPMessagePassing
<<include>>
UDPProvider
<<include>>
UDPPort
Implementation

Figure 8.1: The Message-passing design. UDP is taken as an example, but it can be mapped to TCP, raw Ethernet or other network protocols as well.

follow the state transition commands. The application's functionality is implemented in the activities, which are therefore the main building bricks of the DataCollection applications. Examples of activities will be presented in chapter 9. The requirements on the DataCollection application control are documented in [DC-06].

## 8.3 Error Reporting

The Error Reporting package allows the logging of error and debug messages either to standard output, standard error or to the message reporting system (MRS) provided by the OnlineSW. Each DataCollection software package can define its own set of error messages and error codes. Error logging can be enabled/disabled on a per package basis. Furthermore, debug messages and error messages are treated logically differently, so the debug message could go to standard output while all normal application logs go to MRS. The user only interfaces via a set of macros to the Error Reporting system. This allows the optimization of the applications at compile time by compiling debug messages out of the DataCollection code in case they are not needed. More details about the Error Reporting, implemented by the author, are available in [DC-04], [DC-15] and [DC-20].

## 8.4 Configuration Database

All DataCollection applications obtain their configuration parameters from the configuration database provided by the Online SW. The access layer to the configuration database allows the underlying implementation to change without implying changes to the application. The application's view of the database is hidden by configuration objects which access the database, providing a more convenient way to access configuration information. The configuration objects themselves are created by a code generator, which parses[5] the configuration database schema file. See [DC-03], [DC-11] and [DC-52] for more details.

## 8.5 System Monitoring

This part of the framework allows every DataCollection application to make arbitrary information available to some outside client. In practice this infrastructure is used to publish operational statistics like counters or rates.

[5]to parse: a program interprets a text file.

The package makes this information available in various different ways, including the OnlineSW. The system monitoring services are implemented and attached to the DataCollection applications as activities. Detailed documentation about the system monitoring is available in [DC-05] and [DC-33].

## 8.6 OS Abstraction Layer

The Operating system (OS) abstraction layer consists of packages hiding all OS specific interfaces. E.g. the threads package hides the details of the underlying POSIX [GAL95] thread interface, which could be replaced by another implementation without propagating the changes to the application code. The requirements on the OS Abstraction layer are spelled out in [DC-02].

## 8.7 Event-Format

This package supports formatting event data according to the structure specified in [BEE98]. The ATLAS full event data is structured in Subdetector-fragments, which are then divided into ROS-fragments, which are again divided into further subfragments. Every fragment contains a header and a number of subfragments. The Event-format library links the event fragments to their header information and their subfragments. In addition the Event-format library needs to provide the tools to navigate through the event structure.
As the data fragments of a given event are not necessarily residing in a continuous memory space, but distributed among several buffers in memory, the Event-format library must be aware of how the event fragments are stored. This awareness is implemented in so-called Storage types. One important example is the I/O vector storage type. This storage type avoids copying when sending out an event which is distributed to many places in memory: it allows for scatter-gather send operations. A vector of pairs of pointers and sizes of data fragments (a so called I/O vector) are handed over to the operating system in one single system call. This strategy avoids copying data in the memory without increasing the number of system calls. It will be discussed in section 9.3.

## 8.8 Event-Storage

The Event-storage library is responsible for writing events to disk. Usually it is used by the SFO, but it can also be used by the SFI when a setup without

Event-filter is operated. The package allows the data of a run to be split into different files of a defined size. The requirements of the Event-storage library are described in [DC-37].

## 8.9 Time-Stamping

The Time-stamping library provides hooks to resolve the timing behavior of the software. Two dedicated implementations exist. The first one is based on NetLogger [TIE02] and used to understand the timing behavior of a distributed system [DC-32]; the second provides fine-grained time resolution to understand the timing behavior inside a single application [DC-48]. The output of both implementations satisfies the NetLogger format and therefore the NetLogger visualization tool can be used to analyze the data generated by both of them. An example of results achieved with the Time-stamping library is shown in chapter 10.

## 8.10 Multi-threading

Modern operating systems (e.g. Linux or Windows 2000) support multi-tasking. Multi-tasking means that a user can run multiple processes on one computer in parallel. The so-called scheduler of the operating system assigns time slices on the CPU(s) to the different tasks.
Multi-threading is a lightweight version of Multi-tasking. In Multi-tasking the different tasks are well shielded from each other and they have separate memory areas assigned. This is not the case for multiple threads, which share the same memory area. This concept has the advantage that switching and passing of information between different threads is easier and faster. It allows a complex program structure to be broken up into simpler logical blocks. The disadvantage of this concept is that, due to the usage of a common memory area, the threads can interfere, which may lead to slowdown, memory leakage or even to crashes of programs.
Multi-threading is a central design paradigm of the DataCollection software. It allowed the design of applications to be kept simple by breaking up the functionality into well-contained functional blocks. The performance of the DataCollection applications also benefits from this paradigm, especially when running on dual-CPU machines, because both processors can be fully loaded running threads in parallel.

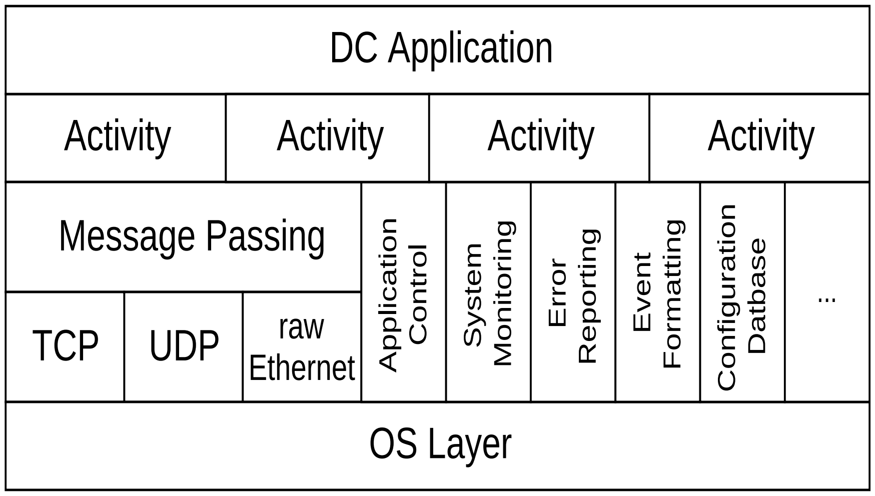


Figure 8.2: An overview of the DataCollection framework architecture

## 8.11 Composition of DataCollection applications

A DataCollection application (e.g. L2SV or SFI) is a program which is built upon the DataCollection framework. The main building blocks of an application are the activities. An activity is a module inside the application which obeys the state transition commands of the external run control. Internally, an activity runs one or multiple threads, which execute a part of the application's functionality. For the implementation of the functionality and for more generic services (e.g. Error Reporting) the framework functionality is accessible for the application developer.
To summarize: a DataCollection application is a collection of activities. Activities benefit from all framework services and implement the application's functionality. This architecture is drawn in Fig. 8.2 and will be illustrated with examples in chapter 9.

# Chapter 9

# The Design of the Event Building Applications

The EB applications are the DFM, the SFI and partly the SFO. These applications are DataCollection applications as generically described in section 8.11. The sequence of interactions between the applications (except the SFO) is drawn in Fig. 6.2. The author made major contributions to the design and implementation of all three EB applications.

## 9.1 Design of the Dataflow Manager

The DFM receives Level 2 decisions from the L2SVs. It assigns every Level 2 accepted event to an SFI for EB and receives an EoE message when the event is built. It clears Level 2 rejected and built events from the ROSs. Its design is drawn in Fig. 9.1 and documented in [DC-23].

There is almost no parallelism in the task of the DFM: either it receives a Level 2 decision from the L2SV or an EoE message from the SFI. In both cases the program has to access the EB bookkeeping facility, the load-balancer. Therefore there is no reason to split the main task into different threads, because the threads would interfere, locking each other on the bookkeeping facility. However, there are independent service threads, e.g. for timeout or system monitoring, which access the load-balancer not at all or only at a low rate ($\sim$ 10 Hz).

The bulk of the work is done in the Input activity. This activity takes all the action needed for treating the incoming messages: in case of an incoming Level 2 decision it assigns an event to an SFI using the Load Balancer algorithm. In case of an 'end-of-event' message the event is removed from the load-balancer. In addition, Level 2 rejected and built events are passed

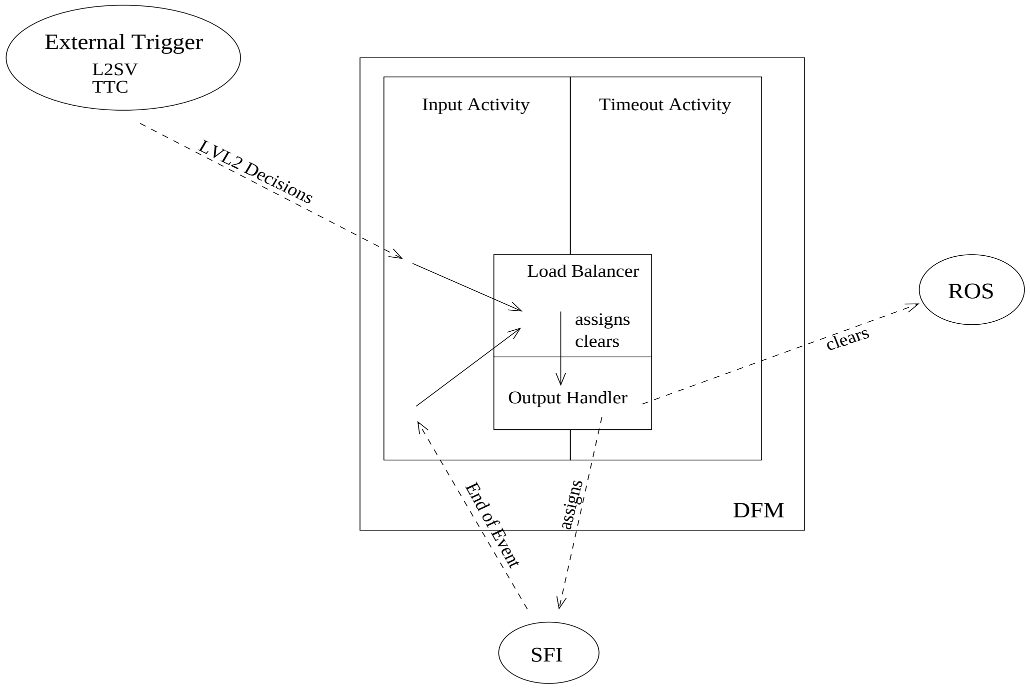


Figure 9.1: Schematic view of the DFM design

to the output handler to be deleted from the ROS. Under certain conditions the Input activity may receive busy/non-busy messages from individual SFIs: these SFIs are then taken out of/put into the load-balancing.
The Timeout activity in the DFM scans at a low rate through all events held by the load-balancer. It identifies events which were assigned to an SFI, but for which no EoE message was received. This may lead to a reassignment of this particular event to an SFI for EB. This events are marked as possibly duplicated in their header to catch the case the event was built but the EoE message was delayed or lost.

## 9.2 Design of the Subfarm-input

The SFI application provides the main part of the EB functionality. Therefore it is the most complex EB application, running different independent tasks in parallel:

- Dispatching incoming messages from the network
- Requesting event fragments from the ROSs
- Identifying event fragments and building events

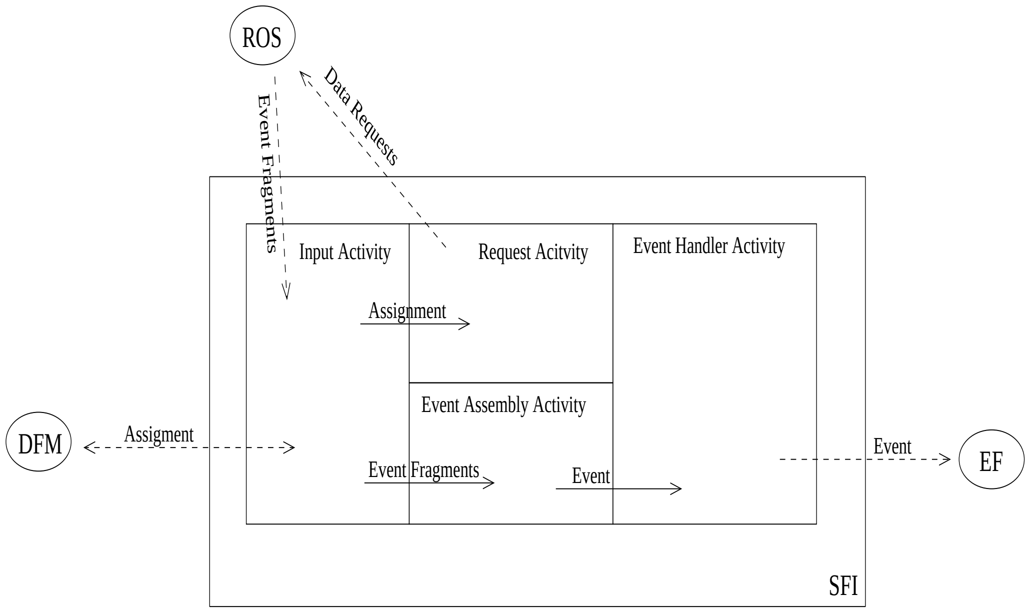


Figure 9.2: Schematic view of the SFI design

- Re-asking for missing event fragments
- Sending the events to the Event-filter
- Providing events for monitoring

Each of the subsets of the SFI functionality are implemented in a specific activity. The activities directly involved in EB and their interactions are drawn in Fig. 9.2. The SFI design document is [DC-44].

### 9.2.1 Input Activity

Dispatching incoming messages from the network is done by the Input activity. It puts the received messages into a DataCollection buffer (see section 8.1). In case the incoming message is an event assignment from the DFM, the event identifier is put into a queue to the Request activity. In case it is an event fragment from the ROS, the buffer is kept and a pointer to the buffer is put in a queue to the Event-assembly activity.

### 9.2.2 Request Activity

The Request activity sends a data request for each event to every ROS. In order not to overload the network and the Input activity, the number of outstanding requests at any given time is limited. This measure is called traffic-shaping and discussed in section 9.3.

### 9.2.3 Event-Assembly Activity

The Event-assembly activity identifies the incoming event fragments: it reads the event identifier and the data source from the message header. This allows the fragment to be put into a fragment directory. As soon as all fragments of a given event have arrived, they can be put in the event structure using the Event-format library, described in section 8.7. The events are not copied to a continuous memory space, but the pointers to the DataCollection Buffers, where the event fragments reside, are held in the order specified by the event format.

### 9.2.4 Event-Handler Activity

The Event-handler activity controls the EFIO server in the SFI. For every connection to an Event-filter a EFIO Handler is created. The EFIO Handler retrieves pointers to events from a queue and sends them out sequentially on request of the Event-filter.

### 9.2.5 Event-Sampler Activity

The Event-sampler activity is the interface of the SFI to the event monitoring facility of the OnlineSW. It keeps events ready to be sent out on the request of an external monitoring client.

### 9.2.6 Service Activities

In addition, there are a couple of service activities implemented:

- The Timeout activity identifies missing fragments of events and reports them to the Request activity for a re-ask.
- Two other activities are needed for the system monitoring. These two activities are not SFI specific and therefore a part of the DataCollection framework (see chapter 8).

## 9.3 Evolution of the Subfarm-Input Design

The SFI design, which is described above, is the result of a long evolution process. The evolution was heavily driven by performance measurements and optimizations in a very short feed-back loop: the author worked on a powerful testbed, running the SFI against FPGA and ALTEON ROS emulators (appendix A). There were several strategies to be chosen and problems to be identified and solved in order to provide performant and efficient SFI code. A chronological summary of the evolution of the SFI design is presented in [GAD03].

### 9.3.1 General Strategy

There are two relevant rates in the dynamics of the SFI application: the event rate and the fragment rate. The target value for the event rate on an SFI is in the order of 40 Hz. As an event consists of 1600 fragments from as many ROBs, the fragment rate is 64 kHz. For every event fragment which arrives at the SFI a request was sent out before. Therefore the total message rate on the SFI is 128 kHz. The obvious strategy to meet these requirements is to do as many operations as possible (even heavy operations) with the event rate, e.g. preparing the structure for the full event or composing the queue of all fragment requests for a given event. The operations which occur at the aforementioned high fragment rate (like unpacking a ROS-Fragment) should be very lightweight and fully optimized.

### 9.3.2 Multi-threading

Currently, high end PCs are typically SMP dual processor devices. The two processors are driven by the same operating system and share the same memory space. As multiple threads within the same process can run on different CPUs, multi-threading allows both CPUs of the PC to be loaded and to exploit the available computing power fully.

### 9.3.3 Advanced STL Memory Allocators

The DataCollection applications are heavily based on the STL [JOS99], which provides data containers such as vectors or maps. STL has its own memory management dealing with its own memory pool. There is a global pool for a process, which is shared between the threads. To avoid memory corruption, the pool is locked for all the other threads whenever a thread is accessing it. If another thread needs to access the memory pool at the same time, it is blocked until the first thread has unlocked the memory pool. If such jams of threads occur frequently, this results in a big performance loss of the application. To avoid this, the so-called *pthread allocator* can be used, which is using an independent memory pool for each thread. Applying the pthread allocator boosted the performance of the SFI by more than a factor of two. However, the pthread allocator may have a serious drawback: as there is no flush mechanism between the multiple memory pools, a memory leak may occur if memory requested in one thread is systematically released in a different thread. Therefore DataCollection has implemented a more advanced allocator, which uses a memory pool per thread as well, but with a flush mechanism between the different memory pools via a global pool. As the global pool may be accessed by different threads, it needs to be locked as well. Due to the memory buffering in the thread-owned memory pools, accessing the global pool happens at a low rate only and therefore the probability that one thread blocks another thread is much smaller. No performance loss due to thread jams is observed.

### 9.3.4 Advanced Buffer Pools

It is the nature of the SFI that it has to buffer event data until each event has completely arrived and can be built. The buffer space for the event fragments is the memory of the SFI PC. A process can request this memory dynamically from the operating system and must return it when it does not need it anymore. As this procedure is quite CPU time consuming, it was decided to allocate the needed buffer memory once and to re-use it for the whole lifetime of the process.
These preallocated buffers need to be managed by a buffer pool, which knows which buffers are available and which buffers are occupied, to avoid memory corruption. As only one thread at a time can be allowed to request a buffer from or return it to the pool, the pool must be locked while the operation is happening. This may lead to threads blocking each other, which results in performance loss, as already discussed for the STL allocators. The chosen solution is similar to the one for the STL allocator problem: a buffer sub-pool per thread with a flush mechanism to the global pool was implemented: a thread which is using all the buffers in its sub-pool can request new buffers from the global pool, a thread which releases more buffers than it requests returns empty buffers from its sub-pool to the global pool. In a stably running system the balance is neutral. Accessing the global pool happens at a low rate only and therefore the probability that one thread blocks another is low.

### 9.3.5 Traffic-Shaping

Sending out an event request to a ROS takes $\sim 6\mu$ s, while dispatching an incoming message, identifying it and putting it into the event structure takes $\sim 20\mu$ s. In case the Request activity runs uncorrelated to the other activities

of the SFI it will overload the application. As more data will flow into the PC than the application can absorb, the operating system will drop event fragments from its kernel buffers. Many different approaches have been tried to throttle the rate of the Request activity. The final solution to this problem was to limit the number of outstanding requests, blocking the Request activity when the threshold is reached via a Semaphore[1] mechanism between the Input activity and the Request activity. This mechanism is called credit based traffic-shaping and it is one of the author's main breakthroughs while working on the EB system.
This mechanism not only protects the SFI against overloading itself, but it also helps stabilize the whole EB system to a large extent. Limiting the number of outstanding requests on the level of a single SFI means to limit the number of messages in the whole EB system with a very fine granularity on a per ROS fragment level. If this limit is set at a safe value no buffers in switches will overflow and messages will only be lost very rarely. In addition, thanks to the credit based traffic-shaping the system becomes self-regulated: whenever problems in the network occur and switch buffers start filling up, the latency of a request-reply transaction will increase. Therefore the request rate on the SFI will automatically slow down and the network can recover. As soon as the network has recovered, the latency in the request-reply traffic will decrease and the request rate will go back to the maximum.

### 9.3.6 Scatter-Gather Send

The SFI receives $\sim 1600$ event fragments and has to send out the full event in a continuous TCP byte stream. The most naive and simple approach would be to put all the 1600 fragments into a continuous memory space and to ship it out. The disadvantage of this simple solution is that the SFI needs to copy internally a lot of data until all the fragments are in the continuous memory space. Therefore the paradigm chosen to fulfill this task is the so-called Scatter-gather Send. This means that, after an event fragment has arrived at the SFI and has been put into a DataCollection buffer, it is not moved in memory until it is sent out. A key tool to achieve the scatter-gather send is the IOVector storage type of the event-format library, which was already briefly discussed above (see section 8.7). The IOVector storage type holds pointers to the event fragments and the fragment sizes. On request it produces an I/O vector (a vector of pointers to event fragments and their sizes) in the order specified by the Event-format library. This vector can

[1]A synchronization mechanism between two or more threads (see in the glossary).

then be passed to the operating system via two system calls[2]. The operating system then sends the elements of the vector in the correct order to the network. On the other end the receiver gets the event as a continuous byte stream.

### 9.3.7 Inter-Thread Communication

As explained in section 8.10, the communication between multiple threads is lighter and faster than that for multiple processes. Despite this fact, inter-thread communication can slow down the performance of an application significantly. Two such problems were identified and solved in the SFI:

- In order to implement the credit based traffic-shaping, a counting mechanism between the Request activity (sending out fragment request) and the Input activity (receiving the requested fragments) is needed. This mechanism is based on a so-called counting Semaphore. For every request issued the counting semaphore is decreased, for every received fragment it is increased. If there are no credits left, the Request activity blocks on the semaphore until it is increased by the Input activity. This switching of the semaphore takes $\sim 6\mu$s. This problem was solved by grouping the Semaphore switches: a switch signifies not one credit but $n$ credits. This means whenever the Request activity unblocks on the semaphore it is allowed to send out $n$ data requests. Accordingly the Input-thread does not increment the Semaphore before it has received $n$ event fragments.

- The Input activity dispatches messages and the Event-assembly activity does the logical EB. Between the two activities a queue to hand over pointers to DataCollection buffers (where the data fragments reside) is used. As dispatching the message takes more time than doing the logical EB, this queue is emptied frequently and the Event-assembly activity blocks on the queue. Whenever a fragment is put into the queue by the Input activity, the Event-assembly activity wakes up and does the necessary processing. Blocking and waking up the underlying thread of the activity takes a significant amount of CPU resources. This amount is so large, that applying optimizations to the Event-assembly activity, and therefore provoking more block-wakeup cycles, slowed down the application in total. Therefore the queue grouping

[2]As 1024 IOVector elements can be passed to the operating system within one system call in maximum, two system calls are needed to send out an event consisting of 1600 fragments.

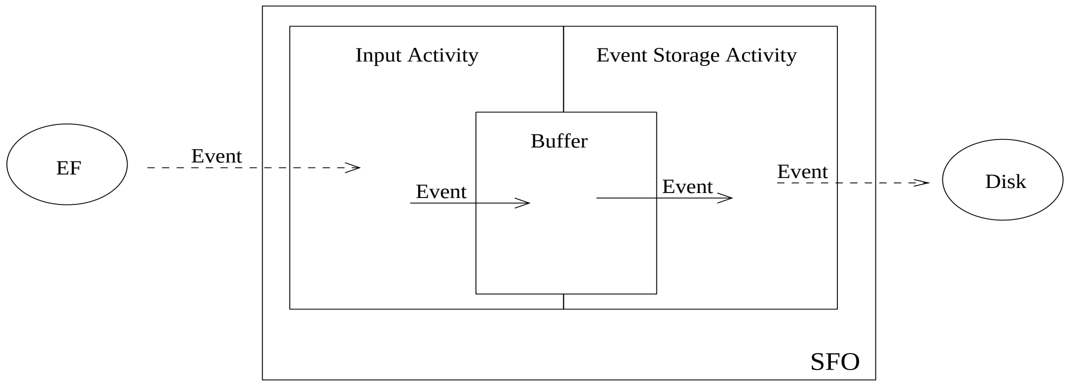


Figure 9.3: Schematic view of the SFO design

was introduced: the Event-assembly activity does not wake up before $n$ entries are waiting in the queue.

## 9.4 Design of the Subfarm-output

The SFO is a simple application receiving events from the Event-filter and writing them to disk. This task can be split into two disjunct jobs:

- Reading the events sent by Event-filters from the network and putting them into an internal buffer (memory of the PC)
- Taking the events from the internal buffer and writing them to disk

The design of the SFO is very simple and drawn in Fig. 9.3. The first part of the functionality is provided by EFIO-Handlers in the Input Activity. An EFIO-Handler is a thread serving one TCP connection to an Event-filter node, therefore there is one handler for every Event-filter node the SFO receives data from. The EFIO Handler serves the events sequentially, reads them from the network connection to the Event-filter and stores them in the internal buffer of the SFO.
The second part of the SFO functionality is provided by the Event-storage activity. It writes events sequentially, which are stored in the internal buffer, to the local hard disk using the Event-storage library.

# Chapter 10

# A Study on the Event Building Performance

Two kinds of tests were provided to validate the performance of the EB system:

- Application tests to evaluate the performance and the scalability of an isolated component
- System tests to evaluate the performance and the scalability of multiple components

Both application and system tests were needed to validate and develop views on the ROS architecture.
EB performance measurements were reported in a number of notes and publications: [DC-60], [LEV03], [HAE03], [GAD03], [BEC03a] and [LEH03]. The author provided all the performance measurements presented except for the DFM application tests.
The tests were exclusively performed on the so-called *wedding-list* testbed, which was built for Dataflow performance studies [WEB03]. The testbed consisted of 36 dual-CPU PCs. They were equipped with Intel XEON processors with a clock frequency of 2.0, 2.2 or 2.4 GHz, with Intel e1000 Gigabit Ethernet NICs and with 1 GB RAM. The operating system was Linux with the kernel version 2.4.9 for SMP machines. The PCs were interconnected with Gigabit Ethernet via two BATM T6 switches(see appendix C). For specific measurements two PCs were interconnected back-to-back[1]. For run control and system monitoring they were interconnected with a secondary network in order not to interfere with the DataCollection message flow on

[1]Back-to-back means that the Gigabit Ethernet NICs of two PCs are directly connected with an Ethernet cable, without having a switch in between.

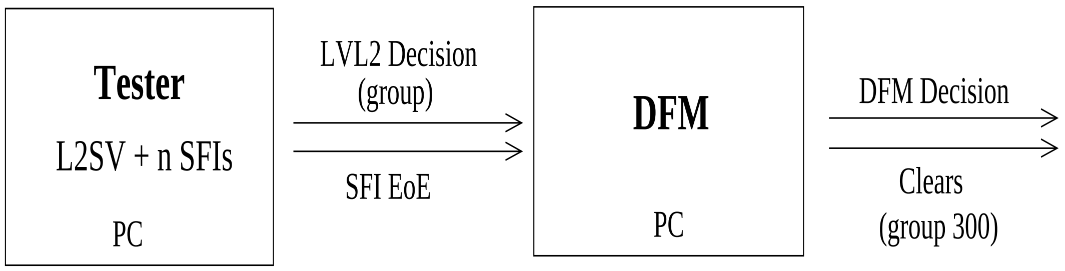


Figure 10.1: DFM test setup

the primary network. As test equipment a dedicated tester program for the DFM and hardware ROS emulators were employed (see Appendix A). The emulators were interconnected with the PCs either directly via a BATM T6 central switch or via BATM T5 concentrator switches and a BATM T6 central switch. The terms *central switch* and *concentrator switch* are explained in section 7.1.

## 10.1 Event-building Application Tests

### 10.1.1 Dataflow Manager Performance

The main aim of the DFM application tests was to determine the maximum EB rate the DFM can handle and to investigate the scalability of the application. Scalability is important, because the DFM will need to handle a big system with $\sim$ 75 SFIs and hundreds of events being built concurrently. The DFM is not expected to be the bottleneck in the system, but measurements were needed to confirm this expectation and to determine the maximum rate the DFM can sustain.

#### DFM Test Setup

The DFM application tests were mainly performed on a two node-system, connected back-to-back. On one node the DFM application ran and on the other node a specific DFM tester application was deployed. The CPU clock speed of the tester PC and of the DFM PC was 2.2 GHz. The schema of the test setup is drawn in Fig. 10.1

The tester application emulated the behavior of the L2SV and of a configurable number of SFIs. It sent Level 2 decision messages and EoE messages to the DFM. The Level 2 decision messages were grouped (the strategy of grouping Level 2 decision is explained in section 6.2) and there was always

one Level 2 accept per message. The delay (in number of events) between a Level 2 accept and the corresponding EoE message determines the number of concurrently built events in the EB system. The DFM sent all its messages to the network without having real SFIs connected as destinations. This means that the messages sent by the DFM could not be delivered to any destination and were dropped in the network. Therefore a connectionless protocol[2] was used for the message-passing.
The measurements were done in such a way that the tester program slowly ramped up its speed until the DFM could not handle the message rate any more and started losing Level 2 decision messages and EoE messages. This immediately triggered a bunch of WARNINGS (timeouts in case of the EoE messages or EoEs of unknown events in case of the Level 2 decision message). The DFM was modified to stop at the first such warning and the speed which was reported last could be identified as the maximum rate the DFM can sustain. This test procedure ensures that the DFM is the bottleneck in the test system and therefore its performance can be assessed. If the tester application or the back-to-back connection were the bottleneck, the DFM would not issue any WARNINGS, as it could handle all incoming messages.
In order to validate this test procedure, comparative cross-check measurements with small EB systems (e.g. one tester, one DFM, one software ROS emulator and up to eight SFIs interconnected with the T6 switch) were successfully provided.

### DFM Test Results and Discussion

The measurements done for the DFM were mainly dedicated to assessing the scalability of the application. The number of concurrently built events and the number of SFIs in the EB system were varied and their effect on the SFI performance was studied. An additional varied setting, were the grouping factors of the Level 2 decision message and the clear message. The idea of

[2]DataCollection uses two connectionless protocols: raw Ethernet and UDP (User Datagram Protocol: see appendix B). Sending messages to non-existing destinations has some technical particularities: for raw Ethernet frames it means that the messages were sent to a non existing Ethernet (MAC) address. For UDP the situation is slightly more complicated: a dummy UDP receiver was needed, listening and receiving messages on a given port. This was necessary in order to prevent arp messages being exchanged through the network between the NICs, which otherwise would corrupt the result. As UDP is IP-based and we ran an IP over Ethernet system, the IP address resolution protocol (arp) was involved to resolve the destination's address. If this fails, arp feeds the failure back to the sender. UDP will then suppress the next outgoing message but send the following one. This means that without a dummy receiver only around 50% of the messages would be sent out by the DFM and the DFM performance would be overestimated.

grouped messages is explained in section 6.2. The measurements done for the DFM were:

- EB rate vs number of SFIs. Every SFI builds 2 events concurrently. The measurement was done using the UDP and the raw Ethernet protocols and with a grouping factor of the Level 2 decision message of 100 and 300. The grouping factor of the clear message was 300.
- EB rate vs number of SFIs. The total number of concurrently built events was constant. The grouping factors were 100 for the Level 2 decision and 300 for the clear message. The UDP network protocol was used.
- EB rate vs number of concurrently built events. The number of SFIs was constant. The grouping factors were 100 for the Level 2 decision and 300 for the clear message. The UDP network protocol was used.
- EB rate vs CPU speed of the PC running the DFM. The UDP network protocol was used.
- In addition the DFM was time-stamped to investigate how much time the application spent on given actions.

**EB rate vs number of SFIs, two events per SFI concurrently built** The results presented in Fig. 10.2 show that the scaling behavior of the DFM is almost flat: around 10% performance drop was observed, increasing the number of SFIs from 1 to 200 (thus increasing the number of concurrently built events from 2 to 400). For a grouping factor of 300 the message rate between the L2SV and the DFM is a factor of three lower, which explains the better performance of this scenario. The raw Ethernet protocol performs better than the UDP protocol. This result is consistent with the Linux network performance study presented in [DC-54].

**EB rate vs number of SFIs, number of concurrently built events constant** Keeping the number of concurrently built events (400) constant, but varying the number of emulated SFIs, the DFM performance drops by 3% when going from one to 200 SFIs (Fig. 10.3).

**EB rate vs number of concurrently built events, number of SFIs constant** The number of emulated SFIs in the system was kept constant at a value of 50. The number of events in the EB system was varied from 3 to 800. Fig. 10.4 shows the result: a performance drop of 4% was observed.

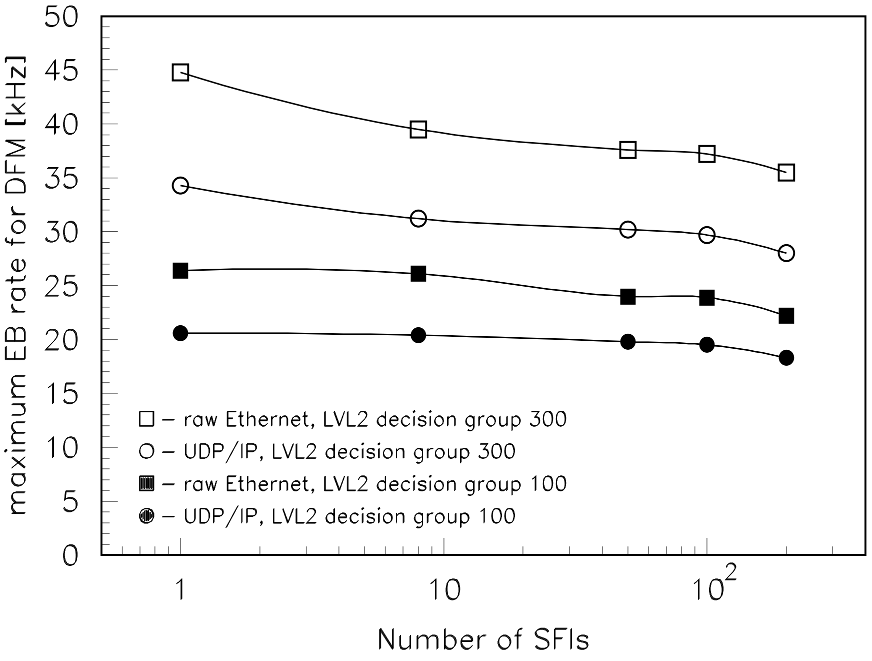


Figure 10.2: DFM performance versus number of SFIs for two events concurrently built per SFI

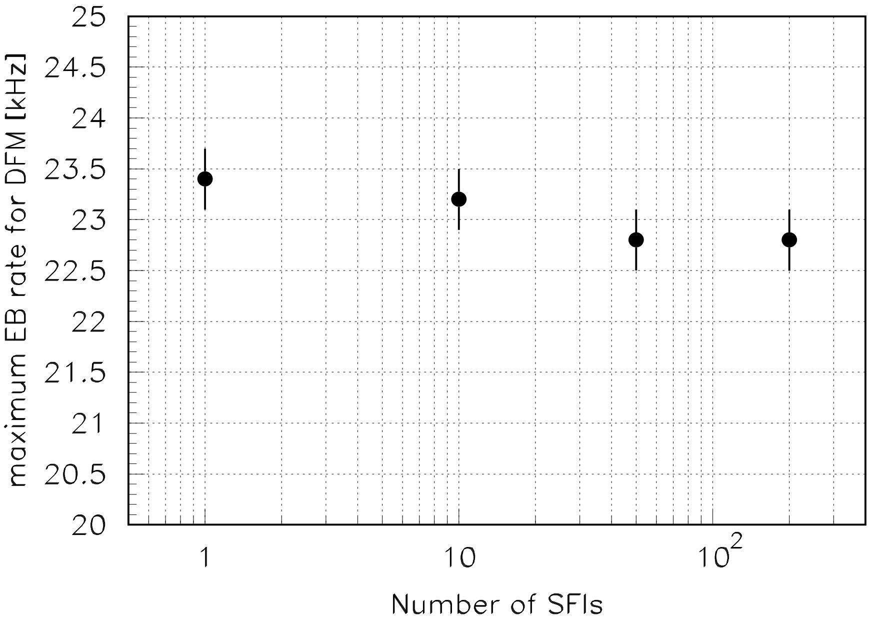


Figure 10.3: DFM performance versus number of SFIs with constant total number of concurrently built events

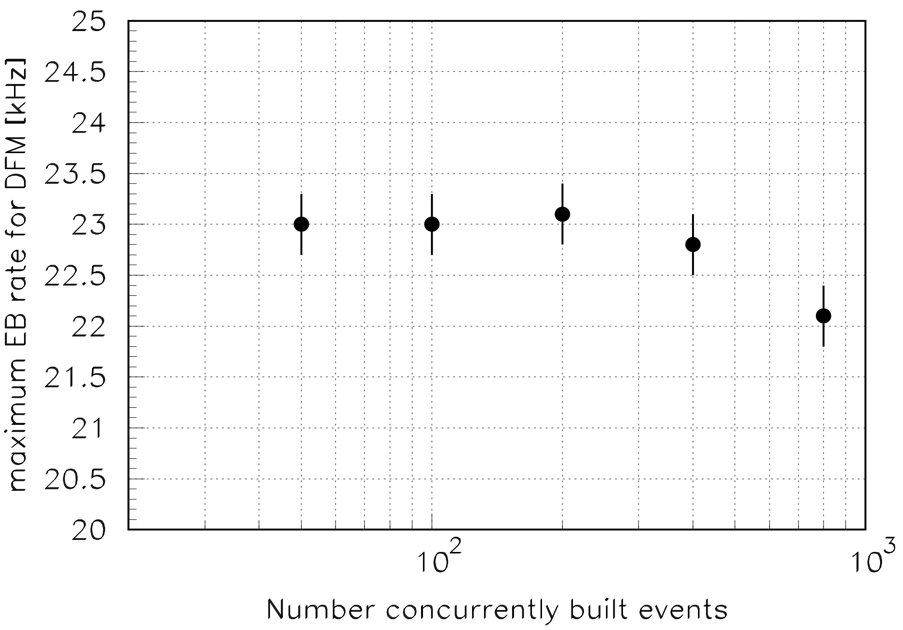


Figure 10.4: DFM performance versus number of concurrently built events for a fixed number of SFIs (50)

**EB rate vs CPU speed** As shown in Fig. 10.5, the DFM performance scales linearly with the CPU speed. This means that the DFM performance is purely CPU bound. This behavior is not surprising, because the DFM sends and receives system messages of limited size and no big data messages. At the observed rate of 25 kHz on a 2.4 GHz CPU both the input and the output bandwidths of the DFM are 1.7 MB/s, which is far below the Gigabit Ethernet capabilities.

**Time-Stamps** A typical example of a time-stamp evaluation is shown in Fig. 10.6. The time the DFM spends on an incoming EoE message was measured. Adding up this result with results from other time-stamp measurements (e.g. receiving a Level 2 decision) allowed the calculation of the EB rate the DFM should be able to sustain. This calculation agreed with the rate measurements presented in this section.
The same procedure was successfully applied for the SFI application as well.

**Summary and Conclusions** The presented results show that the DFM performance is one order of magnitude above the ATLAS requirements (an

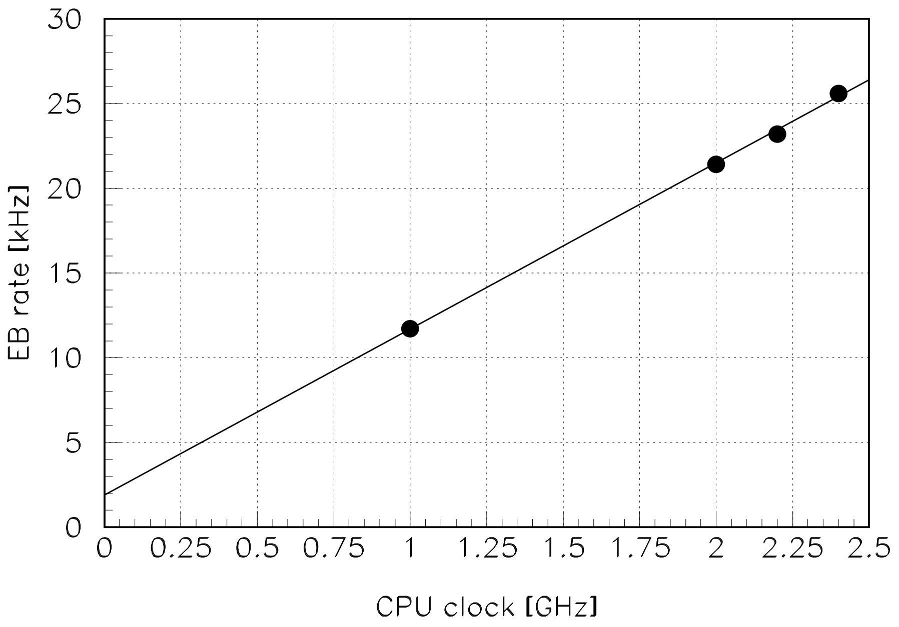


Figure 10.5: DFM performance versus CPU speed

EB rate of 3 kHz)for all measured scenarios. Therefore we conclude that the DFM will not be the bottleneck for the ATLAS EB system. Neither will the DFM be the limit for the SFI Application tests and EB system tests, which will be described below.
The test results show that the DFM scales very well for both varied parameters, the number of concurrently built events and the number of SFIs. As all the DFM internal data structures used for the event bookkeeping are built using STL container classes, this shows that the scalability of STL is sufficient for our requirements in the case of the DFM. In addition, the measurements show that the implementation of the DataCollection Message-passing is scalable, as communicating to more than $10^2$ destinations (on the level of the software, it is not important whether those destinations really exist or are 'ghost' destinations as for this series of measurements) does not result in a big performance penalty.
The question of bus-based vs switch-based ROS has no implication on the DFM, as in the message-flow (see section 6.2) the DFM either exchanges unicast messages with the L2SVs and the SFIs or multicast messages with the ROSs (one single send operation to reach all the destinations).

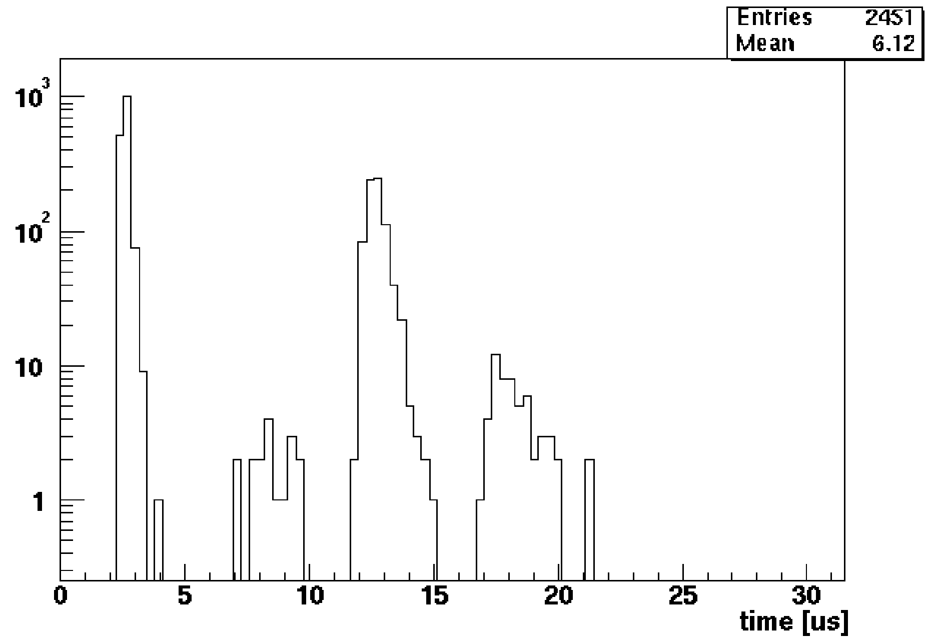


Figure 10.6: Time-stamp evaluation of the DFM receiving an EoE message. There two main peaks, one at 2.5 $\mu$s and one at 12.5 $\mu$s. They represent two different cases: due to the grouping of the clear messages (see section 6.2), a received EoE message may or may not trigger a clear message to be sent to the ROSs. This means that composing a clear message and sending it to the network takes 10 $\mu$s. The two tails at 8 and 18 $\mu$s are caused by the Linux operating system which occasionally interrupts and delays the execution of the DFM process.

### 10.1.2 Subfarm-Input Performance

The main aim of the SFI performance tests was to determine the EB rate a single SFI can sustain and the bandwidth of event data it can absorb.
As the SFI requests and receives the event data from the ROS, differences in the SFI performance between the bus-based ROS scenario and the switch-base ROS scenario can be expected. A switch-based ROS architecture means that every SFI reads data from many more data sources than in a bus-based scenario (one point of access from the EB system per ROL vs. a concentration of 12 ROLs via a PCI BUS). Advocating the switch-based ROS scenario requires showing that reading small data fragments from many sources does not result in a performance problem on the level of the SFI.
In order to understand the aim, the setup and the results of the SFI measurements, it is important to understand the two ROS implementation scenarios, which are described in section 6.3.1, and some technicalities of Gigabit Ethernet explained in section 7.2, especially the significance of the *Ethernet frame*.

#### SFI Test Setup

The setup for the SFI application tests was significantly more complex than in the case of the DFM tests. A DFM assigns events to the SFI. For event data sources ALTEON or FPGA emulators were deployed (see appendix A). The inter-connection to the DFM and the data sources was done via a Gigabit Ethernet network: the ALTEON emulators were connected directly to the central switch as shown in Fig. 10.7, whereas the FPGA emulators were connected via an additional layer of concentrator switches as drawn in Fig. 10.8. Optionally, depending on the test scenario, there was an additional back-to-back Gigabit Ethernet connection between the SFI and an Event-filter emulator. This allows the SFI to be run in full through-put mode, requesting and receiving event fragments from the ROS emulators and sending them to the Event-filter emulator. The SFI, the DFM and the Event-filter emulator ran on the fastest available PCs in the testbed (2.4 GHz clock frequency). As the event data sources were hardware ROS emulators for all tests, the raw Ethernet protocol was used exclusively.
A technical particularity worth noting is the so-called *alias* operation mode of the FPGA and ALTEON ROS emulators. In the switch-based ROS scenario there will be 1600 data sources in the TDAQ system. The number of available ROS emulators was smaller and therefore a realistic number of data sources and realistic event sizes needed to be emulated. The solution was to give no static identity to the ROS emulators, but enable them to act as multiple ROSs. So every emulator was contributing multiple fragments to

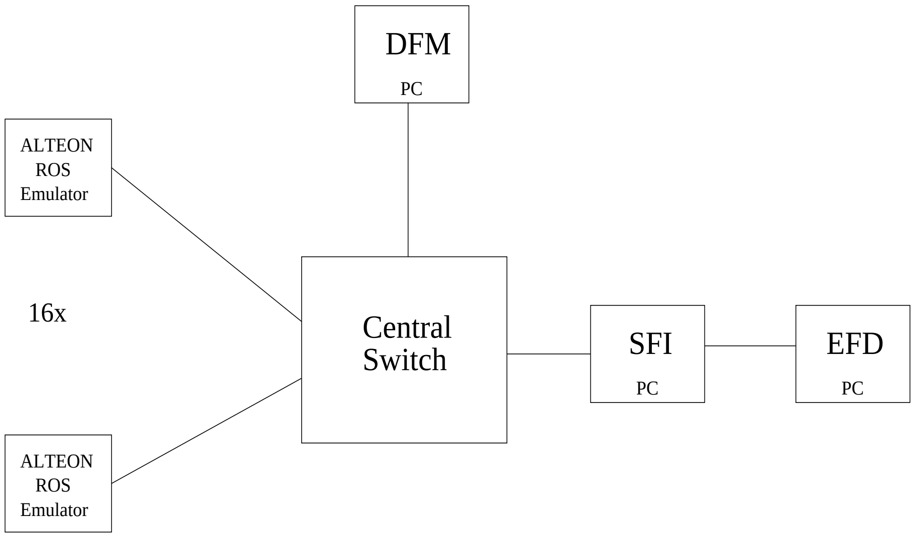


Figure 10.7: SFI test setup: the SFI is connected to 16 ALTEON ROS emulators via a central switch. It is also connected back-to-back to an emulation of the Event-filter.

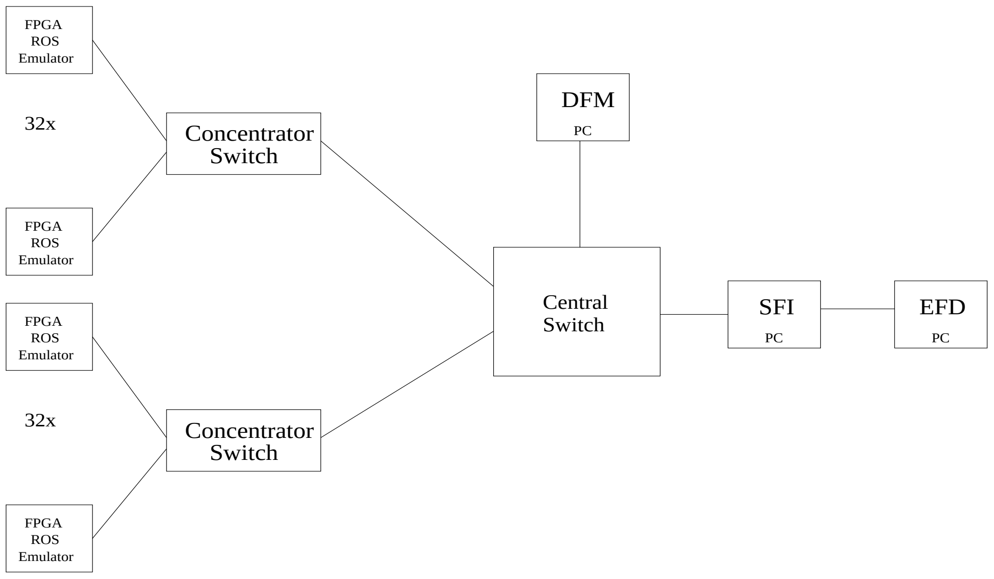


Figure 10.8: SFI test setup: the SFI is connected to two sets of 32 FPGA ROS emulators via a central switch and a concentrator switch. It is also connected back-to-back to an emulation of the Event-filter.

a given event.

### SFI Test Results and Discussion

The measurements done with the SFI were:

- Keeping the number of emulated data sources constant and varying the event size. This means that the size of the single fragments were varied from almost 0 (minimal header information) to a full Ethernet frame. This measurement was provided running against the FPGA ROS emulators acting as 1600 data sources.
- Keeping the event size constant at 2.2 MB[3] and varying the number of emulated data sources. This means a variation of the data fragment size the ROS emulators serve. Varying the fragment size from one to eight Ethernet frames by keeping the total event size constant effectively compares a ROS concentrating one ROL with a ROS concentrating eight ROLs[4]. This comparison quantifies the difference in performance on the level of a single SFI running in a switch-based or in a bus-based ROS scenario. Because this test required the ability of the ROS emulator to send multiframe messages, the ALTEON emulators were used for this test.

**EB Performance vs. Event Size** The results of the first measurement varying the event size while keeping the number of fragments constant are shown in Fig. 10.9. Two sets of data points are plotted: the message rate the SFI sustains and the bandwidth of event data the SFI absorbs. One observes that the SFI is message rate limited, and that this limit is only weakly dependent on the event size. The SFI performance in this setup is purely CPU limited, as the measured maximum input bandwidth into the SFI is 80 MB/s or around 64% of the Gigabit Ethernet line speed.

**SFI EB and throughput performance vs. ROS concentration factor** The plot in Fig. 10.10 shows two series of measurements: the SFI doing input only and the SFI doing throughput to the Event-filter for different ROS concentration factors. Quantifying the effect of staging the EB with a

[3]At the time these measurements were provided, the working assumption on the ATLAS event size was $\sim$ 2 MB. 2.2 MB became an established value for the measurements because it corresponds to 1600 full Ethernet frames

[4]At the time these measurements were provided, the working assumption was that a bus-based ROS concentrates eight ROLs. The current working assumption is that a ROS concentrates 12 ROLs.

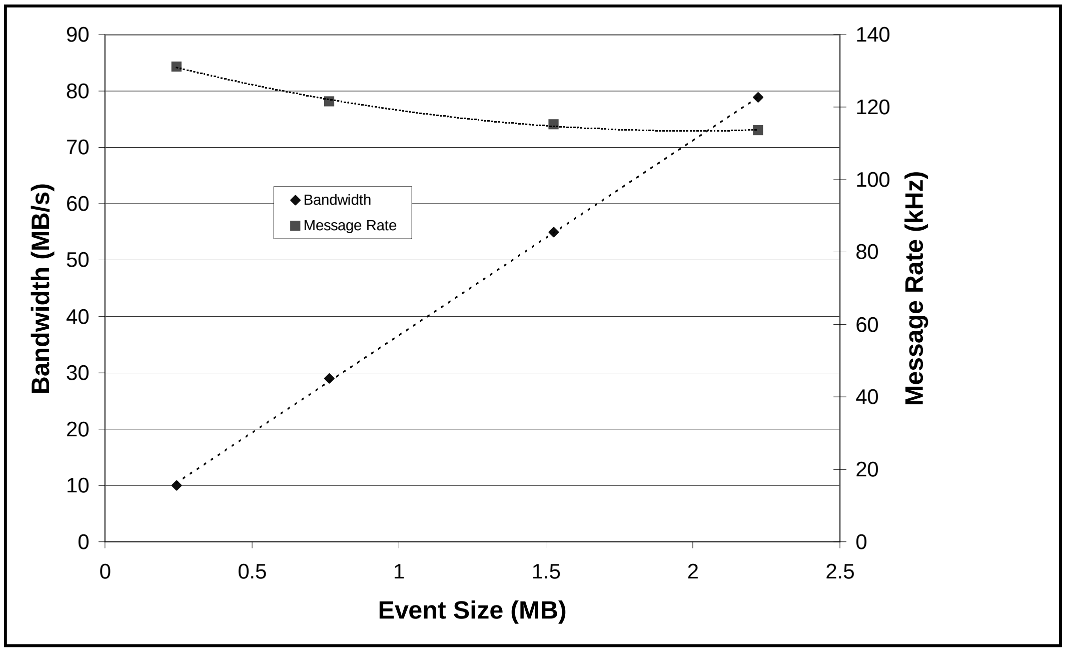


Figure 10.9: SFI EB performance reading data from 1600 sources. The event size was varied.

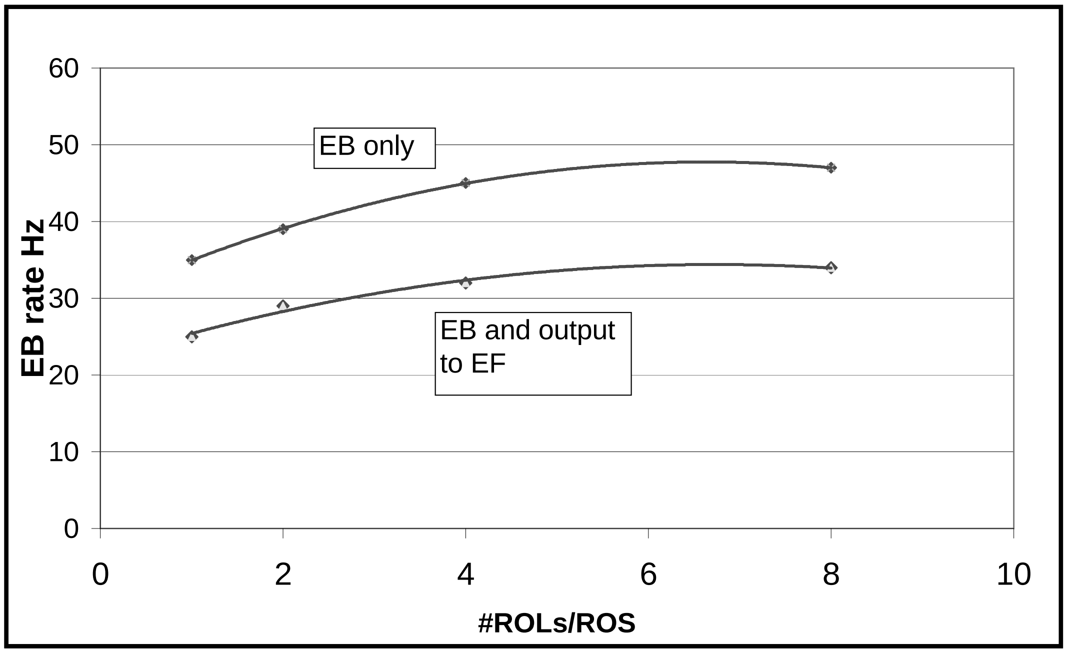


Figure 10.10: SFI EB and throughput performance for different concentration factors (# of ROLs/ROS)

bus-based ROS as described above, a performance gain of 30% in EB rate is seen, going from a ROS concentration factor of 1 (1600 sources, single frame messages) to 8 (200 source, 8 frame messages)[5].

**Summary and Conclusions** The first measurement shows that the SFI performance is event fragment rate limited and that this limit is almost independent of the fragment size (varied within one Ethernet frame). As mentioned above, the measurement shows that the SFI performance is purely limited by its CPU resources.
The second measurement shows a disadvantage in performance for the switch-based ROS scenario. This disadvantage is expected to disappear with faster CPUs, as the Input/Output limitation of the network will then determine the speed of the SFI. The maximum bandwidth which was seen in this measurement was 95 MB/s: in this case the CPUs of the SFI PC were not fully loaded anymore, which indicates that the network I/O limit has been reached, whereas in the other cases the CPUs were fully loaded. Therefore with faster CPUs all the data points on the plot will move from the CPU limited area to the network I/O limited area.
The performance difference between an SFI doing EB only and an SFI doing throughput is significant (35 Hz vs 25 Hz EB rate) and not yet understood. It seems that there is no parallelism when handling two NICs concurrently, one for EB and one for data output to the Event-filter. The results of additional investigations suggest that the NIC drivers are not the problem and that probably the Linux kernel excludes the handling of two NICs by two CPUs in parallel. Tests with more modern Linux SMP kernels need to be provided in the future.

### 10.1.3 Conclusions Drawn from The Application Tests

The EB application tests show, that:

- The DFM performance is sufficient for EB system tests and even for the full ATLAS experiment
- The SFI performance is promising. Depending on the scenario, an SFI can absorb 80 - 95 MB event data per second. The SFI does not reach the ultimate goal of fully exploiting the Gigabit Ethernet capabilities,

[5]The gain in input bandwidth was not 30% but only 19%. This is explained with the difference of event size due to the different number of headers (200 vs 1600 ROS headers per event)

but it drives a big fraction of the Gigabit Ethernet bandwidth. Therefore it can be applied for system tests which are relevant for the EB of the final ATLAS experiment.

Therefore the application tests show that the DFM and the SFI are ready to perform valid EB system tests.

## 10.2 Event Building System Tests

The upper limit of the EB performance is defined by the standalone performance of the DFM and the standalone performance of the SFI. It is an upper limit only because running many SFIs in an EB system could lead to interference effects and therefore disturb and slow each other down. Such interference effects could occur on the level of the data sources, because they have to treat request from multiple SFIs, or in the central switch. Therefore it is important to investigate the performance of an EB system varying the number of SFIs. Ideally, every SFI which is added to the EB system contributes the same EB rate as it would drive standalone, without any interference occurring between the single SFIs. Plotting the EB rate versus the number of SFIs should show a linear, scalable behavior.
An interesting question arises in the context of the two different ROS architecture scenarios: whether routing big multi-frame messages (bus-based ROS) or small single-frame messages (switch-based ROS) through the network will show the same scalability behavior or if there will be more interference observed in the first case. This question will be discussed below.

### 10.2.1 EB System Test Setup

The setup for EB system tests was very similar to the setup for SFI application tests. The differences were:

- More than one SFI in the system, all running on 2.4 GHz PCs. The maximum number of available 2.4 GHz PCs was eight.
- In order to operate a uniform system, the SFIs did EB only. There was no output to the Event-filter and the events were deleted from the SFI memory after they were completely built.

A schematic view on the EB test setup is show in Fig. 10.11.
There were two sets of measurements done:

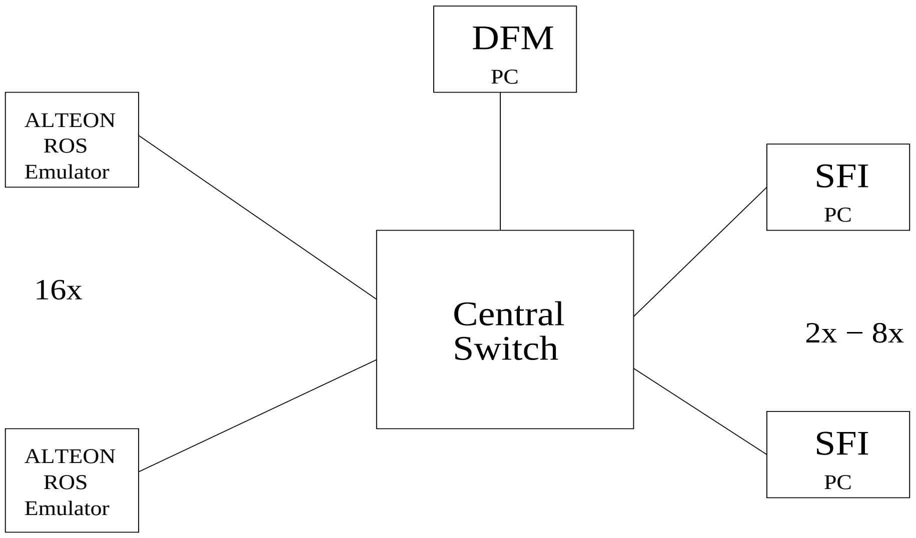


Figure 10.11: EB test setup: $n$ SFIs are connected to 16 data sources via a Gigabit Ethernet switch.

- 1600 fragments per event were collected. The fragment size was one Ethernet frame. No Ethernet flow control was needed to run this setup stably. This measurement reflects the switch-based ROS scenario.
- 200 fragments per event were collected. The fragment size was eight Ethernet frames. Ethernet flow control was needed to run the setup stably. This case reflects the bus-based ROS scenario.

For both scenarios the number of SFIs in the system was increased and the total EB rate was measured. As the event size was 2.2 MB for all measurements, the total EB bandwidth is proportional to the total EB rate.

### 10.2.2 EB System Test Results and Discussion

The results for the two scenarios studied are both presented in Fig. 10.12. In the case of the bus-based ROS scenario, a linear scaling behavior was observed up to eight SFIs. The maximum EB rate achieved was 350 Hz (17% of the ATLAS EB rate) with eight SFIs, each contributing $\sim$ 44 Hz. The speed of the SFI was not exactly uniform due to different NICs. In the case of the switch-based ROS scenario, the scaling was also linear, but the overall performance was lower.
As already indicated when discussing the EB system test results we have to distinguish the two fundamentally different cases:

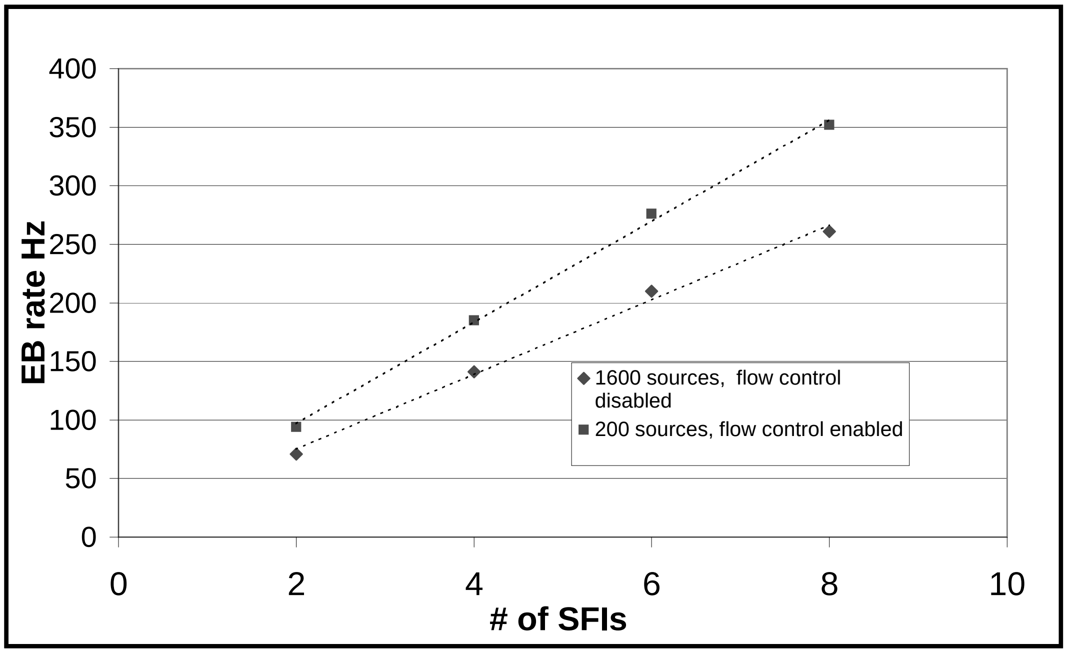


Figure 10.12: EB scalability for 200 (upper data series) and 1600 (lower data series) data sources

- routing big data fragments from few data sources to the SFIs (bus-based ROS)
- routing small data fragments from many data sources to the SFIs (switch-based ROS)

The first case showed a slightly better performance than the second, because the single SFIs gain from the staged EB, as was shown in the previous section 10.1.2. However, routing big data fragments through the network has serious implications on the traffic pattern: The switching network shows an extremely high reliability in message delivery in the case of small data fragments. The SFIs had to take recovery action on a very low rate only (one out of $10^9$ event fragments was re-asked), and the system ran in a stable condition. In the case of big messages, the scalable behavior of the network ends when running more than four SFIs. The rate of message loss becomes high, and the system falls into an unstable condition. As the system becomes unstable, the rate of packet loss is not quantifiable. The problem was overcome by applying the Ethernet flow control mechanism, which issues back-pressure on the sender if buffers in the switch become overloaded. The scalability plot for the bus-based ROS scenario was obtainable only by applying Ethernet flow control. As flow control is a feature of Gigabit Ethernet, it is reasonable to apply it. However, the need to rely on flow control, as was the case for the bus-based ROS measurement, is problematic. Flow control blocks ports of the switch. If it is frequently applied in a running system one gives up the most important scalability feature of the switch: being non-blocking. This would lead to a non-scalability in the system. Such questions need to be further and carefully studied in the future.

# Chapter 11

# Conclusions and Outlook

After having implemented, tested and measured the DataCollection Software, especially the EB part of it, the following conclusions can be drawn:

- Thanks to the framework approach, it was possible to develop the DataCollection software within two years, including extended testing.
- The framework approach will ease future maintenance of the software.
- The performance of the applications built on top of the framework is promising and sufficient for ATLAS, even with today's hardware.
- The scalability behavior of the EB system is promising (linear scaling up to eight SFIs).
- The advantage to the SFI performance of the bus-based ROS approach over the switch-based ROS approach is $\sim 30\%$, due to the shortage of CPU resources in the SFI PC. This shortage will vanish with faster CPUs. As the PCs serving as SFIs at the startup of ATLAS are expected to have 8 GHz CPUs [ATL03] and the presented measurements were performed on 2.4 GHz CPUs, the SFI performace will be the same in both scenarios, limited by the Gigabit Ethernet link into the SFI. Therefore ATLAS should profit from the advantages of the switch-based ROS architecture which were discussed in section 6.3.1. In addition there is a risk that the bus-based ROS approach may not scale, because in this case the EB system relies fully on Ethernet flow control.
- The performance of the ROB communicating to the network (switch-based ROS scenario) needs to be carefully assessed in the future. If this performance is insufficient, the advantages of the switch-based ROS scenario will vanish.

The development process of the DataCollection software is not finished. Our experience with the software and feedback from users (especially at ATLAS test-beam) will lead to a further evolution and improvement of the software. This process will be influenced by the decision on the ROS scenario (bus-based vs switch-based), which the ATLAS TDAQ collaboration has to take at the end of the year 2004.

# Appendix A

# ROS Emulators

For the functional and performance tests of the DataCollection software we needed to send emulated detector data to the DataCollection network. The sources sending Event Fragments were three kinds of emulators.

## A.1 SW ROS Emulator

The SW ROS emulator (ROSE) is a full DataCollection application, built on top of the framework. It provides dummy or preloaded data after receiving a data request message from the SFI or the L2PU (see section 6.2). In addition it counts the number of clears it gets from the DFM. The SW ROS emulator was not used for EB performance tests, but it was needed for the development of the EB applications and for functional tests.

## A.2 FPGA ROS Emulator

The FPGA ROS emulator [LEV03] is a hardware device: an FPGA drives a Fast Ethernet[1] port. Is uses the raw Ethernet protocol of the DataCollection message passing. After receiving a data request message the emulator sends out a fully formatted variable-sized ROS fragment. The size is determined by a field in the header of the DataCollection message. The minimum payload size is 4 Bytes, the maximum payload size is 1460 Bytes.

[1]Ethernet protocol as for Gigabit Ethernet, but the line speed is limited to 100 MBits per second.

## A.3 ALTEON ROS Emulator

The ALTEON emulator [LEV03] is a programmable Gigabit Ethernet NIC. It implements the same functionality as the FPGA emulators. In addition, it is able to send out multiframe messages up to a total size of 64 kB and counts the clear messages it gets from the DFM.

# Appendix B

# Network Protocols

The DataCollection Message-passing is able to exchange messages using three different network protocols: TCP/IP, UDP/IP and Raw Ethernet. A popular reference for network protocols is [STE98].

## B.1 UDP/IP: User Datagram Protocol

UDP is a simple non-reliable protocol. It sends messages from the source to the destination. At the destination a checksum is recalculated and compared with the one calculated at the sender. If the checksums differ, the message will be dropped. As it is a connectionless protocol, there is no feedback to the sender whether the message was valid and has arrived at the destination. Therefore the protocol is not reliable and reliability, if required, has to be added on the application level. The addressing and routing of messages is based on the Internet Protocol (IP). In our specific case, running UDP in a pure Ethernet LAN, the Address Resolution Protocol (arp) is used. IP and arp are explained in section B.4.
Despite its non-reliability, UDP is very adequate to the purpose of the DataCollection, since in a well-dimensioned LAN and operating applications which implement an effective traffic-shaping strategy, message loss can be minimized. The big advantage of UDP is, that it is scalable because the application does not need to serve many connections in parallel, but many sources can send messages to the same port of the destination.

## B.2 TCP/IP: Transmission Control Protocol

TCP provides more services and is significantly more complex that UDP. It establishes a connection between two communication partners. Between

these two partners a byte-stream flows. Reliability is provided by confirming every packet which arrives at the destination and retransmitting the data if no acknowledgment arrives at the source. In addition, TCP provides flow control: the protocol always tells its peer how many bytes of data it is willing to accept from it.
The reliability of TCP may be helpful for system messages in the DataCollection network, as reliability for this kind of messages is vital. However, TCP is not appropriate for EB or RoI Collection due to the following reasons:

- An application has to hold an open connection to all its communication partners in parallel. This is possible in a small system but does not scale.

- As TCP is a byte stream, a partner to partner communication is blocked if a piece of this stream is missing due to packet loss. The transmission can be continued only after the recovery mechanism has acted and the missing piece arrived.

- On the arrival of all transmitted network packets, the reception has to be acknowledged by the destination. In a request-reply traffic scenario as in RoI collection or EB, the acknowledgment for the request is added to the reply message, but the acknowledgment for the reply has to be sent as a single message over the network. This means that in case of running TCP the number of messages in the network is 50% higher compared to the case of a connectionless protocol.

- TCP hides problems in the LAN communication rather than solving them. If a part of the network is a bottleneck or if a destination is overloaded, TCP will nevertheless deliver all data to the destination, but the recovery mechanism will take CPU power at source and destination and the network will be loaded with a lot of additional messages.

More detailed explanations about the usage of TCP/IP in the DataCollection environment are given in [DC-62].

## B.3 Raw Ethernet

Raw Ethernet is a low-level protocol accessing directly the facilities of the Ethernet network technology. As the DataCollection network is a pure Ethernet LAN, it is possible to operate with this low-level protocol without using higher-level protocols like IP. As this is easier to implement on the level of the hardware ROS emulators than UDP or TCP, it was chosen to run the biggest part of the DataCollection performance tests. However, on a Linux operated PC a user needs to have privileged status to be able to run raw Ethernet, which makes it more difficult to maintain in a future system than standard protocols like UDP or TCP.

## B.4 Underlying Protocols: IP and arp

The Internet Protocol is responsible for routing and delivering messages through a network. In addition, it is responsible for splitting the messages into fragments of the maximum supported size of the underlying network technology (e.g. Ethernet) at the sender and to reconstruct the messages at the receiver.
The address resolution protocol is responsible for resolving MAC-addresses if IP runs over Ethernet. arp looks for a host with the given IP address, this host will reply with its MAC-address. This allows IP to hand over the message fragments to the underlying Ethernet protocol with the MAC-address of the destination.

# Appendix C

# Ethernet Switches

Switches route the network traffic from one physical connection to the others. If one wants to push the throughput of a LAN close to the bandwidth specifications of the underlying Ethernet technology, one needs to set up a fully switched network topology. Fully switched means, that no physical network connections are shared between multiple senders and therefore no *collisions* can occur (multiple senders transmitting to the same physical connection at the same time). If one wants to connect a set of PCs with a fully switched network, one needs to connect every PC directly to a switch.
The task of an Ethernet switch is to receive an Ethernet frame from an input port, to look up the destination in the header of the frame, to look up in an internal address table which output port corresponds to the destination's address and to send the frame to this port. While there are ongoing lookups or short-time peaks in the traffic to some destination, the switch must buffer Ethernet frames after receiving them from the input port and before sending them to the output port.
In case of random traffic (every source sending randomly but equally distributed to every destination), the switch ideally should be able to absorb data at line speed at every input port and to serve data at line speed to every output port (in case of Gigabit Ethernet the line speed is 1 GBit/s).
If the traffic is non-random, switch buffers may fill up and overflow. This is for example the case if:

- many senders transmit data to a few or even a single destination.
- $n$ senders transmit bursts of $m$ Ethernet frames[1] to the same destination, even if the overall traffic pattern is well balanced.

[1]The experience gained from the EB system tests described in section 10.2.2 indicates $n \geq 4 \wedge m \geq 4$

If such a case occurs, Ethernet flow control (see section 7.2) issues XOFF commands to the senders, which stops them transmitting any data. As soon as the buffers have recovered from the overflow, an XON is issued to the senders, which then can resume. If flow control is disabled or if the XOFF commands are issued too late, the overflowing buffers need to drop frames, which leads to message loss.

## C.1 BATM T5

The BATM T5 switch [WEB08] was used as a concentrator switch (see section 7.1) when running tests against FPGA emulators (see appendix A). The switch is equipped with 48 Fast Ethernet ports and two Gigabit Ethernet ports. It distributed incoming requests from the two Gigabit ports to the 48 Fast Ethernet ports and concentrated event data coming from the Fast Ethernet ports to the two Gigabit ports.

## C.2 BATM T6

The BATM T6 switch [WEB08] was used as a central switch for Level 2 and EB (see section 7.1). It was equipped with 31 Gigabit Ethernet ports.

# Appendix D

# Glossary

## D.1 Specific Terms

| | |
|---|---|
| Class | Description on which the creation of objects is based. |
| Inheritance | Creating specific classes based on more general classes. The features of the general classes (base classes) are propagated to the specific classes. |
| Kernel | The essential part of Linux or other operating systems, responsible for resource allocation, low-level hardware interfaces, security etc. |
| Semaphore | A semaphore offers a thread the ability to query the number of resources available, e.g. credits for traffic-shaping. |
| Method | An internal function of the class working on members of this class |
| Object | Objects are miniature programs, consisting of both functions (methods) and variables (data members). Every object is described by a class. |
| Pointer | A pointer in C++ is an address describing the location of a data structure. Working with pointers allows memory copies to be avoided, which helps to gain performance. |
| To parse | A program interprets a text file |
| Template | A template is a set of statements in which one or more types are parametric. |

|  |  |
|---|---|
| Traffic-shaping | Regulating the network traffic in such a way, that no connections become a bottleneck and no buffers overflow. |
| Virtual Method | Methods of the base class which are called in case the inherited class does not own a method of the same name. |

## D.2 Acronyms

|  |  |
|---|---|
| API | Application Programming Interface |
| arp | Address Resolution Protocol (IP over Ethernet) |
| CPU | Central processing unit |
| DAQ | Data Acquisition |
| DFM | Dataflow manager |
| EB | Event-building |
| EFIO | Event-filter input/output |
| EoE | End-of-event |
| HLT | High-level trigger: common term for Level 2 *and* Event-filter as the algorithms on both trigger levels are will run as software processes on PCs |
| IP | Internet Protocol |
| LAN | Local Area Network: A local area network is a group of computers and associated devices within a small geographic area e.g. in a building. |
| LAr | Liquid Argon |
| LHC | Large Hadron Collider |
| MAC | Medium access control: the device address in Ethernet |
| OS | Operating System |
| NIC | Network Interface Card |
| ROB | Read-out Buffer |

| | |
|---|---|
| RoI | Region of interest |
| RoIB | Region of Interest Builder |
| ROL | Read-out Link |
| SCT | Semi-conductor Tracker |
| SFI | Subfarm-input |
| SFO | Subfarm-output |
| SMP | Symmetric Multiprocessing |
| STL | C++ Standard Template Library |
| TDAQ | Trigger and Data Acquisition system |
| TRT | Transition Radiation Tracker (Straw Tube Tracker) |

# Bibliography

[ATL94] ATLAS Collaboration, *Technical Proposal for a General-Purpose pp Experiment at the Large Hadron Collider at CERN*, CERN/LHCC/94-43

[ATL99] ATLAS Collaboration, *ATLAS Detector and Physics Performance Technical Design Report*, CERN/LHCC/99-14

[ATL03] ATLAS Collaboration, *ATLAS High Level Trigger, Data Acquisition and Controls Technical Design Report*, CERN/LHCC/2003-22

[BEE98] Chris Bee, David Francis, Livio Mapelli, Robert McLaren, Guiseppe Mornacchi, Jorgen Petersen and Fred Wickens *The Raw Event Format in the ATLAS Trigger & DAQ*, ATL-DAQ-98-129

[BEC03a] Hans Peter Beck et al. *The base-line DataFlow system of the ATLAS Trigger & DAQ*, ATL-COM-DAQ-2003-037

[BEC03b] Hans Peter Beck, *The Scaling Potential of an Optimized ATLAS TDAQ System*, ATL-COM-DAQ-2003-054

[CRA02] Robert Cranfield, Micheal LeVine, Robert McLaren, *ROS User Requirements Document*, ATL-DQ-ES-0007

[DEN90] D. Denegri, *Standard Model physics at the LHC*, Proceedings of the LHC Workshop, Aachen, 1990, CERN 90-10 Vol.I, 1990

[GAD03] Szymon Gadomski et al. *Experience with multi-threaded C++ applications in the ATLAS DataFlow software*, ATL-DAQ-2003-007

[GAL95] Bill O. Gallmeister, *POSIX.4, Programming for The Real World*, O'Reilly & Associates, 1995

[HAE03] Christian Haeberli et al. *The ATLAS-TDAQ DataCollection Software*, ATL-COM-DAQ-2003-014

[JOS99] Nicolai M. Josuttis, *The C++ Standard Library, A Tutorial and Reference*, Addison-Wesley, 1999

[LEH03] Giovanna Lehmann et al. *The DataFlow System of the ATLAS Trigger and DAQ*, ATL-COM-DAQ-2003-018

[LEV03] Micheal J. LeVine et al. *Validation of the ATLAS Trigger/DAQ Network architecture using hardware data emulators*, ATL-DAQ-2003-009

[SIN97] Charan Singh, *Gigabit Ethernet Handbook*, McGraw-Hill, 1997

[STA03] Stefan Stancu et al. *Network Performance Investigation for the ATLAS Trigger/DAQ*, ATL-DAQ-2003-008

[STE98] W. Richard Stevens, *UNIX Network Programming*, Second Edition, Prentice Hall PTR, 1998

[STR00] Bjarne Stroustrup, *The C++ Programming Language*, Special Edition, Addison-Wesley, 2000

[TIE02] Brian Tierney and Dan Gunter, *NetLogger: a toolkit for distributed system performance tuning an debugging*, LBNL Tech Report LBNL-51276, 2002

[DC-01] Reiner Hauser, *Application Framework Strawman Document*, DC Note 01, ATL-DQ-ES-0016

[DC-02] Reiner Hauser, *OS Layer Requirements*, DC Note 02, ATL-DQ-ES-0017

[DC-03] Reiner Hauser, *Configuration Database Requirements*, DC Note 03, ATL-DQ-ES-0018

[DC-04] Andre Bogaerts, Per Werner and Haimo Zobernig, *Error Reporting Requirements*, DC Note 04, ATL-DQ-ES-0019

[DC-05] Per Werner, *Requirements for System Monitoring Interface*, DC Note 05, ATL-DQ-ES-0021

[DC-06] Haimo Zobernig, *Application Control Requirements*, DC Note 06, ATL-DQ-ES-0022

[DC-08] Reiner Hauser and Hans Peter Beck, *Requirements for the Message Passing Layer*, DC Note 08, ATL-DQ-ES-0024

[DC-09] Jim Schlereth and Micheal LeVine, *LVL2 Supervisor Requirements*, DC Note 09, ATL-DQ-ES-0025

[DC-10] Christian Haeberli and Hans Peter Beck, *The DFM Requirements*, DC Note 10, ATL-DQ-ES-0026

[DC-11] Reiner Hauser, *Design of the Configuration Database Interface*, DC Note 11, ATL-DQ-ES-0027

[DC-12] Hans Peter Beck and Christian Haeberli, *Message Flow: High-Level Description*, DC Note 12, ATL-DQ-ES-0028

[DC-13] Reiner Hauser, *Design of the Message Passing Interface*, DC Note 13, ATL-DQ-ES-0029

[DC-14] Jim Schlereth, *RoI Builder Requirements*, DC Note 14, ATL-DQ-ES-0030

[DC-15] Christian Haeberli and Hans Peter Beck, *Error Reporting Design*, DC Note 15, ATL-DQ-ES-0031

[DC-16] Christian Haeberli, Hans Peter Beck and Remigius Mommsen, *The SFI Design*, DC Note 16, ATL-DQ-ES-0032

[DC-19] Andre Bogaerts and Fred Wickens, *LVL2 Processing Unit Application Design*, DC Note 19, ATL-DH-ES-0009

[DC-20] Christian Haeberli, *Error Reporting HOWTO*, DC Note 20, ATL-DQ-OR-0001

[DC-21] Reiner Hauser, *Message Dispatching in the DC Framework*, DC Note 21, ATL-DQ-ES-0034

[DC-22] Hans Peter Beck and Fred Wickens, *Message Format*, DC Note 22, ATL-DQ-ES-0035

[DC-23] Christian Haeberli, *The DFM Design*, DC Note 23, ATL-DQ-ES-0036

[DC-32] Andre Bogaerts, Weidong Li, Fred Wickens, Per Werner and Haimo Zobernig, *Instrumentation for Timing Measurements of the DC Software*, DC Note 32, ATL-DQ-EN-0007

[DC-33] Per Werner, *System Monitoring in the DC Framework*, DC Note 33, ATL-DQ-EN-0008

[DC-34] Per Werner and Andre Bogaerts, *Pseudo-ROS Requirements*, DC Note 34, ATL-DQ-ES-0039

[DC-35] Hans Peter Beck, Christophe Meessen, Remigius Mommsen, Andrea Negri, Sarah Wheeler and Francois Touchard, *Protocol Specifications of Event Transfer between DataCollection and EventHandler*, DC Note 35, ATL-DQ-ES-0040

[DC-37] Szymon Gadomski and Hans Peter Beck, *EventStorage Requirements* , DC Note 37, ATL-DQ-ES-0041

[DC-38] Szymon Gadomski and Hans Peter Beck, *SFO Requirements*, DC Note 38, ATL-DQ-ES-0042

[DC-39] Jim Schlereth, *RoI Builder Control Software Design*, DC Note 39, ATL-DQ-ES-0043

[DC-41] Per Werner, *Pseudo-ROS Design Document*, DC Note 41, ATL-DQ-ES-0044

[DC-43] Reiner Hauser and Hans Peter Beck, *ATLAS TDAQ/DCS DataCollection Architectural Design*, DC Note 43, ATL-DQ-ES-0046

[DC-44] Remigius Mommsen, *Detailed Design of the SFI*, DC Note 44, ATL-DQ-ES-0047

[DC-45] Reiner Hauser and Hans Peter Beck, *ATLAS TDAQ DataCollection Requirements*, DC Note 45, ATL-DQ-ES-0048

[DC-46] Jim Schlereth, *Level 2 Supervisor Design*, DC Note 46, ATL-DQ-ES-0049

[DC-48] Hans Peter Beck, Andre Bogaerts, David Francis, Szymon Gadomski, Christian Haeberli, Markus Joos, Valeria Perez Reale, *A High Resolution Time-Stamping Library for TDAQ*, DC Note 48, ATL-DQ-EN-0010

[DC-52] Reiner Hauser, *Code Generation for Configuration Database Objects*, DC Note 48, ATL-DQ-ES-0050

[DC-54] Piotr Golonka, *Linux network performance study for the ATLAS Data Flow System*, DC Note 54, ATL-DQ-TR-0001

[DC-57] Piotr Golonka and Krzysztof Korcyl, *Calibration of the ATLAS DataCollection Component Models*, DC Note 57, ATL-DQ-TP-0001

[DC-59] Hans Peter Beck, Bob Dobinson, Krzysztof Korcyl, Micheal LeVine , *ATLAS TDAQ: A Network-based Architecture*, DC Note 01, ATL-DQ-EN-0014

[DC-60] Christian Haeberli, Szymon Gadomski and Hans Peter Beck, *ATLAS DataCollection, Event Building Test Report*, DC Note 60, ATL-DQ-TR-0003

[DC-62] Richard Hughes-Jones, *The use of TCP/IP for Real-Time Messages in ATLAS Trigger/DAQ*, DC Note 62, ATL-DQ-EN-0015

[WEB01] *The ATLAS Home Page*, http://atlasinfo.cern.ch/Atlas

[WEB02] *The LHC Home Page*, http://lhc-new-homepage.web.cern.ch/lhc-new-homepage/

[WEB03] *The ATLAS TDAQ DataCollection Home Page*, http://atlas.web.cern.ch/Atlas/GROUPS/DAQTRIG/DataFlow/DataCollection/DataCollection.html

[WEB04] *The Fermilab Home Page*, http://www.fnal.gov

[WEB05] Franco Bedeschi *Physics at Tevatron*, Summer Student Lectures 2003, http://agenda.cern.ch/fullAgenda.php?ida=a034358

[WEB06] Fabiola Gianotti *LHC Physics*, Summer Student Lectures 2003, http://humanresources.web.cern.ch/HumanResources/external/recruitment/Summies/LectProgr2003.htm

[WEB07] *The ATLAS Online Software Home Page* http://atlas-onlsw.web.cern.ch/Atlas-onlsw/OnlineSoft.htm

[WEB08] *The BATM Home Page* http://www.batm.com

# Acknowledgments

I would like to express my gratitude to:

- Prof. Dr. Klaus Pretzl for giving me the opportunity to work for the ATLAS experiment and for his continued support
- Hans Peter Beck for pushing the vision of the DataCollection software through many storms, for his advice and guidance
- Szymon Gadomski for the harmonic and efficient collaboration and for proof-reading my thesis
- Reiner Hauser for the technical advice and guidance
- Andrew Lankford for making the realization of the DataCollection vision possible during his term as the TDAQ project leader
- Stefan Stancu and Micheal LeVine for providing and supporting the hardware ROS emulators
- Irene Neeser for proof-reading my thesis
- Marian Zurek for the system administration of the wedding-list testbed
- Fred Wickens for being the co-referee for this thesis
- The Bern ATLAS team - Sonja, Valeria, Alina, Hans Peter, Szymon and Gianluca - for fruitful discussions and fun during coffee breaks and spare time we spent together
- The ATLAS Dataflow Group for the not always easy but successful collaboration
- The wedding-list working group and the wedding-list institutes for defining and funding the testbed
- Last but not least, I thank Christina Fontana from all my heart for her love, her support and her countless journeys to Geneva

# Curriculum Vitae

| | |
|---|---|
| Name: | Christian Häberli |
| Birthday: | June 3rd, 1976 |
| Place of Birth: | Bern, BE, Switzerland |
| Places of Origin: | Münchenbuchsee, BE and Zürich, ZH |
| Nationality: | Swiss |

| | |
|---|---|
| 1983-1989 | Primary and secondary school |
| 1989-1991 | Lower grammar school |
| 1991-1996 | Grammar school, Matura type C (mathematics and sciences) |
| 1996-2000 | Student of physics, mathematics and astronomy at the University of Bern, Switzerland |
| 1999-2000 | Diploma student under the supervision of Prof. Dr. Klaus Pretzl at the Laboratory for High Energy Physics at the University of Bern |
| 2001-2003 | PhD student under the supervision of Prof. Dr. Klaus Pretzl at the Laboratory for High Energy Physics at the University of Bern, working at CERN |