DISSERTATION

submitted to the
Combined Faculty of Natural Sciences and Mathematics
of the
Ruprecht–Karls University
Heidelberg
for the degree of
Doctor of Natural Sciences

put forward by
Tommaso Colombo, MSc
from
Merate, Italy

Date of oral exam:

Optimising the data-collection time of a large-scale data-acquisition system

Advisor: Prof. Dr. Holger Fröning

Abstract

Data-acquisition systems are a fundamental component of modern scientific experiments. Large-scale experiments, particularly in the field of particle physics, comprise millions of sensors and produce petabytes of data per day. Their data-acquisition systems digitise, collect, filter, and store experimental signals for later analysis. The performance and reliability of these systems are critical to the operation of the experiment: insufficient performance and failures result in the loss of valuable scientific data.

By its very nature, data acquisition is a synchronous many-to-one operation: every time a phenomenon is observed by the experiment, data from its various sensors must be assembled into a single coherent dataset. This characteristic yields a particularly challenging traffic pattern for computer networks dedicated to data acquisition. If no corrective measures are taken, this pattern, known as *incast*, results in a significant underutilisation of the network resources, with a direct impact on a data-acquisition systems' throughput.

This thesis presents effective and feasible approaches to maximising network utilisation in data-acquisition systems, avoiding the incast problem without sacrificing throughput. Rather than using abstract models, it focuses on an existing large-scale experiment, used as a case-study: the ATLAS detector at the Large Hadron Collider.

First, the impact of incast on data-acquisition performance is characterised through a series of measurements performed on the actual data-acquisition system of the ATLAS experiment. As the size of the data sent synchronously by multiple sources to the same destination grows past the size of the network buffers, the throughput falls. A simple but effective mitigation is proposed and tested: at the application-layer, the data-collection receivers can limit the number of senders they simultaneously collect data from. This solution recovers a large part of the throughput lost to incast, but introduces some performance losses of its own.

Further investigations are enabled by the development of a complete packet-level model of the ATLAS data-acquisition network in an event-based simulation framework. Comparing real-world measurements and simulation results, the model is shown to be accurate enough to be used for studying the incast phenomenon in a data-acquisition system.

Leveraging the simulation model, various optimisations are analysed. The focus is kept on practical software changes, that can realistically be deployed on otherwise unmodified existing systems. Receiver-side traffic-shaping, incast- and traffic-shaping-aware work scheduling policies, tuning of TCP's timeouts, and centralised network packet injection scheduling are evaluated alone and in combination. Used together, the first three techniques result in a very significant increase of the system's throughput, which gets within 10% of the ideal maximum performance, even with a high network traffic load.

Zusammenfassung

Datenerfassungssysteme sind fundamentale Komponenten moderner wissenschaftlicher Experimente. Großexperimente der Elementarteilchenphysik nutzen Millionen von Sensoren und erzeugen Petabyte Daten pro Tag. Deren Datenerfassungssysteme digitalisieren, sammeln, filtern und speichern die Daten zur späteren Analyse. Die Funktion und Zuverlässigkeit dieser Systeme sind kritisch für den Betrieb des Experiments. Schlechte Funktionalität führt zum Verlust von wertvollen wissenschaftlichen Daten.

Bei Datenerfassungssystemen kommunizieren viele Systeme synchron mit einem einzelnen System: so werden bei interessanten Ereignissen die Daten vieler Sensoren zu einem kohärenten Datensatz zusammengefügt. Diese Charakteristik führt zu einem herausfordernden Datentransferschema für Computer Netzwerke, die an Datenerfassungssystemen angeschlossen sind. Werden keine geeigneten Maßnahmen ergriffen, kommt es zum Incast. Was zu einer Unterauslastung des Netzwerkes führt, welche den Datendurchsatz im Datenerfassungssystem reduziert.

Diese Arbeit präsentiert effektive und machbare Ansätze die Netzwerkausnutzung in Datenerfassungssystemen zu maximieren, indem der Incast reduziert wird bei gleichzeitiger Erhaltung des Datendurchsatzes. Anstatt an abstrakten Modellen dies zu beschreiben, wird die Studie an dem existierenden ATLAS Detektor am Large Hadron Collider durchgeführt.

Erst wird der Einfluss vom Incast auf die Datenerfassungsleistung durch eine Reihe von Messungen an dem Datenerfassungssystem des ATLAS-Experiments charakterisiert. Wenn die Größe der Daten, welche synchron von mehreren Quellen an das gleiche Ziel gesendet werden, wächst, so sinkt der Datendurchsatz abhängig von der Größe der Netzwerkpuffer. Eine einfache aber wirksame Verbesserung wird vorgeschlagen und getestet: auf der Anwendungsebene können die Empfänger die Anzahl der Absender beschränken, von denen sie gleichzeitig Daten sammeln. Diese Lösung kann einen Groß-

teil des durch Incast verloren gegangenen Datendurchsatzes wiederherstellen, führt jedoch auch zu eigenen Leistungseinbußen.

Weitere Untersuchungen wurden durch die Entwicklung eines vollständigen Paket-Level-Modells des ATLAS-Datenerfassungsnetzwerkes in einem ereignisbasierten Simulationsframeworks ermöglicht. Vergleiche von realen Messungen und Simulationsergebnissen zeigen, dass das Modell genau genug ist, um zur Untersuchung des Incast-Phänomens in einem Datenerfassungssystem verwendet zu werden.

Mithilfe des Simulationsmodells werden verschiedene Optimierungen analysiert. Der Fokus liegt auf praktischen Softwareänderungen, die realistisch auf ansonst unmodifiziert bestehenden Systemen implementiert werden können. Dazu gehören empfängerseitigen Traffic-Shaping, Incast- und Traffic-Shaping-bewusste Arbeitsplanung, Abstimmung der Timeouts von TCP und zentralisierte Netzwerkpaket Injektionsplanung. Diese werden alleine und in Kombination ausgewertet. Zusammen benutzt ergeben die ersten drei Techniken, selbst bei hoher Netzwerkauslastung, eine sehr signifikante Erhöhung des Datendurchsatzes, der innerhalb von 10% um den Idealwert nahe der maximalen Leistung liegt.

Contents

1 Introduction 13

2 Background: data-acquisition networks 17

2.1 Data-acquisition systems . . . 17
2.1.1 Performance targets . . . 18
2.1.2 Traffic pattern . . . 19
2.2 Common network technologies in data acquisition . . . 19
2.2.1 Ethernet and IP . . . 20
2.2.2 The Transmission Control Protocol (TCP) . . . 28
2.3 The *incast* pathology . . . 32
2.4 Related works: incast avoidance and mitigation . . . 33

3 The ATLAS experiment at the Large Hadron Collider 37

3.1 High-energy physics . . . 37
3.2 The Large Hadron Collider . . . 38
3.2.1 Construction . . . 38
3.2.2 Physics performance . . . 40
3.3 The ATLAS Experiment . . . 42
3.3.1 Detector layout . . . 43
3.3.2 Inner detector . . . 45
3.3.3 Calorimeters . . . 47
3.3.4 Muon spectrometer . . . 49
3.4 The ATLAS trigger and data-acquisition system . . . 52
3.5 Front-end . . . 53
3.6 First-level trigger . . . 54
3.6.1 Calorimeter trigger . . . 55

3.6.2 Muon trigger . . . 56
3.7 Data-Acquisition and High-Level Trigger . . . 57
3.7.1 Data format . . . 59
3.7.2 Readout System . . . 59
3.7.3 High-Level Trigger . . . 60
3.7.4 Data-Collection Manager . . . 60
3.7.5 Data Logger . . . 62
3.7.6 Network . . . 62
3.8 The ATLAS Data-Acquisition messaging system . . . 63
3.8.1 Requirements . . . 63
3.8.2 Existing solutions . . . 65
3.8.3 Implementation . . . 66
3.8.4 Benchmarks . . . 67

4 Static traffic shaping for current data-acquisition systems 73
4.1 Performance issues in data-acquisition networks . . . 73
4.2 Evaluation of the impact of the incast pathology on data-acquisition performance . . . 74
4.2.1 Measurement set-up . . . 75
4.2.2 Results . . . 79
4.3 Request-side traffic shaping . . . 86
4.3.1 Incast mitigation . . . 87
4.3.2 Effectiveness evaluation . . . 88

5 Simulation model 93
5.1 Model development . . . 93
5.1.1 Hosts . . . 94
5.1.2 Applications . . . 95
5.1.3 Network switches . . . 97
5.1.4 Complete model . . . 98
5.1.5 Parameters . . . 98
5.1.6 Runtime . . . 101
5.2 Model validation . . . 102
5.2.1 Goals . . . 102
5.2.2 Analysis and comparison of measured and simulated results . . . 103

6 Enhancements for next-generation data-acquisition systems **113**
6.1 Work assignment policies 113
6.2 Variable fragment sizes 115
6.3 Reducing TCP's minimum retransmission time-out 120
6.4 Centralised traffic scheduling 123
6.4.1 Scheduler timing 129
6.4.2 Clock synchronisation 131
6.4.3 Scheduler granularity 132
6.5 Switch buffer space 135
6.6 Discussion 138

7 Conclusion **141**

Acknowledgements **145**

Bibliography **147**

1 Introduction

Contemporary particle physics experiments consist of tens of millions of sensors, producing very large amounts of data (up to tens of petabytes per day). In these experiments, the data-acquisition (DAQ) system is responsible for gathering the data from the various sensors of the experiment, collating the data fragments into coherent sets, reducing the data volume by compressing and discarding redundant and non-interesting information, and safely storing the relevant data for subsequent analysis. The performance and reliability of data-acquisition systems are critical to the functioning of the experiment: failures and poor performance result in the permanent loss of extremely valuable experimental data.

Data-acquisition systems usually comprise two stages. The first stage interfaces directly with the experiment's sensors to perform signal treatment and digitisation. The second stage aggregates all the data sources into one coherent output and selects scientifically interesting data for permanent storage. Naturally, the first stage is tightly coupled to the experiment's sensors. Accordingly, it usually consists of custom electronics and point-to-point links specifically designed for the experiment in question. On the other hand, for cost, flexibility, and maintainability reasons, the second stage consists of software running on a distributed computer cluster, which can be built with commercial off-the-shelf (COTS) components.

The performance of a data-acquisition system depends on two main quantities:

- the data-processing time, i.e. the time used for signal treatment, digitisation, and data selection;
- the data-collection time, i.e. the time used for gathering, collating, and transferring data in the system.

A reduction of either of these time intervals directly translates to an increase of the system's throughput. The data-processing time depends on factors that are usually specific to a particular experiment. The data-collection time, instead, is essentially determined by the design of the data-acquisition system alone.

By its very nature, data acquisition is a synchronous many-to-one operation: for each phenomenon observed by the experiment, signals from all the experiment's sensors must be assembled into a single coherent dataset. This requirement is particularly challenging for networks in data-acquisition clusters. If no corrective measures are taken, this communication pattern, known as *incast*, causes a severely inefficient utilisation of the network, which in turn leads to large data-collection times.

This thesis presents effective and feasible approaches to minimising the data-collection time in data-acquisition clusters, avoiding the incast problem without sacrificing throughput. Rather than using abstract models, it focuses on an existing experiment, used as a case-study: the ATLAS detector at the Large Hadron Collider. Due to the mission-critical nature of data-acquisition systems, a systematic study of their performance envelope is often impeded by operational constraints, such as system availability requirements or limited opportunities of performing hardware or system software modifications. Therefore, a two-pronged strategy is employed: the results of a measurement campaign on the existing ATLAS data-acquisition system are used to validate an event-based simulation model of the system itself. Once the simulation model is proven to be accurate enough to reliably reproduce the key traits of the system, it is used to evaluate possible strategies to reduce the the system's data-collection time.

This work makes the following contributions:

- **Design and development of the ATLAS data-acquisition messaging library and applications** – In order to operate, data-acquisition systems require a simple but high-performance communication layer providing reliable, in-order, point-to-point message delivery. Many existing solutions such as message brokers, remote procedure call middleware, or high-performance computing middleware would satisfy the requirements, but they come at the price of additional, unnecessary complexity. A leaner custom solution can provide easier troubleshooting and increased control over the network usage patterns. The ATLAS messaging library and data-collection software were developed as a preliminary part of this thesis work. They have been successfully used in production since 2015. Their requirements, design, and implementation are discussed in section 3.8.
- **Characterisation of the impact of incast on data-acquisition performance and**

its mitigation with receiver-side traffic shaping – The effects of the synchronous many-to-one nature of data acquisition are explained and characterised using measurements performed on the actual data-acquisition system of the ATLAS experiment. The measurements also show that the incast problem can be mitigated at the application layer: the data-collection receivers can limit the number of senders they simultaneously collect data from. Assuming that the senders have enough buffering capacity available, this simple receiver-side traffic-shaping strategy is shown to be effective in spreading the many-to-one communication over a larger time interval, increasing the network usage efficiency. This work is shown in chapter 4 and was published in [24].

- **Development and validation of a simulation model of the ATLAS data-acquisition network –** An event-based packet-level simulation model is developed, with the goal of investigating additional solutions to the incast problem in data acquisition. Comparing the simulation results with real-world measurements, the model is shown to reproduce the behaviour of the system accurately. The model and its validation tests are described in detail in chapter 5 and were presented in [23].

- **Practical and effective mitigation of incast in data acquisition** - The simulation model is leveraged to test the impact of various other solutions aimed at further reducing the system's data-collection time. Many kinds of technical modifications to a data-acquisition cluster can potentially result in a reduction or elimination of the incast problem. This contribution focuses on practical software changes, that can be realistically deployed on existing systems. The considered modifications are: receiver-side traffic-shaping (already mentioned above), incast- and traffic-shaping-aware work scheduling policies, reduction of TCP's minimum retransmission timeout, and centralised network packet injection scheduling. The results obtained show that a combination of the first three techniques yields a very significant reduction of the data-collection time, coming close to the ideal system performance. This evaluation is discussed in chapter 6 and some of its results were published in [25].

This thesis is organised as follows. In chapter 2, the building blocks of a data acquisition system are introduced, with special attention given to the most common network technologies and traffic patterns in data acquisition clusters. Since most of this thesis deals with data acquisition networks based on TCP/IP over Ethernet, a brief introduction to these protocols is given, highlighting the features that are most relevant in this context. Finally, the incast pattern is introduced, and related works that like this thesis aim at

eliminating or mitigating incast are discussed. In chapter 3 the main components of the ATLAS experiment are outlined, and the data acquisition system is described in detail. The ATLAS messaging layer and data-collection software, which were developed as a preliminary part of this thesis work, are highlighted. The series of measurements aimed at characterising the impact of incast on data-acquisition performance is shown in chapter 4. In the same chapter, a simple receiver-side traffic-shaping strategy is evaluated and shown to successfully mitigate the effects of the incast problem. However, even with this improvement, the data-collection time of the system is still suboptimal. Further solutions are investigated using an event-based packet-level simulation model, described and validated in chapter 5. In chapter 6, the model is used to evaluate the impact of incast-mitigation solutions that can be deployed on existing data-acquisition systems. The merits of each solution are discussed in the same chapter, followed by the conclusion in chapter 7.

Background: data-acquisition networks

2

Most modern scientific experiments rely on data-acquisition systems to digitise, collect, and store experimental signals for later analysis. This chapter provides an introduction to the basic functions of a data-acquisition system, specially focusing on the computer networks that are a key part of medium- and large-scale data-acquisition systems. In particular, the typical data-acquisition traffic pattern is defined, and the network protocols used in most data-acquisition systems are briefly discussed. The so-called incast pathology brought about by data-collection operations is described, and related works that offer incast mitigation or avoidance solutions are discussed. Their limits are also evidenced.

2.1 Data-acquisition systems

The data-acquisition process consists of the following consecutive operations:

- Analogue signal treatment
- Signal digitisation
- Digital signal treatment
- Data collection
- Data selection
- Data storage

While, the signal treatment, digitisation, and data-storage operations are self-explanatory, the meanings of "data collection" and "data selection" are worth specifying. Most experiments do not consist of a single sensor. Instead, multiple sensors measure different quantities of the same phenomenon. In such cases, the data-acquisition system must include one or more data-collection steps: i.e., it must gather all the different measurements from their sources and ensure that they all refer to the same observation. In some experiments, the amount of data produced is often so large that it is impractical to store all of it for later analysis. In such cases, the data-acquisition system is also used as a data-selection system: after a fast analysis, only measurements that are considered scientifically interesting are stored.

Just like the experiments they support, data-acquisition systems range from small and simple, consisting of a single computer with a single data-acquisition device, to large and complex, comprising many racks of electronics, computers, and network equipment. Naturally, the concrete implementation of the data-acquisition operations listed above depends on the experiment's requirements. Analogue signal treatment and digitisation are normally handled with specialised electronics interfacing a sensor and a computer. Digital signal treatment, data collection, and data storage are implemented with a mix of specialised hardware and software running on commercial off-the-shelf (COTS) computers and networks.

Generally speaking, specialised hardware is more expensive and less flexible than a combination of software and COTS hardware, so the latter is preferred if it can guarantee sufficient performance. Especially for large-scale experiments, this preference results in data-acquisition systems that include large computer networks. This thesis focuses on these COTS data-acquisition networks.

2.1.1 Performance targets

Data-acquisition systems operate under strong latency and throughput constraints. As a general rule, the average throughput of the system must match the output rate of the experiment. The system's buffers must be large enough to absorb both output bursts from the experiment and spikes in the system's own latency. If any of this conditions is not respected, potentially interesting and valuable experimental data is lost. Even worse, if the violation of the conditions corresponds to a specific situation of the experiment, the data loss might introduce a bias in the experimental results. For example, if a certain phenomenon measured by the experiment causes the experiment's sensors to

produce data at a higher throughput than the data-acquisition system can handle, the acquired dataset will under-represent that phenomenon with respect to others.

Data-acquisition throughput and latencies depend on two main factors:

- the data-processing time, i.e. the time spent formatting, analysing and selecting the data;
- the data-collection time, i.e. the time spent gathering and moving data in the system.

Obviously, a reduction in either of these two quantities leads to a corresponding increase in the average data-acquisition throughput. The data-processing time is normally dictated by the complexity and efficiency of the algorithms specific to the experiment. No general solutions can be applied to reduce it. The data-collection time, instead, is largely determined by the design and effectiveness of the data-acquisition system itself.

2.1.2 Traffic pattern

Data-acquisition networks have to deal with a particularly problematic traffic pattern:

- The communication pattern is many-to-one.
- Data fragments are transmitted in small bursts, rather than a smooth flow.
- The data transmission bursts happen synchronously, i.e. all data sources transmit data at the same time.

These three characteristics do not reflect a design error, but rather the nature of data acquisition itself. The purpose of a data-acquisition system is to gather together data fragments from the different sensors that make up an experiment. Therefore, at least in one stage of the system, the traffic pattern will be necessarily many-to-one. The timings of the data transfers are also normally driven by the experiment itself, which in general does not produce a continuous flow of data. Instead frequent bursts of data are generated in correspondence with the experiment detecting and measuring a certain phenomenon.

2.2 Common network technologies in data acquisition

Unless special requirements dictate otherwise, data-acquisition networks are usually based on the industry-standard protocol stack made up of: Transport Control Protocol

(TCP) on the transport layer, Internet Protocol (IP) on the network layer, and Ethernet on the data-link and physical layers. All the protocols in the stack are open standards supported by a multitude of hardware and software vendors. Although data-acquisition networks based on different protocols, such as InfiniBand, exist, the "standard" stack is usually preferred due to cost and availability of expertise to design and maintain the system.

In the following sections, some of the aspects of this protocol stack that are most relevant to data acquisition are briefly presented. This is in no way meant to be a comprehensive overview of these complex protocols. In-depth presentations are available for example in [61] for Ethernet and [28] for the whole stack.

2.2.1 Ethernet and IP

The Ethernet family of technologies were introduced commercially in the 1980s and adopted by the IEEE as standard number 802.3 [34]. At the time, Ethernet was a local-area network (LAN) technology that connected computers with a shared transmission medium (a single coaxial cable attached to all computers in the network). The defining trait of Ethernet was its distributed media access control scheme, known as Carrier Sense Multiple Access with Collision Detection (CSMA/CD). With continuous updates and extensions, the standard has not only kept the pace of technological evolution, but also expanded towards other areas of application such as metropolitan-area and wide-area networks (MANs and WANs). Thanks to this continued evolution, Ethernet became the dominant data-link technology in use today.

Modern Ethernet does away with the performance, scalability, and reliability limitations of a shared-medium protocol in favour of full-duplex point-to-point links and a switched architecture. The most commonly used transmission media are now inexpensive twisted copper pair and high-performance optical fibres. Currently commercially available link speeds range from 1 to 100 Gb/s over transmission distances ranging from a few centimetres on a circuit board to tens of kilometres on long-distance fibres.

In practically all deployments, Ethernet is used in conjunction with IP [37] on the network layer. In principle, Ethernet, as a data-link-layer technology, should only be concerned with moving packets from one end of a link to the other. On the other hand, IP, as a network-layer protocol, is responsible for moving packets from their source to their destination, if necessary routing them via intermediate nodes. In practice, this distinction is not so sharp: as explained in the following sections, the IEEE 802 family of

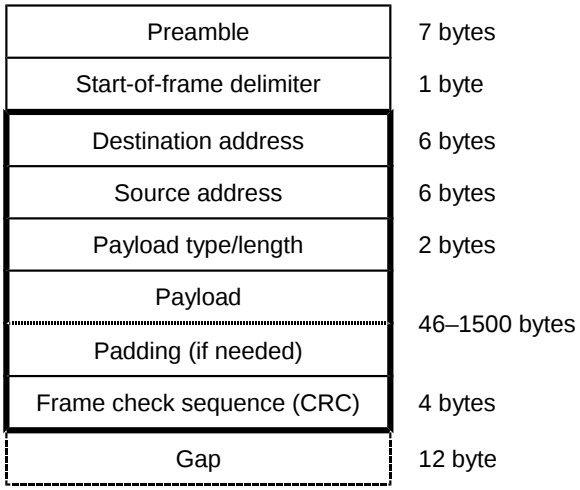


Figure 2.1 Ethernet physical layer data format: preamble, start-of-frame delimiter, frame, and inter-frame gap.

standards also defines a device known as a network bridge, which, fundamentally, can be seen as a simple router for data-link-layer packets.

Frames and datagrams

The basic unit of data on an Ethernet link is called a *frame* and is represented in figure 2.1. A frame consists of a 14-byte header, the payload, and a 4-byte trailer. The header contains the destination address, the source address, and, depending on the frame variant, an indication of either the network-layer protocol of the payload or its size. An Ethernet address is a 6-bytes-long globally-unique identifier assigned to each network interface by its manufacturer. The trailer contains a cyclic redundancy check (CRC) sequence for the header and payload. The maximum payload size, called maximum transfer unit (MTU), is 1500 bytes.

The minimum frame size is 64 bytes; if the payload is too small it is padded with zeros so that the minimum frame size is reached. At the physical layer, an Ethernet frame is preceded by two fixed bit sequences: a 7-bytes sequence of alternating 1 and 0 bits, called preamble, and a 1-byte start-of-frame delimiter. Each transmission is followed by a mandatory gap of 12 bytes. Therefore, the total overhead is 38 bytes per frame.

The frame's payload normally consists of an IP data unit, called *datagram*, represented in figure 2.2. Normally, the length of an IP datagram header is 20 bytes, unless optional

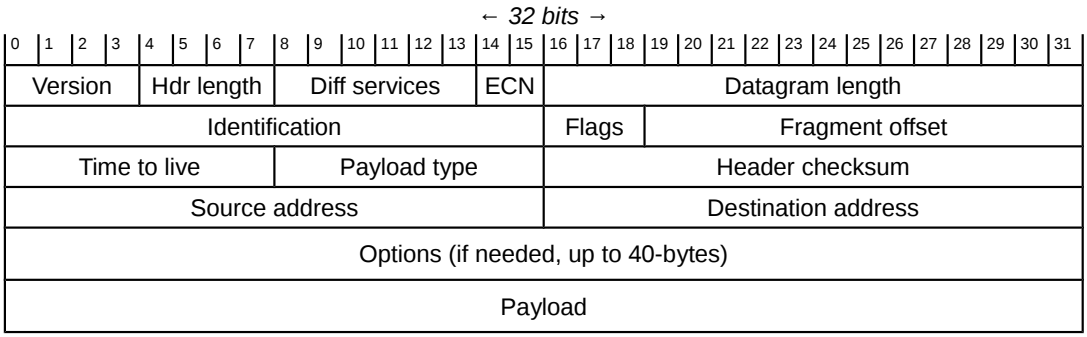


Figure 2.2 Structure of an IPv4 datagram.

fields are present, which is rare. The header begins with the version number field. The differences between the two IP versions currently in use (IPv4 and IPv6) are not relevant in the context of data-acquisition systems, so, for simplicity, only IPv4 is described here. Following the version field, are the header length, the so-called "differentiated services" and "explicit congestion notification" (ECN) fields, and the total length of the datagram. The differentiated services field can be used to specify the datagram's classification for quality-of-service purposes. The ECN [57] field is optionally used for marking a datagram that passed through an intermediate node with a high amount of internally queued traffic. When the packet is received by the destination, an ECN-compatible transport-layer protocol (such as TCP) can then use this indication to ask the source to reduce its sending rate. An IP datagram cannot exceed the MTU of the data-link that transports it. Since different links can have different MTUs, IP routers can fragment a datagram into smaller datagrams. An endpoint receiving these datagrams must be able to re-assemble the original datagram. The identification, flags, and fragment offset fields are used for this purpose. This feature is not particularly relevant for IP over Ethernet: the MTU is fixed to 1500 bytes. The time-to-live field is used to prevent infinite routing loops. The remaining fields are rather self-explanatory. The header includes: an indication of transport-layer protocol of the payload, a checksum of itself, and the source and destination addresses. IPv4 addresses are 4-bytes long and are assigned to network interfaces by the network administrator. As already mentioned, the optional fields are rarely included, so they are not described here (see [28] for a comprehensive description). The normal IP header adds another 20 bytes of overhead, which, together with Ethernet's, adds up to a total of 58 bytes.

Bridging

Ethernet switches are a specific implementation of the network bridges defined by IEEE standard number 802.1Q [35] (formerly 802.1D). Bridges maintain a forwarding table, i.e., a mapping associating a device's address to the port the device is connected to (either directly or through other bridges). The forwarding table is populated by address learning: when a bridge receives a frame on a certain port, it reads the source address of the frame. In this way, the bridge learns that the device with that address is connected to that port. The forwarding operation relies on this information: when a bridge receives a frame, it reads its destination address and looks up the corresponding port in the forwarding table. If a match is found, the frame is sent out on the matching port. Naturally, it is possible that the forwarding table does not contain an entry for the destination address yet. In that case, the bridge sends out a copy of the frame on all ports, except for the port it came from. This operation, known as "flooding", ensures that the frame reaches its destination even when the forwarding tables are incomplete.

The forwarding algorithm described above, achieves two important results. It is transparent, i.e., it does not require the source or destination nodes to be aware of the existence of bridges in the network. It is also distributed, i.e., it does not require a centralised entity with knowledge of the network topology in order to work. However, due to the flooding mechanism, it imposes a constraint on the network topology: no loops are allowed, i.e., the bridges must be organised in a tree structure, so that a single path exists between any two nodes in the network. If a network contains a loop, flooded frames will circulate endlessly around it, eventually saturating the links. To prevent this situation, the standard includes a protocol with the purpose of automatically suppressing loops. Under this Spanning Tree Protocol (STP), the bridges exchange informations about the links and paths interconnecting them. Based on this data, a root bridge is elected and non-root bridges select a single path to communicate with it. Bridge ports corresponding to alternative paths to the root are blocked, thus breaking all possible forwarding loops. If one of the selected paths becomes unavailable, for example because of link failure, the ports corresponding to an alternative path are re-activated.

Bridges are appealing because of their simplicity and their self-configuring nature. However, this simplicity comes at the expense of both scalability an flexibility. The scalability issue stems from the fact that, for correct network operation, a bridge's forwarding table must contain, after address learning, an entry for each node in the bridged network. Since the forwarding table is consulted every time a frame arrives at the bridge, fast lookups are crucial. For this reason, forwarding tables are usually implemented with ex-

pensive specialised memory. Therefore, a trade-off exists between the bridges' cost and the maximum number of nodes supported by the network. With topology constrained to a tree, bridged networks are not particularly flexible. STP ensures that the network can continue to operate correctly even in the presence of loops and makes designing redundant networks topologies possible. However, the resulting "logical" tree topology might lead to inefficient network usage. For example, a direct path between two bridges might be blocked in favour of the path passing through the root bridge and, naturally, alternative paths between two bridges cannot be used at the same time because that would create a loop [54]. Both these issues are solved by the more sophisticated IP routing.

Routing

IP overcomes the scalability limitations of bridging by using topological addressing. Roughly speaking, this means that network administrators configure nodes that are "near" each other in the network topology with IP addresses that begin with the same prefix. In this way, it is possible to refer to groups of nodes or to entire networks just by specifying their common address prefix, rather than by enumerating the addresses of all the nodes. Groups of nodes sharing the same address prefix are called "subnets".

In IP routing [58], decisions are taken by each node independently. Before sending a datagram, a node consults its configured routing table. Each entry in the routing table associates a subnet prefix to the address of one or more intermediate nodes that can deliver the datagram to destinations in that subnet. The intermediate nodes (called routers or IP switches) will consult their own routing table to choose the next node, and so on. The process continues until the datagram arrives at a router in the same subnet as the destination, which can then directly deliver the datagram via the appropriate data link. Thanks to the topological addressing scheme, routing tables do not need to contain the address of every host in the network: it is sufficient that they list which router can be used to reach which subnets. Nothing guarantees that misconfigured routing tables will not result in a routing loop. To prevent network saturation caused by routing loops, the time-to-live counter in the IP datagram header is initialised to a positive value by the source node. Every router the packet goes through decrements the counter. When the counter reaches zero, the packet is dropped instead of being forwarded.

In general, routing tables are not self-configuring. The network administrators can either populate them manually, or they can rely on routers communicating with each

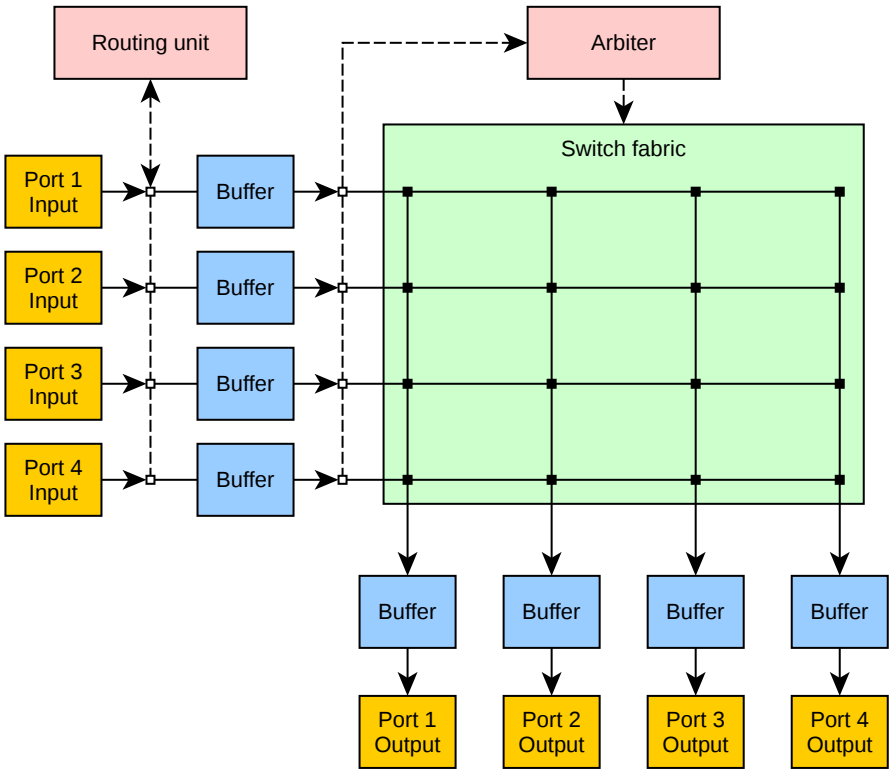


Figure 2.3 Logical building blocks of a network switch.

other using a routing protocol. With a routing protocol, routers learn about the topology of the whole network by exchanging information about the subnets they are part of. The topology information is used, together with link cost parameters set either by the protocol itself or by the network administrators, to choose the best paths to reach each subnet. A specific routing protocol might require a specific network topology to work, but this is not true in general for IP networks: the administrators are free to choose the topology they deem more suitable. Finally, a node's routing table can indicate more than one router for the same subnet prefix. The node can then choose among multiple paths to reach the same destination, enabling redundancy and load-balancing.

Switches

The logical architecture of network switches, including Ethernet bridges and IP routers, is shown in figure 2.3.

Switches' network interfaces are commonly referred to as ports. Ports are logically divided into an input and output port. When a packet arrives through a link at one of the switch's input ports, its headers are sent to the routing unit, which determines the packet's output port(s). The full packet is stored in the port's input buffer. The switch fabric interconnects all input ports to all output ports. Packets cannot always cross the fabric instantly: for example, with the very common *crossbar* fabrics, two packets with the same output port cannot traverse the fabric at the same time. Therefore, an arbitration unit decides when and which input ports are allowed to access the fabric. Packets crossing the fabric are stored in their output port's buffer until they can be sent on the port's link.

Input and output buffers serve two different purposes: input buffers store packets until the fabric is available to forward them, output buffers allow packets to cross the fabric even if their destination port is busy sending out other packets. Switch implementations differ in how they allocate space in buffers. The simplest allocation scheme is *input queuing*. As the name implies, only input buffers are used in this scheme: packets are queued at each input port, in order of arrival, until their output ports are free to accept them. This scheme is cheap and easy to implement, but it suffers from a huge drawback known as head-of-line blocking: if the packet in front of the queue is destined to a busy output port, packets behind it in the queue will have to wait until it can be forwarded, even if their output ports are free. For this reason it is very rarely used.

On the other side of the spectrum is the *output queuing* scheme, which avoids head-of-line blocking completely. In this scheme, input buffers are not present. Therefore, the switch fabric is required to be fast enough to handle all possible combinations of packets as soon as they arrive at the switch. This requirement is rather burdensome: for example it can be shown that an output-queued switch with N ports and a crossbar fabric would need to operate N times faster than the port speed. The continuous progression of link speeds made this architecture less and less feasible.

Most switch implementations use more sophisticated queuing schemes, combined with a suitable arbitration algorithm, to achieve performance close to or equivalent to output-queued switches while avoiding their burdensome fabric speed requirement. A common trait of these schemes is the use of *virtual output queues* (VOQs): each input port maintains a separate queue for each output port, instead of keeping all the arriving packets in a single queue. For each input port, the arbitration unit can then choose which virtual output queue is served first. Thus, a well-designed arbitration algorithm can avoid head-of-line blocking. See [29] for an example of such an algorithm and for

references to others.

Service model and flow control

Both Ethernet and IP offer best-effort (i.e., unreliable) packet delivery: the network does not guarantee that all packets will reach their destination. Furthermore, there are no ordering guarantees: packets might reach their destination in a different sequence than when they were sent. This is common in topologies where multiple paths to the same destination are available. In case of problems packets are simply discarded. Besides the obvious case in which a route to the destination cannot be found (for example because of a severed network link), there are two other scenarios in which packets are discarded: transmission errors and network congestion.

Ethernet senders compute the CRC checksum of each frame and store it in the frame's trailer. Ethernet network interfaces can detect transmission errors by verifying that received frames still match their checksums. Similarly, IP datagrams also contain a checksum which however only protects the integrity of the header (but not the payload). If either one of these checks fails, the packet is discarded by the receiving interface. Discards are silent: no notification whatsoever is sent to the packet's source.

Network congestion occurs when an intermediate node, i.e., a switch, receives more packets than it can send out. This can happen with a switch that has two ports of differing bandwidths: a flow of packets arriving at full speed on the faster port cannot be sent out on the slower port at the same rate. Another source of congestion are multiple flows of packets arriving on different input ports to be sent out on the same output port. In case of congestion, packets will accumulate in the switch's buffers waiting to be sent out, increasing the latency. If the congestion persists, the buffer space is exhausted and additional packets are discarded. Source nodes detecting increased latency or discarded packets might use this information to infer that a network path is congested and try to remedy the problem by lowering the throughput of the data flow(s).

As an alternative to discarding packets, the Ethernet standard allows a congested switch to send a special "pause frame" to request that the node at the other end of a link stop sending data for a period of time specified in the frame. This mechanism suffers from a major drawback: when a link is paused, all data flows on that link are stopped, even if they were not contributing to the congestion. As a consequence, this feature is very rarely used.

2.2.2 The Transmission Control Protocol (TCP)

Given the best-effort nature of IP networks, an application that requires the reliable delivery of a stream of data to its destination must, at least:

- detect packet losses and retry sending the discarded packets;
- implement a mechanism that permits reconstructing the original order of packets in case of out-of-order delivery.

TCP [63] offers a standardised way to do this: it provides applications with a connection-oriented, reliable, byte stream service on top of the unreliable IP services. To maximise reliability, TCP implements the following additional features:

- Multiplexing: TCP enables different applications on the same host to share the same IP network.
- Protection against data corruption (i.e., checksums).
- Flow control: TCP tries to avoid sending more data than the receiver is prepared to handle.
- Congestion control: as explained in section 2.2.1, network congestion can lead to packet loss. Therefore, TCP tries to detect and avoid congestion.

Reliability is implemented using acknowledgements (ACKs) and retransmissions: in basic terms, the receiver confirms the arrival of packets by sending an explicit confirmation message (the ACK) to the sender. If the sender does not receive an ACK within a certain timeout, it assumes that either the packet or the ACK were discarded and sends another copy of the packet.

Segments

TCP receives a stream of data from the application and subdivides it in units, called segments, that are suitable for encapsulation in an IP datagram. Segments, represented in figure 2.4, consist of a header containing information needed for TCP to work and a payload containing the application data. A TCP sender tags every unique segment that it transmits with a *sequence number* that identifies its position in the data stream. The receiver can use this sequence number to detect duplicate segments or to re-order segments that were received out-of-order.

Normally, the length of a TCP segment header is 20 bytes, unless optional fields are present. The header begins with the source port and destination port fields. These two

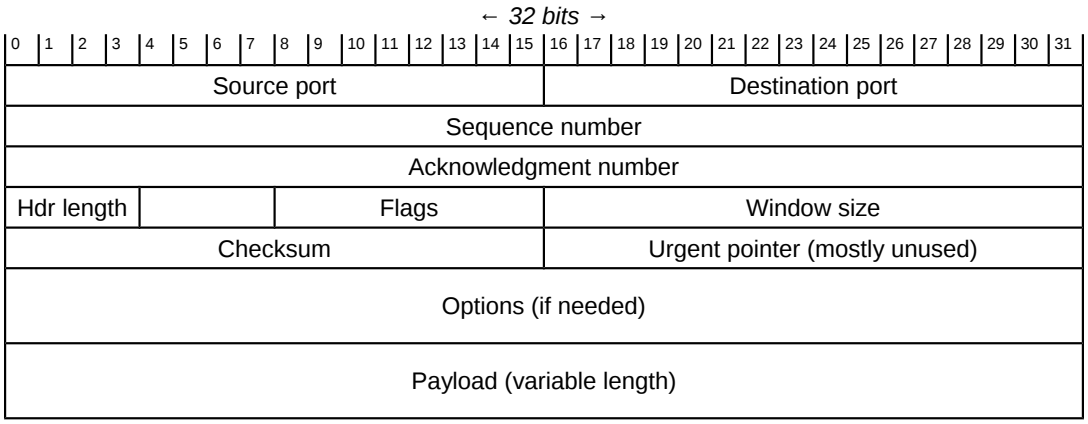


Figure 2.4 Structure of a TCP segment.

16-bit integers are used to implement multiplexing: a TCP port number uniquely identifies an application within an end-node. As explained above, the sequence number field contains the position in the data stream from the sender to the receiver of the first byte in the segment's payload. The acknowledgement number field contains the next sequence number that the receiver expects. In other words, this is the position in the data stream of the last correctly received byte plus 1. The header length field is self-explanatory. The flags field contains a series of boolean values. The first two values are used to implement the explicit congestion notification (ECN) mechanism already mentioned in section 2.2.1. The rest are used during connection establishment and termination. The window size field is used to implement flow control: it specifies the number of bytes that the receiver can accept. The checksum field is peculiar, in that it covers not only the TCP segment, but also some fields of the IP header. The urgent pointer field was designed as a way to send "specially" marked data. It is almost never used in practice.

As mentioned in section 2.2.1, Ethernet specifies a maximum payload (MTU) of 1500 bytes. Of those, assuming that no optional header fields are present, 20 are occupied by the IP header and 20 by the TCP header. Therefore the maximum TCP payload size, called *maximum segment size* (MSS) is 1460 bytes of application data. Adding the TCP header size to the overhead of Ethernet and IP, the total protocol overhead is 78 bytes, i.e. ~5.3% of the MSS.

Retransmissions

Modern TCP implementations have both an active and a passive packet-loss detection mechanism [13].

The basic idea behind the active mechanism, called *fast retransmit*, is outlined here. If a receiver observes a "hole" in the flow of segments that it is receiving, there are two possibilities:

- the network did not preserve the order of the segments but the missing ones will eventually arrive, or
- the missing segments were discarded.

The more segments after the hole are correctly received, the likelier it is that the missing segments were really discarded and not simply arriving out-of-order. The fast retransmit mechanism requires that a receiver generate an ACK of the last correctly received segment (i.e., the last fragment received before the hole) for every out-of-order segment received. This duplicated ACKs can be used by the sender to infer that there are segments that must be retransmitted: if the sender observes more than a certain threshold of duplicated ACKs it immediately start resending the segments that it presumes were discarded.

Unfortunately, the fast retransmit mechanism only works if the discarded segments are followed by some correctly delivered segments. All other scenarios are covered by the passive mechanism, which is based on retransmission time-outs (RTO) on the sender side. If the sender does not receive an ACK before the RTO expires, it starts resending the unacknowledged segment(s). To prevent spurious retransmissions, the RTO should always be higher than the round-trip time of the TCP connection[1]. To do this, TCP calculates the RTO as follows:

$$RTO = SRTT + 4 \times RTTVAR$$

where $SRTT$ and $RTTVAR$ are TCP's smoothed estimations of the round-trip time and its variation (see [60] for more details). However, to protect against spurious retransmissions caused by interference with other TCP features such as delayed acknowledgements, the RTO has a fixed minimum, specified as 1 s in [60], with lower values used in actual implementations (200 ms in Linux, 30 ms in FreeBSD).

[1] In TCP, the round-trip time is defined as the total interval between the time a segment is sent and its corresponding acknowledgement is received.

Congestion control

To avoid congesting the network, TCP limits the number of unacknowledged segments in transit through the network. This limit is called *congestion window* (CWND). The effect of the congestion control mechanism is to limit the throughput to:

$$Throughput = \frac{CWND}{RTT}$$

where $CWND$ is the size of the congestion window in bytes and RTT is the round-trip time. Therefore, with too small a window TCP will not be able to fully utilise the available network bandwidth. On the other hand, with too large a window, TCP will overwhelm the network, causing congestion. If the congestion gets so high that segments are discarded, they will have to be retransmitted, resulting in lower throughput and wasted network resources.

TCP tries to adapt the congestion window size to the network's conditions. Generally speaking, connections start with a small congestion window, which is progressively increased as long as TCP does not observe some evidence of network congestion. When congestion is detected, the window size is reduced to try to alleviate the congestion. The most common TCP variants assume that the network is congested when they detect that segments are discarded: the sender increases the size of its congestion window as long as it receives ACKs for sent segments, and reduces the window size every time it has to retransmit a segment with the mechanisms described in the previous section. This mechanism is used for example by the "NewReno" [30] variant, used by default by the Windows and BSD operating systems, and the "CUBIC" [32, 31] variant, used by default by the Linux operating system. Some very rarely used variants (e.g. TCP "Vegas" [14] and TCP "FAST" [66]) try to be more proactive and, instead of waiting for segments to be discarded, they interpret an increase in the round-trip-time as evidence of congestion. The two approaches can be combined: this is done for example by the "Compound" variant [62]. A third class of TCP variants requires that routers support the explicit congestion notification mechanism described in section 2.2.1. With ECN, routers are asked to evaluate if they are experiencing congestion, i.e., if some of their buffers are too full (this estimation can be as simple as a buffer occupancy threshold). The router then sets a flag in the header of the IP datagrams that are contributing to the congestion (e.g., the datagrams at the tail of the queues). When the destination receives a datagram marked in this way, it informs the sender by setting a flag in the TCP header of the corresponding ACK. The sender can then use this information to reduce its congestion

window.

TCP variants also differ in how they calculate the amount of bytes by which the congestion window is increased in the absence of congestion or decreased in response to congestion. Generally speaking, except for the variants using round-trip-time as a congestion signal, an additive-increase/multiplicative-decrease (AIMD) algorithm is used [38]. The congestion window is calculated starting from its previous value:

$$CWND(t) = \begin{cases} CWND(t-1) + \alpha(t) & \text{if no congestion is detected} \\ CWND(t-1) \times \beta(t) & \text{if congestion is detected} \end{cases}$$

where the parameter α ($\alpha \geq 0$) is the addictive increase parameter and β ($0 \leq \beta \leq 1$) is the multiplicative decrease parameter. Each TCP variant defines a different way to calculate these parameters with the goal of: fully utilising the available bandwidth, fairly sharing the network with other TCP flows, and quickly adapting to changes in the network conditions.

2.3 The *incast* pathology

The combination of a many-to-one traffic pattern and TCP over a best-effort network is subject to a well-known pathology called *incast*: first observed in data-centre storage networks, it occurs "when a client simultaneously receives a short burst of data from multiple sources, overloading the switch buffers associated with its network link such that all original packets from some sources are dropped" [53].

When many sources send packets at the same time to the same destination, it is reasonable to assume that eventually the packets from the various sources will end up in the same queue in a switch's buffer. If the collective size of the data sent from all sources is larger than the available buffer space, some packets will obviously be discarded. This is not a problem per se: the TCP senders that realise that their packets were discarded will reduce their congestion window to avoid overflowing the buffers; eventually the situation should stabilise. However, this works well only if the data flows are relatively long-lived and the number of sources is not too high.

With many short bursts, consisting of just a few packets, it is possible that all of the packets sent by certain senders are discarded, while all of the packets sent by other senders are delivered. If this happens, the unaffected senders will certainly not reduce their congestion window, so the problem will repeat itself. Worse, since all of

their packets are discarded, the affected senders will not trigger the fast retransmission mechanism described in section 2.2.2. The discarded packets will eventually be re-sent after a retransmission timeout (RTO). In many cases, waiting for a RTO dramatically lowers the throughput of the affected TCP flows. The problem is particularly evident in networks with low round-trip times: modern data-centre networks, for example, have sub-millisecond round-trip times. The minimum TCP RTO is normally orders of magnitude larger than that (see section 2.2.2). A flow that is subjected to incast therefore experiences what is widely described as "throughput collapse". Note that, while this description focused on the inability of TCP to react sensibly to incast, the same pathology essentially affects all protocols that use timeouts to retransmit packets discarded by a best-effort network.

In many cases, data-acquisition networks suffer from the incast pathology, due to the typical bursty, many-to-one traffic pattern described in section 2.1.2.

2.4 Related works: incast avoidance and mitigation

The incast pathology is well-known in literature. While no studies have focused on data-acquisition networks, many solutions have been proposed for the general case of low-latency high-bandwidth networks.

Data center TCP (DCTCP) [4, 3] leverages the explicit congestion notification (ECN) mechanism described in section 2.2.1 and section 2.2.2 to measure the amount of congestion in the network. It adjusts the TCP congestion window based on that measurement, rather than relying on packet drops as a congestion signal. With this faster, finer-grained reaction to congestion, it is able to outperform standard TCP in incast scenarios. However, it can only do so if the number of concurrent senders is not too high (the threshold depends on the available switch buffer space). Unlike many other proposals, DCTCP is being developed from a research work into a well-defined standard [12], and a production-quality implementation of DCTCP is available in recent Linux kernels. Besides the limited scalability to many concurrent senders, DCTCP's main drawback is its reliance on the ECN mechanism, which must be reliably supported by all switches in the network, limiting the purchasing choices and raising the total price of the data-acquisition system.

TIMELY [45] and DX [42] are two alternative congestion control protocols that aim at minimising queuing in the switches. They rely on increases in round-trip-time to detect

queue growth and react accordingly. The round-trip-time measurement must be very precise to detect queue length changes in high-bandwidth networks: a 100 kB queue on a 10 Gb/s link adds a delay of just 8 µs. Round-trip-time measurements of such precision are not possible in the operating system: the noise added by the time it takes to transfer a packet from the network interface card to the operating system is too high. Instead these proposals rely on special network interface cards that can measure the round-trip time directly on packet arrival. Naturally, the cost of these cards significantly drives up the total cost of the system. Moreover, references [45] and [42] only report on the performance of these schemes in incast scenarios with up to 40 senders. It is safe to assume that, like DCTCP, higher sender counts would not be handled as effectively.

In [46] the authors suggest purposely lowering the TCP receive window on the destinations of incast flows. This is an easy solution, but it is only effective in scenarios with a small number of senders: the sum of the sizes of the receive windows on the receiver should be lower than the smallest buffer in the network path from the senders. With a large number of senders, this might require receive windows so small that the concerned TCP connections would be unable to exploit the available network bandwidth during non-incast operation.

ICTCP [67] turns the approach suggested in [46] into a proposed modification of TCP receivers. The receive windows of TCP connections are jointly adjusted to control throughput on incast congestion. ICTCP bases that adjustment on an estimate of the available input bandwidth of the receiving node. As such, it can only prevent incast if the incast congestion occurs at the last switch before the destination node. This is a frequent occurrence, but not necessarily the sole incast scenario in data-acquisition networks. At large scale, these networks usually have more than one aggregation stage (see section 4.1 for the ATLAS case). Only the last aggregation step would be protected with ICTCP.

IEEE 802.1Q [35] congestion notification (QCN) is an optional part of the IEEE standard that defines network bridges (see section 2.2.1). It defines a congestion control mechanism that can be implemented by Ethernet network interface cards and switches: switches identify the flows contributing to congestion and ask the relevant network interface cards to limit the sending rate of those flows. By design, it is targeted towards long-lived traffic flows. Simulation results [26] reinforce this consideration and show that in many cases QCN can actually worsen the network performance in incast scenarios. Moreover, it requires specific hardware support in both switches and network cards.

As section 6.5 will show, switches with large buffers can prevent incast altogether. Ob-

viously, the necessary buffer space depends on the specific data-acquisition workload. Commercially available switches with buffers larger than a few hundreds of kilobytes per port are few and obviously more expensive than their shallow-buffered counterparts. Software-defined switches based on commodity general-purpose PCs are a possible alternative. The proposed system in [40] uses the main system memory as a very large packet buffer. The scalability of this approach is however limited by the bandwidth between the network interface cards and the main memory.

Due to the impracticality of these general solutions, this thesis focuses instead on application-specific options, that can be deployed on existing or future systems without major changes to their hardware or base operating system software.

The ATLAS experiment at the Large Hadron Collider

3

Throughout this thesis, the data-acquisition system of the ATLAS experiment will serve as a case study. ATLAS is a particle physics experiment observing collisions delivered by the LHC proton accelerator at CERN. Both ATLAS and the LHC are among the largest, most complex experimental facilities ever built. This chapter contains some background information on the accelerator and on the experiment. The building blocks of the ATLAS data-acquisition system are described in detail. Particular focus is given to the messaging subsystem and to the data-collection software which were both developed as a preliminary part of this work.

3.1 High-energy physics

Particle physics is the branch of physics that studies the elementary constituents of matter and radiation. Colliding particles at higher and higher energies has proven to be a fruitful avenue to expand our knowledge of nature. Hence, particle physics is often referred to as high-energy physics.

Results from high-energy physics experiments have led to the formulation of the Standard Model of particle physics, which has been strikingly successful in predicting and describing the currently known fundamental particles and the interactions between them. It covers three of the four known fundamental interactions: *electromagnetic force*, *weak nuclear force*, *strong nuclear force*.

Nevertheless, the Standard Model is necessarily an incomplete theory as it does neither account for the fourth known force, gravity, nor provide an explanation to cosmological

problems like the apparent existence of dark matter and the observed matter-antimatter asymmetry. New phenomena, beyond those described by the Standard Model, could be discovered by further increasing the energy of high-energy physics experiments.

3.2 The Large Hadron Collider

The Large Hadron Collider (LHC) is a particle accelerator and collider at the European Organization for Nuclear Research (CERN) near Geneva, Switzerland. The LHC is designed to accelerate beams of either protons or lead nuclei and bring them to collision at the centres of four particle detectors with a centre-of-mass energy of up to 14 TeV[1]. Protons (and the neutrons in the lead nuclei) belong to a class of particles known as *hadrons*: composite particles made of *quarks* held together by the strong nuclear force. Hence, the name Large Hadron Collider.

The LHC physics programme is broad and diverse, ranging from more precise measurements of the Standard Model fundamental constants to the search for new physics phenomena. One of its main goals is finding evidence for one particularly important feature of the Standard Model which was not previously verified: the so-called *electroweak symmetry breaking* mechanism. Without this mechanism the Standard Model is incomplete: it cannot consistently account for the mass of the elementary particles. The verification of the electroweak symmetry breaking mechanism hinges on the existence of a particular particle called the *Higgs boson*. A particle that is consistent with the Standard Model Higgs boson was indeed observed in 2012 [9, 20]. The discovery crowned an enormous effort that spanned almost 30 years from the research and development in the early 1990s to the initial operation in 2008.

The LHC is thoroughly described in [27]. This section attempts to provide the reader with some insight into the design and operation of the machine.

3.2.1 Construction

The LHC is a circular accelerator. It uses powerful electromagnetic fields to accelerate and steer two counter-rotating beams of protons. The circular layout enables continuous acceleration, as the particles can be indefinitely recirculated through the accelerat-

[1] The electronvolt (eV) is a unit of energy commonly used in particle physics. It corresponds to the energy gained by an electron moving across a potential difference of 1 volt. $1\,\text{eV} \approx 1.602 \times 10^{-19}\,\text{J}$. Due to the mass–energy equivalence ($E = mc^2$), the eV can also be used as a unit of mass. $1\,\text{eV} \approx 1.782 \times 10^{-36}\,\text{kg}$.

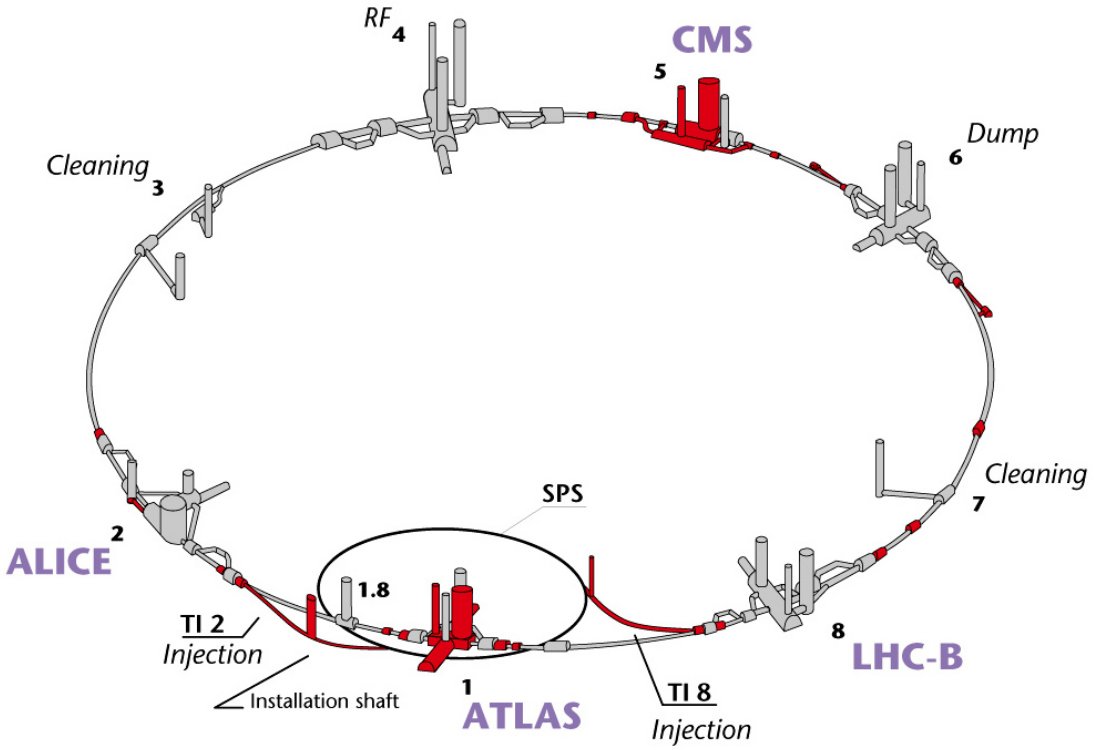


Figure 3.1 Layout of the LHC tunnel [18]. The red parts are new underground buildings built specifically for LHC. The grey parts represent existing LEP infrastructure.

ing sections, thus enabling to reach very high energies in a relatively compact design. It is currently the largest and most powerful particle collider in the world. The high-energy accelerators that preceded the LHC, the Large Electron–Positron collider (LEP) and the Tevatron collider were both circular accelerators. The LEP collided electrons with positrons at a centre-of-mass energy of up to 209 GeV, while the Tevatron collided protons with antiprotons at a centre-of-mass energy of up to 1.96 TeV.

The LHC is housed in the 26.7 km long tunnel represented in figure 3.1. The tunnel straddles the French-Swiss border near Geneva, lying between 45 m and 170 m underground. It has four *interaction points* where the two counter-rotating proton beams are guided to collide. Four different physics experiments are built around the four interaction points, reconstructing the collision events to perform detailed studies of known physics processes and search for evidence of new physics. Two of them, ATLAS [10] and CMS [21], are general purpose detectors; ALICE [2] is dedicated to the study of a particular state of matter, the *quark-gluon plasma*, which is postulated to have existed during

the early universe; LHCb [43] is targeted towards studying the decays of a specific class of particles, the *B hadrons*, in order to better understand the asymmetry between matter and antimatter in the universe.

3.2.2 Physics performance

Each of the two proton beams of the LHC are subdivided in groups of particles called *bunches*. When two bunches are driven to collision, the protons in the bunches will scatter due to the interactions with the other protons. Most of the protons will experience *elastic* scattering, i.e. their kinetic energy will not change due to the interaction, so the centre-of-mass energy of the two colliding protons is zero. A small fraction of them, however, will experience *inelastic* scattering, losing some or all of their energy due to the interaction. The experiments around the interaction points observe the results of these inelastic collisions.

The rate of interactions generated in a collider via a specific physical process is given by:

$$R = \mathcal{L}\sigma$$

σ is the so-called *cross section*: a quantity that is characteristic of the process under study. It expresses the probability that two particles (two protons in the LHC) interact in a specific way. Among other parameters, σ is a function of the centre-of-mass energy of the particle interaction.

$\mathcal{L}$ is the *luminosity*: a quantity that depends only on the physical parameters of the beams in the collider. Among many factors, the luminosity of a collider is proportional to the frequency f at which the bunches of particles collide with each other (*bunch-crossing frequency*) and to the average number μ of inelastic particle collisions per bunch crossing:

$$\mathcal{L} \propto \mu f$$

The luminosity also depends on the shape of the particle bunches and on the geometry of the collisions.

In order to pursue the goals of the LHC physics programme, extremely rare processes need to be studied. Figure 3.2 helps quantify what "extremely rare processes" means, showing the cross sections of various processes at a proton collider. In that figure, σ_{tot} refers to the *total proton-proton inelastic cross section*, which expresses the probability that

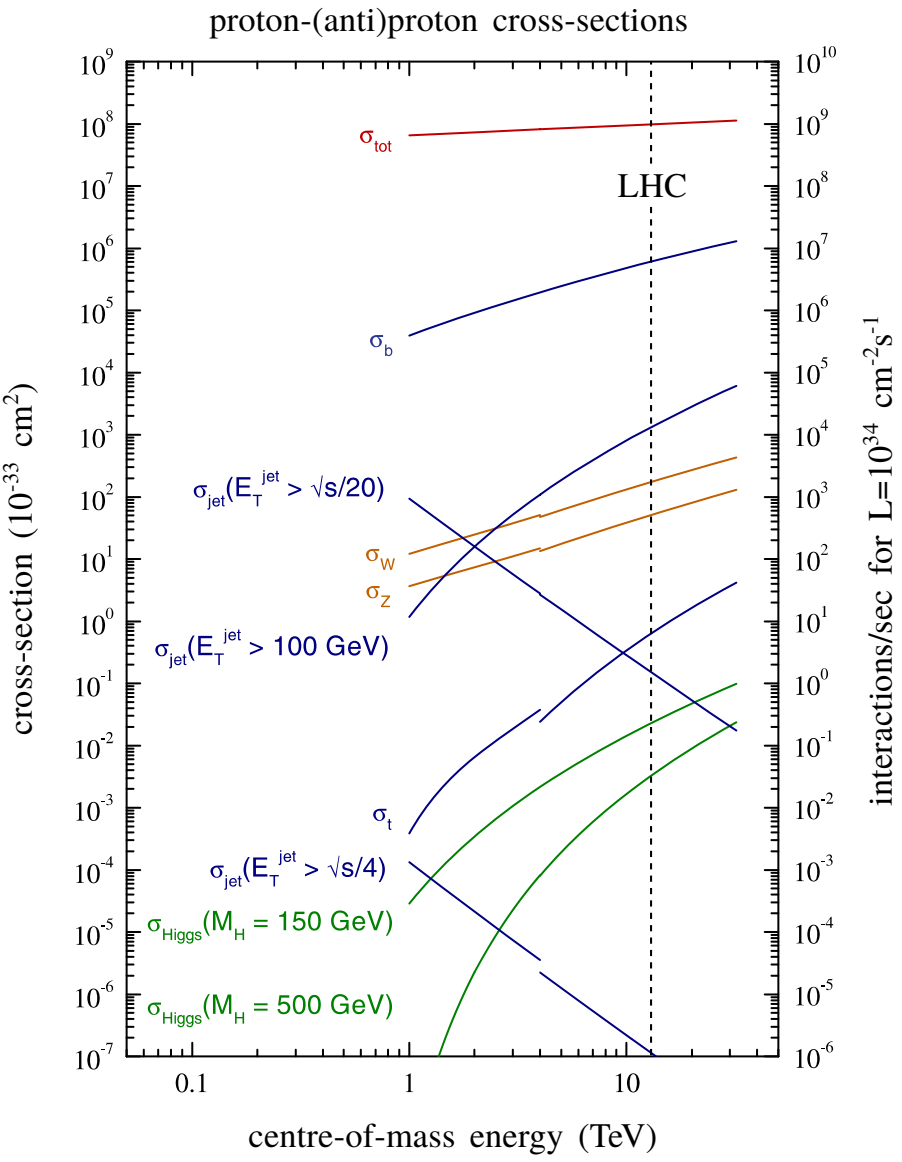


Figure 3.2 Various Standard Model proton-proton cross-sections (and expected interaction rates at the LHC design luminosity) as a function of the centre-of-mass energies. Adapted from [17].

two protons will interact in any kind of inelastic scattering process. At the LHC centre-of-mass energy (14 Tev), the cross section σ_{Higgs} for a Standard Model Higgs boson (with mass $M_H = 150\,\text{GeV}$) is 10 orders of magnitude smaller than σ_{tot}. Thus, with the luminosity being constant, on average, the rate of production of the highly sought-after Higgs boson will be $1/10^{10}$ of the total proton-proton collision rate. Similar considerations apply to most of the physical processes that are considered interesting for the LHC research programme.

The goal of the LHC project is producing a high enough luminosity to make this research programme possible. It is designed to operate at bunch crossing frequency of up to $f = 40\,\text{MHz}$, with an average pile-up of 25 high-energy particle collisions per bunch crossing, i.e. up to 10^9 high-energy proton-proton collisions per second.

3.3 The ATLAS Experiment

ATLAS (A Toroidal LHC ApparatuS) [10] is a general-purpose particle detector designed with the objective of undertaking a wide range of different physics analyses. These include high-precision measurements of Standard Model parameters, searches for the Standard Model Higgs boson and for so-called *supersymmetric* particles, and probes for *exotic* new physics such as *heavy gauge bosons*, *technicolor* particles, or *extra dimensions*.

In order to perform all of these measurements, the detector must be able to identify the full range of particles which may be produced by these processes and measure their momentum and energy. This is complicated by the difficult experimental conditions at the LHC: as mentioned in section 3.2.2, the formidable luminosity comes at the price of high pile-up, with every bunch crossing producing an average of 25 superimposed inelastic interactions in design conditions. Therefore, high spatial resolution is needed throughout the detector to properly distinguish particles from overlapping interactions. Maximum coverage must be achieved as well, minimising the amount of particles that escape the experiment undetected.

The high collision rate also has the unfortunate consequence of creating a difficult experimental environment in terms of radiation dose, rendering radiation-hardness an unavoidable requirement for the detector. A certain amount of built-in redundancy is also necessary to ensure that data-taking is not impacted significantly by failures in some parts of the detector.

3.3.1 Detector layout

Figure 3.3 shows a view of the detector with part of it removed to expose components otherwise hidden. The experiment is organised in a cylindrical shape around the beam line. The central part of the cylinder is referred to as the *barrel*, while the terminal parts are known as *endcaps*. The two proton beams enter the experiment at the centres of the endcaps and collide in the middle of the barrel. The experiment is made up of a combination of several layers utilising different particle detection technologies. Each is referred to as sub-detector.

Going from the interaction point to the outside of the detector the sub-detectors first encountered are those forming the *inner detector*. These are enclosed by a superconducting solenoid magnet. The magnetic field bends the trajectories of the charged particles produced by the collisions, with the curvature radius and direction depending on the charge and velocity of the particle. Thus reconstructing the trajectory of the charged particles enables the determination of their charge and momentum.

Outside of the solenoid one finds the two *calorimeters*, which, in high-energy physics parlance, are detectors that measure the energy of particles. This measure is normally destructive: the particles deposit all their energy in the calorimeter and are absorbed.

A specific kind of particles, the muons, are able to escape the calorimeters without losing much energy. Another tracking system, the *muon spectrometer*, is used to supplement the inner detector in measuring their momentum. The magnet system of the spectrometer consists of 8 large superconducting magnet coils in the barrel region that generate a toroidal magnetic field in air, complemented by two smaller end-caps consisting of 8 coils each in the forward region. Layers of different particle detectors register the passage of muons.

Locations in the detector are defined by a set of coordinates with origin at the beam interaction point. The proton beams are defined as travelling in the z-direction and the x-y plane transverse to that. The x-axis points towards the centre of the LHC ring and the y-axis points upwards, away from the centre of the earth. The distance from the z-axis is called R. The angular coordinates are defined as follows. The azimuthal angle, ϕ, is measured in the x-y plane from the y-axis, and the polar angle, θ, is measured from the z-axis. For a particle of momentum $\vec{p}$, this gives the relations:

$$\phi = \arctan\left(\frac{p_x}{p_y}\right)$$

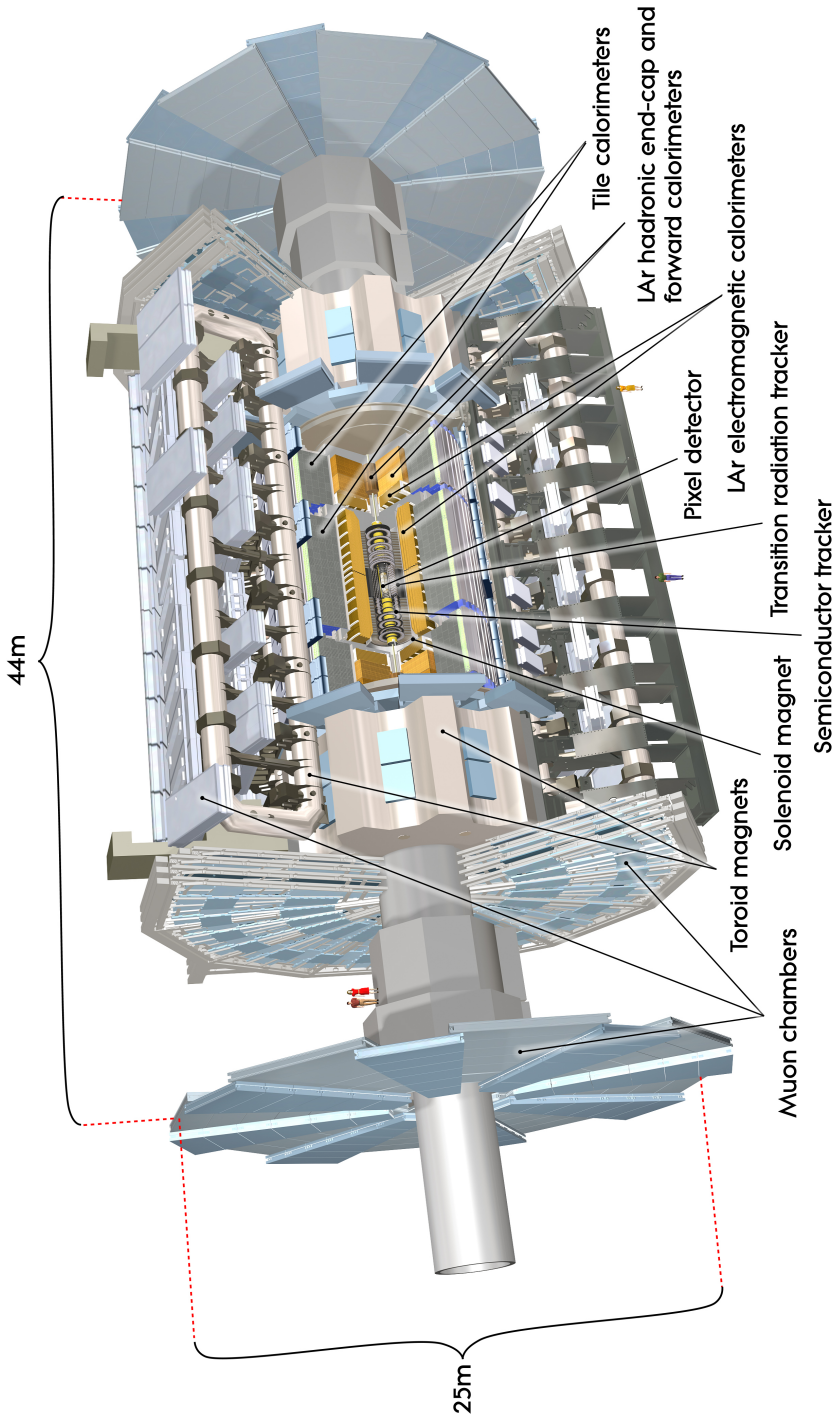


Figure 3.3 Cut-away view of the whole ATLAS detector [48].

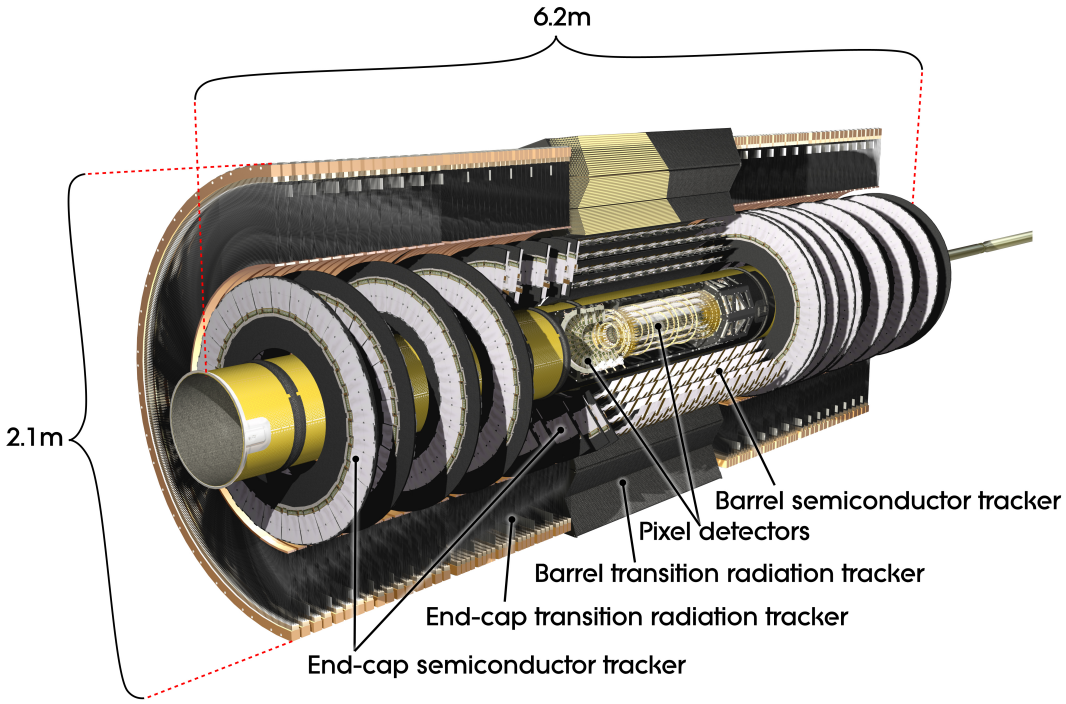


Figure 3.4 Cut-away view of the ATLAS inner detector [49].

$$\theta = \arctan\left(\frac{p_T}{p_z}\right)$$

where $p_T = \sqrt{p_x^2 + p_y^2} = |\vec{p}| \sin\theta$ is called *transverse momentum*. Similarly the *transverse energy* is defined as $E_T = E \sin\theta$.

The ATLAS experiment is described at length in [10]. The following brief descriptions are largely based on that reference.

3.3.2 Inner detector

The inner detector (ID), shown in figure 3.4, is designed for high-precision momentum and vertex measurements of charged particles in the dense particle environment around the interaction point. It is enclosed by a niobium-titanium superconducting solenoid, operated at 4.5 K, that produces a magnetic field of 2 T in its centre. Charged particles induce a signal in different parts of the ID as they traverse it. These signals, called *hits*, can be joined to reconstruct the path of the particle, from which the momentum and origin of the particle can be calculated. The inner detector can reconstruct tracks with

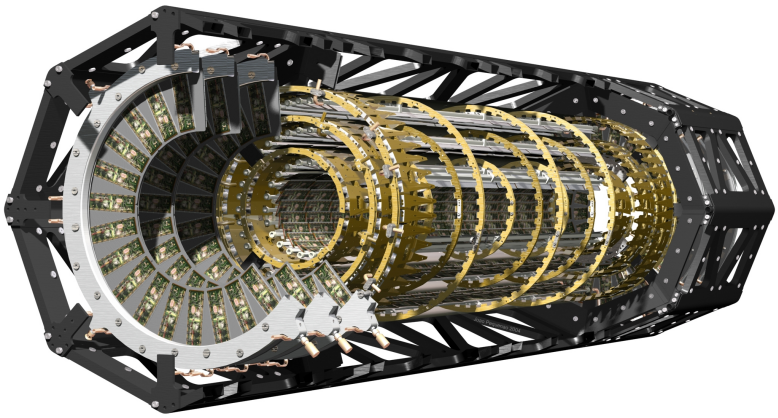


Figure 3.5 Cut-away view of the ATLAS pixel detector [51].

a precision of $O(10\,\mu\text{m})$. Correct operation in the harsh environment in the vicinity of the interaction point requires all sensors and readout electronics to be radiation-hard.

The sub-detector with the highest granularity is the silicon pixel detector. It consists of 1744 modules containing 47232 pixels each, organised in three concentric cylinders parallel to the beam in the barrel region (from $R = 5\,\text{cm}$ to $R = 12\,\text{cm}$) and three discs perpendicular to the beam in each end-cap (see figure 3.5). When a charged particle traverses a pixel, electron-hole pairs proportional to the energy loss are created. A 150 V voltage prevents the pairs from recombining, inducing a signal that is read out and translated to a hit. The intrinsic accuracies of a pixel module are $10\,\mu\text{m}$ (in $R\phi$) and $115\,\mu\text{m}$ (in z for the barrel part, in R for the end-caps). The pixel detector is the most exposed to radiation, so much so that in 2014, after three years of high-luminosity operation, a fourth cylinder was inserted between the the beam pipe and the rest of the pixel detector. This new layer is needed to supplement the radiation damaged innermost layer of the existing detector.

Immediately outside the pixel sub-detector lies the semiconductor tracker (SCT). It consists of 16352 modules containing two layers of silicon micro-strips with a pitch of $80\,\mu\text{m}$. The two layers are at a stereo angle of 40 mrad from each other, enabling the determination of two spatial coordinates. The modules form 4 cylinders in the barrel region (from $R = 30\,\text{cm}$ to $R = 51\,\text{cm}$), where the strips of one of the two layers are oriented in the beam direction, and 9 discs in each end-cap, where the strips of one of the two layers are in the radial direction. Each module has an intrinsic accuracy of $17\,\mu\text{m}$ (in $R\phi$) and $580\,\mu\text{m}$ (in z for the barrel part, in R for the end-caps), with the latter being sacrificed by the small stereo angle chosen, which is needed to keep the amount of ghost hits due to

combinatorial effects small. Both the pixel and the SCT sub-detectors cover the region with $|\theta| > 9°$.

The outermost component of the inner detector is the transition radiation tracker (TRT). It consists of some 420,000 cylindrical *drift tubes*, a type of gaseous ionisation detector. These detectors consist of two electrodes (anode and cathode) separated by a gas. A particle traversing the detector will hit some of the gas molecules, creating ion pairs consisting of an electron and a positive ion. The two electrodes are kept at an electric potential difference, so that the ionisation electrons will drift towards the anode and the positive ions will drift towards the cathode, producing an electrical current, which is measured to detect the passage of the particle. In drift tubes, the sides of the tube are the cathode; the anode is a wire in the centre of the tube. In the TRT, each tube (called *straw*) has a diameter of 4 mm and contains a mixture of xenon, tetrafluoromethane and carbon dioxide. The straws are oriented parallel to the beam direction in the barrel and radially in the end-caps, therefore the TRT only provides information in $R\phi$. A straw's intrinsic accuracy is 130 μm which is an order of magnitude larger than the pixel and SCT sub-detectors, but is compensated by the larger number of hits which is typically 36 per track. The TRT sub-detector covers the region with $|\theta| > 15°$.

3.3.3 Calorimeters

The calorimetry system, shown in figure 3.6, is divided into the electromagnetic calorimeter in the region $|\theta| > 5°$, the hadronic barrel calorimeter covering $|\theta| > 20°$, the two hadronic end-cap calorimeters at $5° < |\theta| < 25°$ and the two forward calorimeters at $1° < |\theta| < 5°$. All the calorimeters in ATLAS are *sampling calorimeters*. In sampling calorimeters, layers of a dense material alternate with layers of an active component. Incoming particles interact with the dense material producing a cascade of secondary particles known as *shower*. The active material emits a signal that is proportional to the energy of the shower that traverses it. Showers produced by photons, electrons and positrons are contained in the electromagnetic calorimeters. Hadronic showers, which start in the electromagnetic calorimeters, are absorbed in the hadronic calorimeters.

The electromagnetic calorimeter uses liquid argon as active medium, with accordion-shaped absorber plates interleaved with readout electrodes. It is divided into a barrel part ($|\theta| > 25°$) and two end-caps ($5° < |\theta| < 28°$). Each end-cap calorimeter is further divided into two coaxial wheels: an outer wheel that overlaps with the inner detector ($9° < |\theta| < 28°$) and an inner wheel ($5° < |\theta| < 9°$). The readout electrodes are seg-

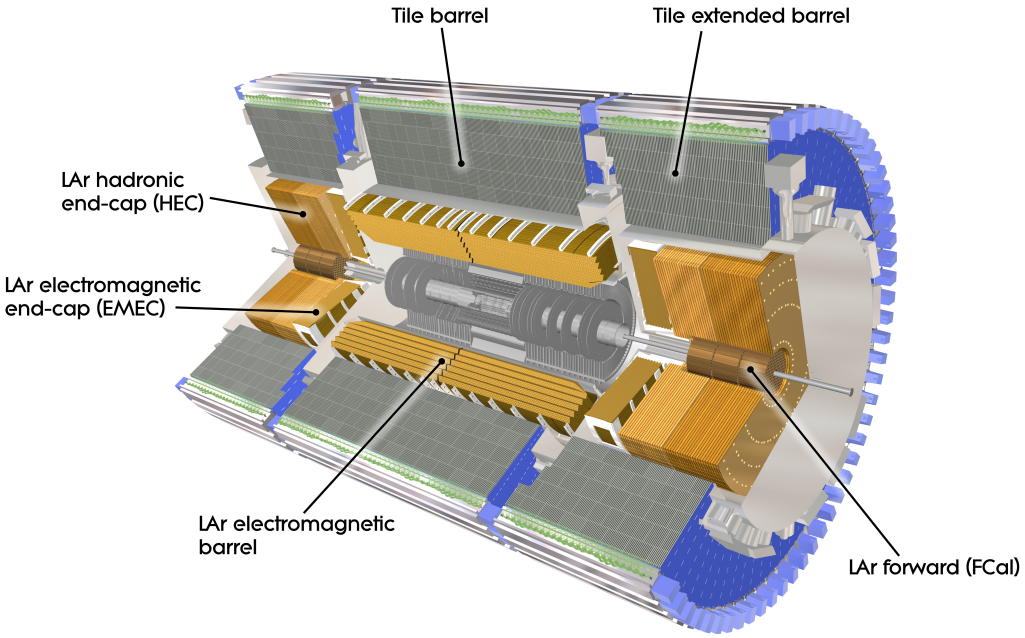


Figure 3.6 Cut-away view of the ATLAS calorimeters [48].

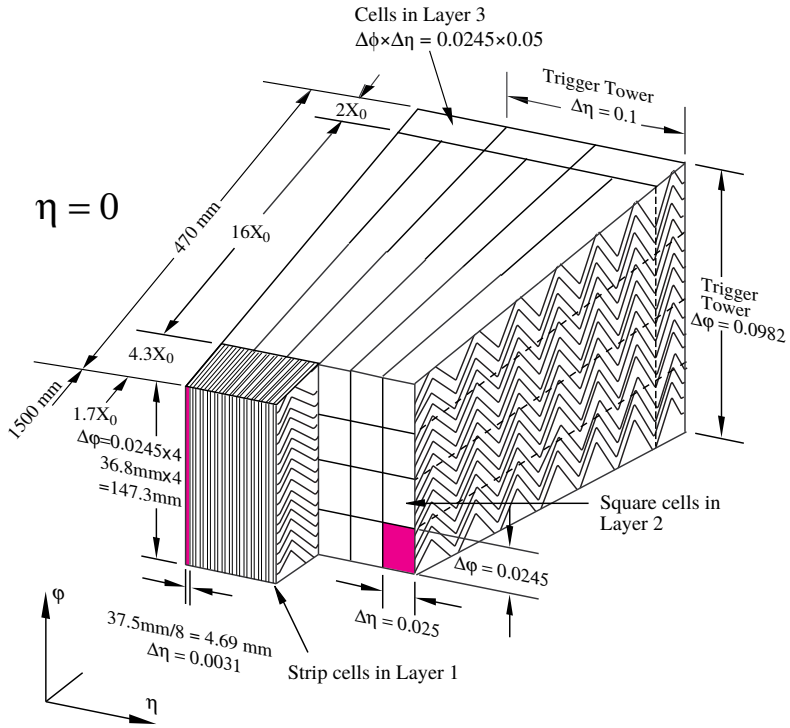


Figure 3.7 Electromagnetic calorimeter granularities [10]. NB: $\eta = -\ln[\tan(\theta/2)]$.

mented into three longitudinal sections, except for the inner wheel, which is segmented into only two sections and has a coarser lateral granularity than the remaining parts. The granularities of each section are detailed in figure 3.7. The design resolution of the electromagnetic calorimeter is $\sigma_E/E = 10\%/\sqrt{E} \oplus 30\%/E \oplus 0.7\%$, with E in GeV.

The hadronic barrel calorimeter uses scintillating tiles as active medium and steel as absorber. Two sides of the tiles are read out by wavelength shifting fibres into photomultiplier tubes. It is azimuthally segmented in 256 modules. Each module is further split into three layers along the radial coordinate.

The hadronic end-cap calorimeters (HEC) use liquid argon as active medium and flat copper plates as absorber. Each end-cap consist of two independent wheels, which are in turn divided into two segments in depth, for a total of four layers per end-cap, with the inner layers having a higher sampling frequency than the outer. The design resolution of the combined barrel and end-cap hadronic calorimeters is $\sigma_E/E = 50\%/\sqrt{E} \oplus 5\%$, with E in GeV.

The Forward Calorimeter (FCal) consists of three modules, where the first module uses copper as passive medium and is optimised for electromagnetic measurements, while the second and third are made of tungsten and dedicated to hadronic energy measurements. Each module consists of a metal matrix, with regularly spaced longitudinal channels filled with the electrode structure consisting of concentric rods and tubes parallel to the beam axis. Liquid argon in the gap between the rod and the tube is the active medium. The design resolution of the FCal is $\sigma_E/E = 100\%/\sqrt{E} \oplus 10\%$, with E in GeV.

3.3.4 Muon spectrometer

The muon spectrometer, shown in figure 3.8, is characterised by the experiment's namesake: the toroidal air-core superconducting magnet system. It consists of 8 huge coils in the barrel region (each in a separate cryostat) and two end-caps with eight smaller coils each (in common cryostats). With a toroidal configuration for both the barrel and the end-cap magnets, the magnetic field lines roughly assume the shape of circles centred in the z-axis. Therefore, the particles will cross the detector almost perpendicularly to the field in a wide θ range, keeping the bending power high even in the endcap regions. The barrel toroid provides a peak field of 3.9 T and the end-cap toroids 4.1 T. Muon tracks are measured in the bending magnetic field in the range $|\theta| > 8°$.

In the barrel region ($|\theta| > 39°$) tracks are measured in chambers arranged in three cylindrical layers, enabling the measurement of their momentum from their sagitta. The

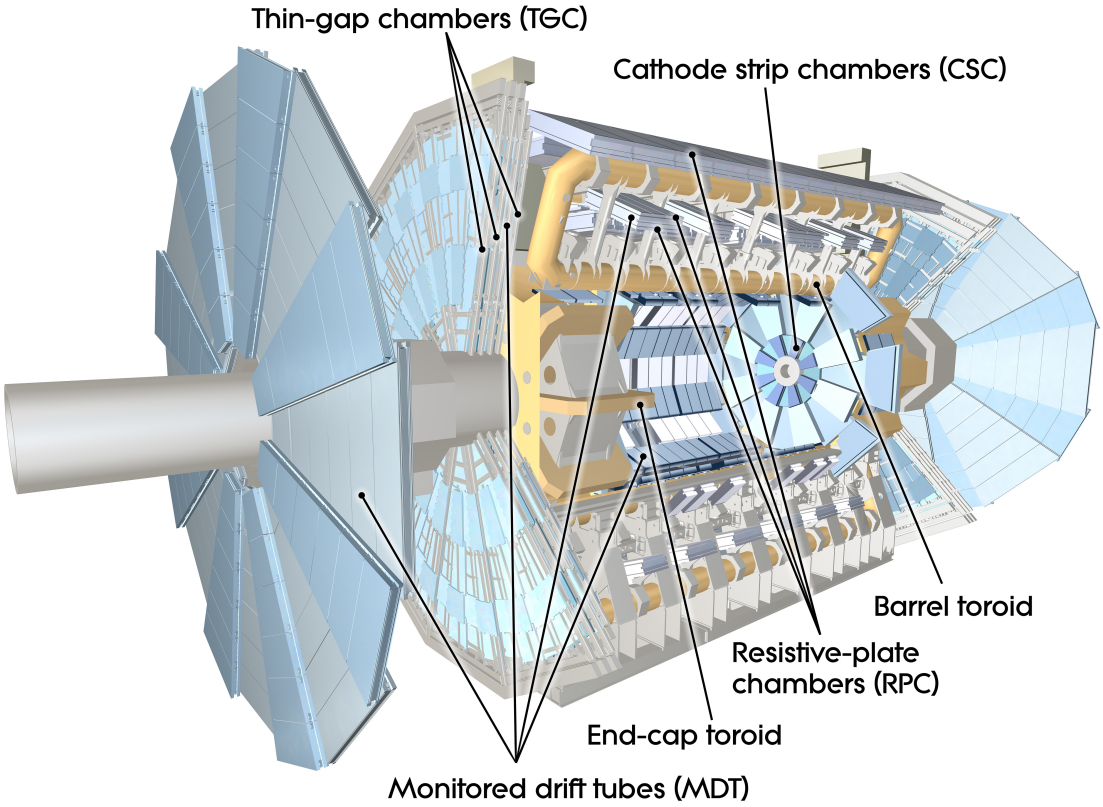


Figure 3.8 Cut-away view of the ATLAS muon spectrometer [50].

end-cap chambers cover the range $40° < |\theta| < 8°$ and are organised in one small wheel and two big wheels per end-cap. Four different gaseous ionisation detector technologies are employed, with the first two (MDT and CSC) being used for precision measurements and the other two (RPC and TGC) for their fast reaction times.

Monitored drift tubes (MDTs) are used in most regions of the spectrometer. MDT chambers are based on single-wire drift tubes of 30 mm diameter operated with a mixture of argon and carbon dioxide. The precision coordinate is obtained measuring the ionization-electrons drift time. The chamber spatial resolution is 35 μm, with the single-wire intrinsic resolution being 80 μm. The MDT chambers only measure one coordinate (z in the barrel region and R in the end-caps). Their measurements are complemented in the orthogonal coordinate by the fast chambers.

Cathode Strip Chambers (CSCs) are *multi-wire proportional chambers* with cathode-strip readout operated with a carbon dioxide, argon and tetrafluoromethane gas mixture. The precision coordinate is obtained by measuring the charge induced on the segmented cathode by the avalanche formed on the anode wire. Spatial resolutions of 40 μm (in R) and 5 mm (in $R\phi$) are achieved by segmentation of the cathode and by charge interpolation between contiguous strips. The CSCs are particularly suited to operation in high particle flux regions having small electron drift time (30 ns), good time resolution (7 ns) and low neutron sensitivity. They are therefore employed instead of MDTs in the innermost ring of the small wheels.

Resistive Plate Chambers (RPCs) are gaseous detectors providing an excellent time resolution of 1 ns, at the cost of a typical spatial resolution of 1 cm (in R and $R\phi$). The basic RPC unit is a narrow gas gap formed by two parallel resistive bakelite plates separated by insulating spacers and operated with tetrafluoroethane mixed with small amounts of sulphur hexafluoride. Primary ionisation electrons are multiplied by a high, uniform electric field and the signal is readout via capacitive coupling of metal strips on both sides of the detector.

Thin Gap Chambers (TGCs) are multi-wire proportional chambers in which the anode wires provide a fast signal while the readout strips, orthogonally arranged with respect to the wires, are used to measure the second coordinate. They are operated with a gas mixture of carbon dioxide and n-penthane. The chamber time resolution is 4 ns, while their spatial resolution varies from 2 mm to 6 mm (in R) and from 3 mm to 7 mm (in $R\phi$).

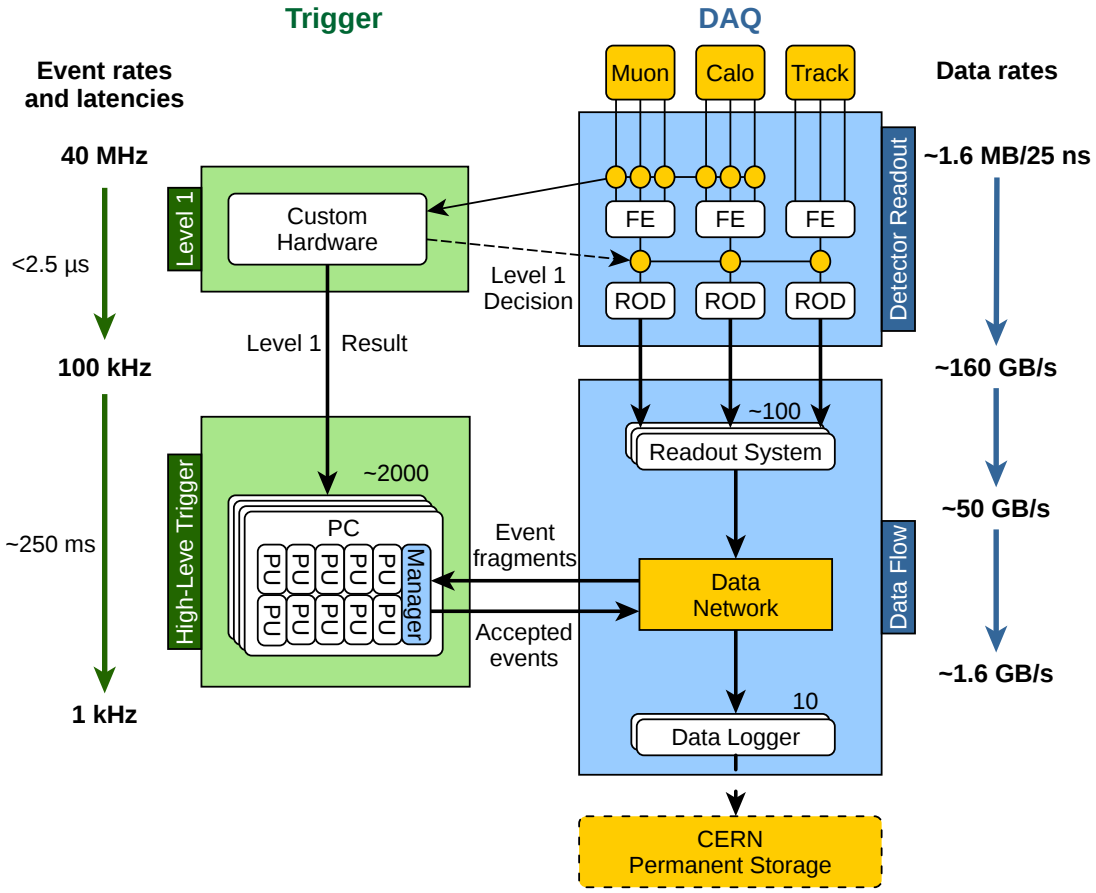


Figure 3.9 Logical view of the components of the trigger and data acquisition system. The trigger path is sketched on the left, along with the input and output event rates and event processing latencies. The data path is sketched on the right, along with the expected input and output data rates, assuming an event size of 1.6 MB.

3.4 The ATLAS trigger and data-acquisition system

In design conditions, the LHC will collide proton bunches in the centre of ATLAS at a rate of 40 MHz. Each bunch crossing and the resulting interactions are referred to as an *event*. A fully acquired event corresponds to 1–2 MB of data. If all events were permanently stored, a storage bandwidth of up to 80 TB/s would be required. However, as discussed in section 3.2.2, most events only result in well-known physical processes that do not merit further analysis. Therefore it is not necessary to save data corresponding to all events. Instead, real-time event selection is used. This selection mechanism is known as *trigger*: the trigger "fires" only when an event with potentially interesting

interaction is observed.

The ATLAS Trigger and Data-Acquisition (TDAQ) system is outlined in figure 3.9. It is organised in two levels of event selection, with the second trigger level refining the decisions made at the previous level and, where necessary, applying additional selection criteria. Starting from the nominal bunch crossing rate of 40 MHz, the rate of selected events must be reduced to a budgeted average rate of a few kHz for permanent storage. In terms of orders of magnitude, this means that only one out of 10^4 events can be recorded, i.e. the trigger system must deliver a total rejection factor of $O(10^4)$ against uninteresting events, while retaining rare new physics processes.

The first trigger level (L1) is implemented in custom-built electronics which analyse the information coming from the fast muon chambers and the calorimeters to operate a coarse event selection with a rejection factor of $O(10^2)$. The second level (High-Level Trigger, HLT) is a distributed software system which access to the detector data at full granularity. The HLT refines the selection of the L1, delivering a further rejection factor of $O(10^2)$ and sends the finally accepted events to the data-logging system.

3.5 Front-end

In total, the ATLAS sub-detectors provide close to 100 million analogue or digital outputs, called *readout channels*.

Although each sub-detector handles its signals using custom electronics, known as *front-end*, whose detailed description is outside the scope of this thesis, these components are built from standardised blocks and are subject to common requirements. The front-end electronics include different functional elements [10]:

- analogue or analogue-to-digital signal processors,
- interfaces to the L1 system for receiving beam timing and trigger signals via the Timing Trigger and Control (TTC) system [11],
- L1 pipelines (analogue or digital) in which the information is retained for a time long enough to wait for the L1 trigger decision,
- derandomising buffers in which the data accepted by the L1 trigger are stored before being sent to the following level,
- dedicated links or buses which are used to transmit the front-end data stream to the next stage.

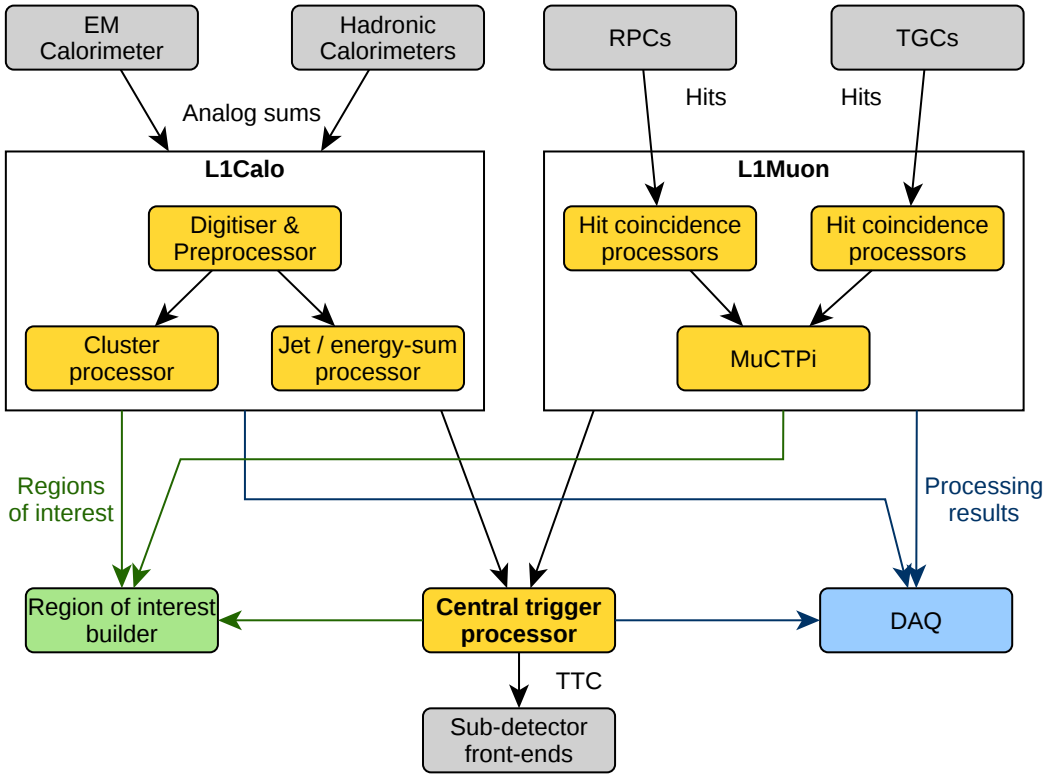


Figure 3.10 Outline of the level-1 trigger system

If an event is accepted by the L1 trigger, its data is moved from the detector front-end pipelines to the Readout Drivers (RODs) electronics boards. The RODs serve as an initial data aggregation step, combining fragments from several front-end data streams. They also implement specific error-detection mechanisms and format the sub-detector data according to a format common to ATLAS. The aggregated bits, called event fragments, are then pushed to the Data-Acquisition system (DAQ) via special-purpose point-to-point optical Readout Links (ROLs).

3.6 First-level trigger

The L1 trigger system, represented in figure 3.10, is designed to reduce the event rate to a maximum of 100 kHz with a latency of less than 2.5 μs. The limiting factor that imposes these maximum thresholds is the performance of the sub-detector front-end electronics.

Most of the interactions that are considered interesting for the research programme,

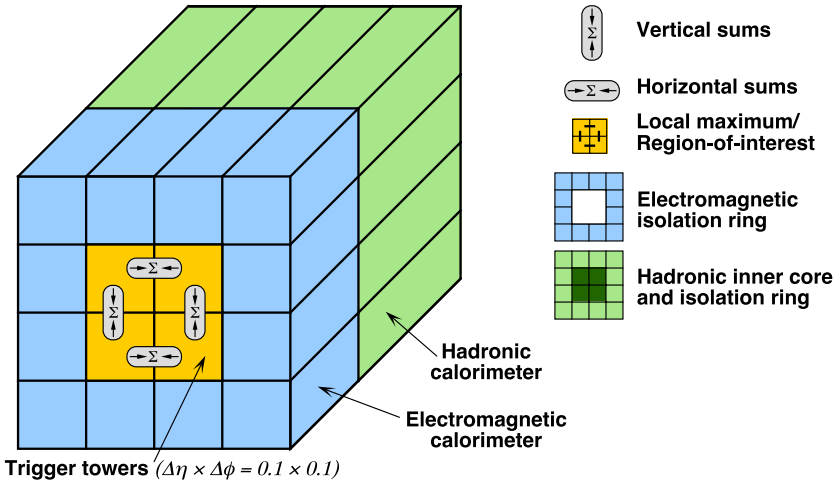


Figure 3.11 Building blocks of the Level-1 cluster processor algorithms

result in the production of one or more isolated particle with high transverse momentum (p_T, see section 3.3.1). The L1 trigger is therefore designed to find the signals left by these particles and use them to quickly decide whether an event is to be sent to the HLT or rejected.

The L1 trigger decision is formed by the Central Trigger Processor (CTP) [8] based on coarse information from the L1 Calorimeter Trigger system (L1Calo) and from the fast muon spectrometer chambers (RPCs and TGCs). The L1 does not use information from the inner detector trackers: reconstructing tracks from hits is a combinatorial problem and given the huge amount of particles traversing the inner detector, doing so would cause the L1 to exceed its strict latency bounds.

The CTP distributes the trigger decision, together with timing signals from the LHC, to the front-end electronics of all the sub-detectors. Following an accept decision, the CTP introduces *dead-time*, by preventing further triggers for a specified number of bunch crossings. This gives the front-ends the time they need to store the event data in their pipelines. The CTP also restricts the total number of accept decisions allowed in a given period to prevent the front-end pipelines from overflowing.

3.6.1 Calorimeter trigger

In order to be fast, the L1Calo [1] does not make use of the full granularity of the calorimeter data. Instead, it operates on analogue sums from calorimeter elements within coarser regions, called *trigger towers*. Electromagnetic and hadronic calorimeters have

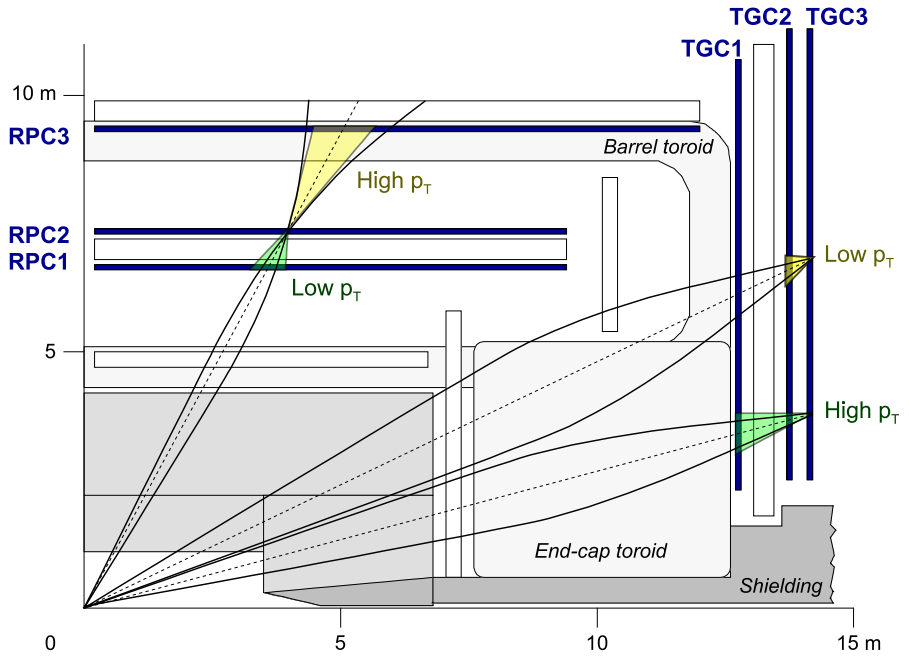


Figure 3.12 Representation of the level-1 muon hit coincidence logic.

separate trigger towers. The analogue inputs are first digitised and associated to a particular LHC bunch crossing. The data is then processed in parallel by two processing systems. The *cluster processor* searches for small localised clusters of towers with high energy deposition, typical of electrons, photons and tauons. The *jet and energy-sum processor* uses 2×2 sums of trigger towers, to identify sprays of hadrons (called *jets*) and to calculate global energy sums. The magnitude of the objects and sums are compared to programmable thresholds to form the trigger decision.

The building blocks of the cluster processor algorithms are shown in figure 3.11: a 2×2 region-of-interest is identified in the electromagnetic calorimeter for which the energy sum from at least one of the four possible pairs of nearest neighbour towers exceeds a pre-defined threshold. A maximum energy threshold can be set for the towers surrounding the region-of-interest, to require that the energy deposition is isolated from others. A similar approach is used in the jet and energy processor.

3.6.2 Muon trigger

The muon trigger logic searches for muons with transverse momentum (p_T) over configurable thresholds. It is directly implemented by the fast muon sub-detectors, RPCs

and TGCs, both using similar algorithms [7]. As highlighted in figure 3.12, the detector chambers are arranged in three planes in both the barrel region and the end-caps. They are grouped into sectors that share common trigger electronics. In each sector, up to two muon candidates are identified by searching for coincidences of hits among the planes. In order to form a coincidence, hits are required to lie within parametrised geometrical regions called *muon roads* (the yellow and green areas in the figure). A road represents the envelope containing the trajectories from the interaction point of muons (or antimuons) with a transverse momentum greater than a chosen threshold (the faster a muon, the larger the curvature radius of its trajectory in the magnetic field). As an example, in figure 3.12 the (anti)muon tracks in the barrel region lie in the "low-p_T" muon road, but do not fit into the "high-p_T" road. Such a track would therefore only fire the "low-p_T" muon trigger. A track with a lower curvature radius would not fire any trigger.

For each bunch crossing, the sector logic sends to the *muon-to-CTP interface* (MuCTPi) the information about the position and transverse momentum of the muon candidates. The MuCTPi combines the information from all the sources, resolving possible double counting of muon candidate tracks that traverse more than one sector.

3.7 Data-Acquisition and High-Level Trigger

The Data-Acquisition (DAQ) system interfaces with the detector readout and the L1 trigger on the input side, with the High-Level Trigger for event selection, and with the mass storage in the CERN computing centre on the output side. Together, the DAQ and HLT systems form a distributed software system running on around 2000 Linux PCs. Two independent Ethernet networks interconnect all the nodes in the system. One network, called *control network*, provides infrastructure and operational services. A second, called *data network*, is used exclusively for transferring event data. The network layout is described in more detail in section 3.7.6.

The data flow in the DAQ and HLT systems is shown in figure 3.13. Once an event is accepted at the L1 trigger, its fragments are pushed into the Readout System, which acts as the interface between the readout and the DAQ. An available HLT Processing Unit incrementally retrieves and analyses the fragments, until a trigger decision can be taken. Fragments of rejected events are deleted from the readout systems buffers, while accepted events are transferred to one of the Data Logger nodes for storage and asynchronous transfer to the CERN central data storage facility. The DAQ input event

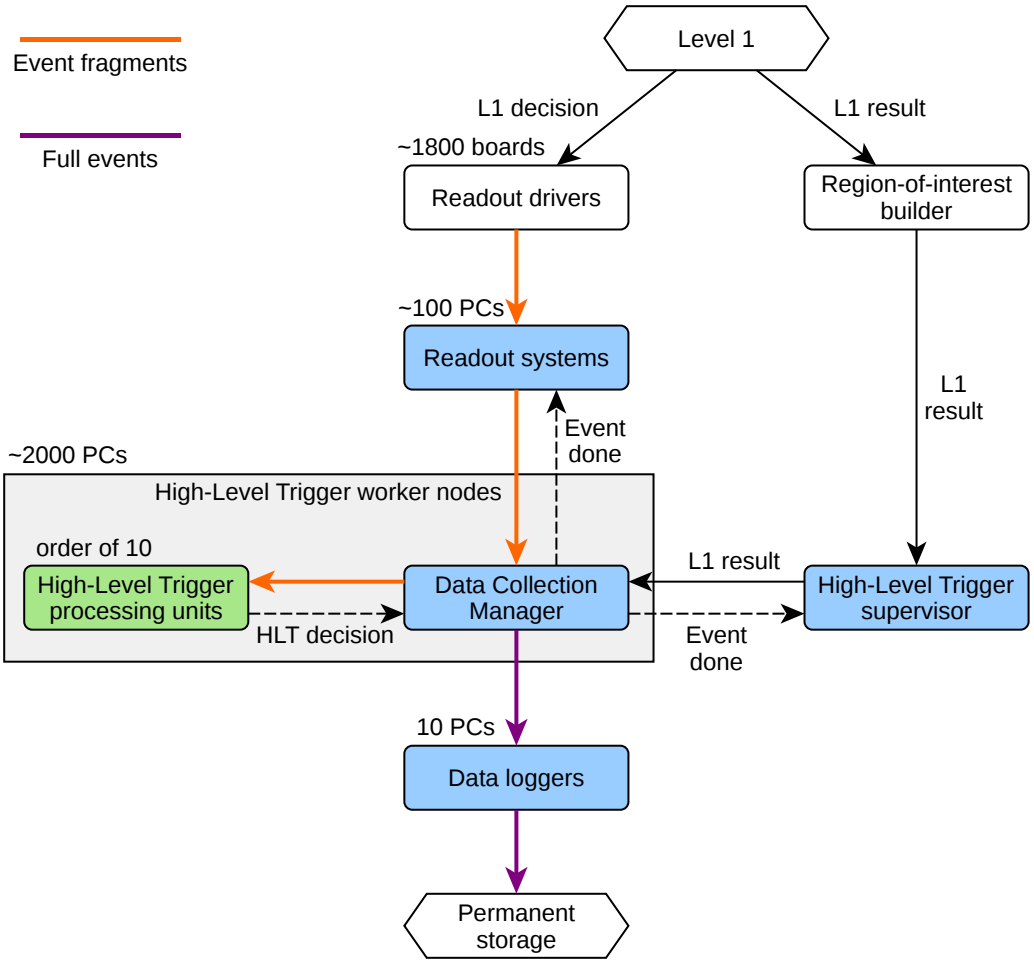


Figure 3.13 Components of the Data-Acquisition and High-Level Trigger systems.

rate is the L1 output rate of 100 kHz, i.e. 100–200 GB/s for event sizes in the range 1–2 MB.

3.7.1 Data format

The format of the ATLAS event data [6] reflects the structure of the data-flow. The DAQ system does not impose any fixed structure to the raw data produced by the sub-detectors readout. The event fragments formatted by the RODs consist of a header, the raw data received from the front-end electronics and a trailer. Among other fields, the header specifies the source of the fragment, the L1 Identifier (L1Id) and the Bunch Crossing Identifier (BCId), along with sub-detector-specific meta-data. The L1Id and BCId are counters that are incremented for every L1 accept decision and bunch crossing, respectively. The trailer contains a check sequence used to verify data integrity.

When an event is finally assembled to form a *full event* after being accepted by the HLT, the fragments are concatenated and a full event event header is added, which includes all the necessary fields to uniquely identify the event throughout the lifespan of the ATLAS experiment.

3.7.2 Readout System

When an event is accepted by the L1 trigger, each of its fragments is pushed from its sub-detector front-end pipeline, via a Readout Driver, into a Readout Link (ROL) that constitutes the interface between the sub-detector specific electronics and the common DAQ system. Each ROL delivers its event fragments to a specific Readout System (ROS) node. There are over 1800 ROLs connecting the sub-detectors to the DAQ.

The ROS consists of around 100 Linux PCs housing two purpose-built PCIe cards, called ROBINs, and connected via four 10 Gb/s Ethernet links to the data network. The ROLs are point-to-point optical fibre links using a custom protocol with a maximum bandwidth of 160 MB/s and in-band flow control (XON/XOFF). Each ROBIN can handle up to 12 ROLs. It receives the incoming event fragments and stores them in the 8 GB on-board memory, or asserts an XOFF flow-control signal if the memory is full. On each PC a multi-threaded application handles the interaction with the rest of the DAQ system via the data network. It receives requests for event data, forwards them to the ROBINs and sends the corresponding fragments to the requester. The application also instructs the ROBINs to delete data on request from the DAQ system.

3.7.3 High-Level Trigger

The HLT system comprises the HLT Processing Units (HLTPU) and the HLT Supervisor (HLTSV). Each HLT worker node hosts one HLTPU per CPU core and a single Data-Collection Manager (DCM), which handles communications with the rest of the system on behalf of the HLTPUs on the node.

At the same time as the event fragments are transferred from the front-end to the ROS, the L1 trigger also sends some additional information about its decision to the High-Level Trigger Supervisor (HLTSV). This information, called L1 result, comes from various sources in the L1 trigger. A ROD-like device called Region-of-Interest builder combines information from all L1-result sources into a single fragment and sends it to the supervisor via a point-to-point link.

The HLTSV runs on a PC in a hardware configuration similar to the ROS, but its purpose is different. Sending the L1 result to the HLTSV communicates that a certain event has been accepted by the L1 and its data is available from the ROS. The HLTSV must then assign the event to an available HLTPU for processing. To do so, the HLTSV sends the L1 result to the chosen HLTPU. If no processing units are available, the supervisor signals this condition by asserting an XOFF flow-control signal on the link from the L1. This will cause the L1 to stop accepting events until the XOFF is cleared.

The L1 decision data contains information about the so-called *regions-of-interest*. A region-of-interest is a geometrical region of ATLAS that observed signals considered interesting by the L1 (e.g. a high-momentum muon or a big energy deposition). Using the regions-of-interest as starting points, the processing unit incrementally retrieves and analyses event fragments, until it has enough information to take a decision. The event can be rejected even without analysing all its fragments, thus limiting the fraction of data to be retrieved from the ROS and, as a consequence, lowering the total cost of the system. Fragments of rejected events are deleted from the ROS buffers, while accepted events are transferred to one of the Data Logger nodes for storage.

3.7.4 Data-Collection Manager

Data integrity guarantees and error detection are of the utmost importance in a DAQ system. To this end, the DAQ and HLT systems are designed to decouple the trigger processing of events from the DAQ operations. The rationale behind this choice is simple: a software bug that causes data loss and is only set off by specific kinds of events would

irremediably introduce a bias in the data being acquired. While it is reasonably possible to ensure that the DAQ software is not affected by such bugs, it is much more difficult to do so with the trigger software.

The DAQ applications do not need to be aware of the specific structure of the event data. Indeed, as far as the DAQ system is concerned, all the operations can proceed manipulating only the headers described in section 3.7.1. As the headers have a relatively simple format, the probability of a software bug that is triggered only by specific data patterns is greatly reduced and can be minimised by testing the software with the limited number of possible variations of the headers. Moreover, since the header format is very stable, the data handling code on the DAQ side changes rarely.

The trigger software, on the other hand needs to interpret the raw sub-detector data in order to be able to analyse and select the events. Furthermore, its algorithms make up a rather large body of software and change often as they are improved or replaced with more efficient versions. This makes it much more susceptible to bugs and crashes caused by unexpected data.

This isolation is achieved by separating the data-flow logic and the trigger logic in two different applications. On each HLT worker node, the Data-Collection Manager (DCM) application handles the former, while multiple HLT Processing Units handle the latter.

Each DCM is connected to all the other applications in the DAQ system:

- When the HLTPUs on its node are available, it asks the HLTSV for new L1 results to process; it then forwards the L1 result to an available HLTPU.
- On behalf of a HLTPU, it requests event fragments from the ROS and makes them available to the HLTPU.
- When a HLTPU decides that an event is to be accepted, it assembles all the event fragments into a full event and ships it to one of the Data Logger for storage.
- Once an event has been either rejected or successfully stored, it asks the ROS to delete its fragments from their buffers.

The event fragments fetched by the DCM are made available to the HLTPU via read-only shared memory, so that no bugs on the trigger side can cause data corruption or loss. In fact, once an event fragment is received, the DCM asks the operating system to consider the memory that holds it read-only, even for the DCM itself, so that in the unlikely case of a data-corrupting bug in the DCM, the application would encounter an error and terminate rather than mangle the data. Moreover, it is possible to configure

the DCM so that it does not ask the ROS to delete the fragments corresponding to an event before the Data Logger has confirmed that the event is safely on disk.

If a HLTPU crashes while processing an event, or simply takes an excessive amount of time to come to a trigger decision, the DCM marks the event as a possible cause for crashes, and sends it to the a Data Logger without further processing, rather than re-assigning it to another unit. In this way, the crash can be investigated later without risking that the offending event cause even more crashes in the system. A similar mechanism is implemented by the HLTSV: in case it loses contact with a DCM, be it because the DCM itself crashed, or the more likely case of hardware failure, it will re-assign the event to another DCM, which will send it directly to storage.

3.7.5 Data Logger

The data-logging system consists of six Linux PCs executing the Data Logger application. Each node is equipped with three 20 TB redundant disk arrays and is connected with two 10 Gb/s links to the data network.

The core functionality of the Data Logger application [22] is to receive event data from the DCM and save it to the appropriate raw-data files. A separate application running on the same machine asynchronously transfers the files to the central mass storage facility at CERN for permanent storage. The raw files are deleted from the Data Logger nodes only once the mass storage facility confirms that the data is stored on both disk and tape.

3.7.6 Network

The layout of the DAQ data network [55] is shown in figure 3.14. The core of the system consists of two large network routers (Brocade MLXe-32) with a maximum capacity of several hundred 10 Gb/s Ethernet ports.

Readout systems are directly connected to both core routers with 4 redundant 10 Gb/s Ethernet links, while the HLT supervisor node has one 10 Gb/s going to each router. HLT worker nodes are organised in racks of at most 40 nodes. Each node in a rack is connected to an aggregation switch with a 1 Gb/s link. The rack switches have 10 Gb/s uplinks to both core routers.

The two core routers are in an active-active redundant configuration: even if one of them becomes unreachable, all nodes can still communicate, possibly at reduced bandwidth,

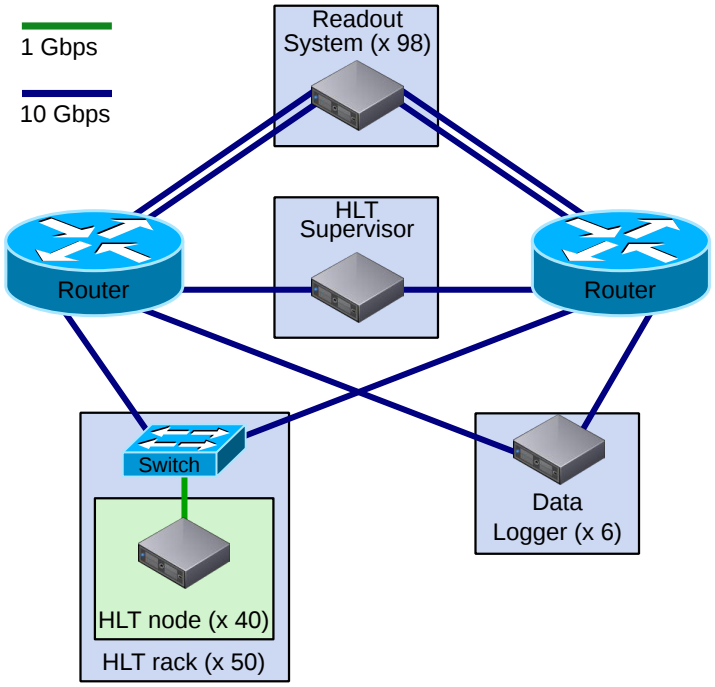


Figure 3.14 Network architecture of the ATLAS Data-Acquisition and High-Level Trigger system.

via the other router. Under normal conditions, with both core routers available, the traffic is equally distributed over the available links. More precisely, the hosts and the switches that are connected to the core routers by two or more links choose the link on which a packet is sent according to a hash of the packet's data-link and network addresses.

3.8 The ATLAS Data-Acquisition messaging system

3.8.1 Requirements

The applications in the ATLAS DAQ exchange network messages with each other in order to accomplish their purpose. In particular:

- The HLTSV sends a message to a DCM to assign a L1 result to an available HLTPU on the node where the DCM runs. The DCM sends a message to the HLTSV when the HLTPU finishes processing and is thus available again. This happens for every L1 accepted event.

Peers		Message rate per connection	Message size	
A	B		A→B	B→A
HLTSV	DCM	50 Hz	1 kB	10 B
ROS	DCM	25 Hz	20 kB	100 B
DCM	Data Logger	0.2 Hz	1 MB	10 B

Table 3.1 Typical messaging patterns between ATLAS DAQ applications.

Application	Connections	Message rate		Bandwidth	
		Send	Receive	Send	Receive
HLTSV	2000	100 kHz	100 kHz	100 MB/s	1 MB/s
ROS		50 kHz	50 kHz	1 GB/s	50 MB/s
Data Logger		500 Hz	500 Hz	500 B/s	250 MB/s
DCM	110	2.5 kHz	2.5 kHz	250 kB/s	50 MB/s

Table 3.2 Messaging requirements of ATLAS DAQ applications in ordinary conditions.

- The DCM sends requests for data fragments to the appropriate ROS nodes on behalf of the HLTPUs on its node; the ROS nodes reply with the data. This happens several times per each L1 accepted event.
- The DCM asks a Data Logger if it can receive an event for storage. The Data Logger requests the event from the DCM. The DCM sends the event and the Data Logger acknowledges.

In essence, each DCM in the system exchanges messages with the HLTSV, all the ROS nodes, and all the Data Logger nodes. The main features of these messaging patterns are summarised in table 3.1. These are typical quantities estimated on the basis of a 100 kHz L1 output rate, a 2 MB average event size, and 2000 nodes in the HLT farm.

The messaging patterns impose relatively different requirements on the different applications in the system. As shown in table 3.2, the HLTSV must handle a very high rate of small messages; the ROS need to handle slightly bigger messages at a slightly lower rate; the Data Loggers must handle big messages but at a low rate. All applications manage a relatively high number of connections.

Even with these relatively disparate messaging requirements, having a messaging layer common to all applications is desirable, for ease of maintainability. The common layer should satisfy the following requirements:

- Can service the messaging patterns in table 3.2 with acceptable performance and with margin to handle unusual conditions

- Reliable, in-order, point-to-point message delivery
- Simple recovery from connection failures

3.8.2 Existing solutions

The requirements outlined in the previous sections are relatively simple. Accordingly, they could be satisfied by a variety of existing messaging technologies. There are three main options to be considered for an implementation:

- *message brokers* such as Apache ActiveMQ and similar products,
- *remote procedure call* software such as the Common Object Request Broker Architecture (CORBA) standard and its implementations,
- *high-performance computing* middleware such as the Message Passing Infrastructure (MPI) standard and its implementations.

All of these options offer much more functionality than required by the DAQ system.

Message brokers provide complex, configurable message queuing, routing and transformation. These features require every message to go through a central service, the broker, which can then apply its configured message handling policies.

Remote procedure call systems enable invocation of routines residing in different (i.e. remote) processes. Their goal is to provide the same interface for local and remote invocations. Thus, most of their functionality is devoted to serialising function calls into messages suitable for transmission (marshalling) and vice-versa.

High-performance computing messaging is dominated by a single standard with multiple implementations: MPI. MPI is designed with a specific use case in mind: parallel applications running on homogeneous computer clusters. It includes features such as process management, collective communication (one-to-many, many-to-one, many-to-many), topology discovery.

None of the functionality described above is really needed to satisfy the requirements of a messaging layer for the DAQ system. Advanced message queueing and routing are not needed, given that all the communication patterns are point-to-point. The same can be said for collective communication. MPI-like process management is rather invasive and duplicates functionality already present in the experiment's control system. Message serialisation is also not necessary: most of the exchanged data is already serialised according to the data format described in section 3.7.1. All these extra features come at

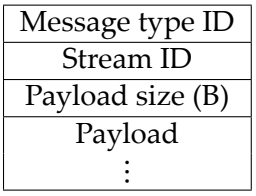


Figure 3.15 Anatomy of a DAQ message

the price of significant additional complexity, which in turn makes troubleshooting and tuning the performance of the messaging system more difficult.

3.8.3 Implementation

The considerations in the previous sections suggest that a simpler messaging layer can reduce complexity and better suit the requirements of the DAQ system.

The first choice made in the implementation was the use of the TCP transport layer. This choice implies relinquishing a certain degree of control over the injection of messages in the network to the operating system. However, given the requirement for reliable, in-order message delivery and given that messages can have sizes of up to several megabytes, the alternative of using UDP is even less appealing. Using a transport layer that does not make delivery guarantees would have required implementing data fragmentation and retransmission in the applications themselves. That is, re-implementing huge parts of the TCP functionality in the applications.

As TCP is a stream-oriented protocol, a form of framing is necessary to separate the messages in the stream. This is achieved by simply pre-pending each message with the three 32-bit fields shown in figure 3.15. The first two fields can be freely used by the applications. Conventionally, the first field (*message type identifier*) is used to identify the class of message being sent, so that the receiving application can correctly parse it. The second field (*stream identifier*) can optionally be used for multiplexing: messages with different stream identifiers belong to logically separate data streams. The third field is essential for the framing to work: it carries the size of the message following the header, so that a receiver can read the correct amount of bytes from the TCP stream.

This lightweight protocol built on top of TCP is sufficient to satisfy the requirement of reliable, in-order point-to-point message delivery and leaves the choice of messaging pattern up to the applications.

Clearly the protocol design alone is not sufficient to satisfy the requirements outlined in section 3.8.1. The implementation needs to reach the requisite performance. As mentioned, all applications in the system must manage a relatively high number of connections with a low message rate per connection (e.g. a ROS manages around 2000 connections from the DCMs, but on average sends just 25 Hz of 20 kB messages on each connection). Such a scenario is quite common in Web servers. Generally, the accepted solution is to employ the *non-blocking I/O* facilities provided by the operating system with the *event-driven* programming paradigm. Handling multiple connections with the common thread-based paradigm requires having one thread per connection, waiting for blocking I/O operations on that connection to complete. With the event-driven approach, instead, one or a few threads react to *events* delivered by the operating system, such as the completion of an I/O operation.

Several existing software libraries facilitate interfacing an event-based application with the necessary operating system facilities. The DAQ messaging layer, called AsyncMsg, is based on the widely used and stable Boost.ASIO C++ library. AsyncMsg leaves the management of the memory for messages under the complete control of the application. Both Boost.ASIO and AsyncMsg are designed to avoid unnecessary data copies, which are particularly detrimental in high-bandwidth applications, and unnecessary memory allocations, which constitute a bottleneck for high-message-rate applications.

3.8.4 Benchmarks

A series of measurements under controlled conditions was performed in order to demonstrate that the AsyncMsg library implementation can satisfy the performance requirement of the ATLAS DAQ system. The most important goal is to verify the scalability of the implementation, in terms of message rate, bandwidth, and number of connections. Two representative benchmarks are shown here. The benchmarks try to stress the AsyncMsg implementation with a messaging pattern that is supposed to mimic the usage that a real DAQ application would make of the library. The ones shown here represent two polar opposites in terms of messaging patterns:

- the HLTSV–DCM messaging pattern, which, as explained in section 3.8.1 is characterised by small, frequent messages;
- the DCM–Data Logger messaging pattern, which is characterised by big, infrequent messages.

Since these tests are aimed at evaluating AsyncMsg only, they were carried out using mock implementations of the HLTSV, DCM, and Data Logger applications, with the aim of introducing as little overhead as possible. A single AsyncMsg server was tasked with handling a varying amount of connections from AsyncMsg clients configured to to mimic the messaging patterns in the first and third lines of table 3.1. The server was running Scientific Linux 6.7. It was equipped with two Intel Xeon E5645 processor running at 2.40 GHz, 8 GB of RAM, and four 1 Gb/s Ethernet network interfaces. This limits the maximum throughput to 500 MB/s, not considering the overhead. The AsyncMsg server was configured to use as many threads as the CPU hardware threads (24).

The results, for the HLSTV–DCM messaging pattern, are presented in figure 3.16. The clients were configured to send 10 B messages at a fixed rate of 50 Hz, to which the server replied with a 1 kB payload. The rate of request and responses handled by the server was measured at 5 s intervals. The average of 20 measurements was then used. As shown in the figure, the rate scales linearly with the number of clients. The rate is also remarkably stable: the relative standard deviation was well under 1‰ for all measurements. The only deviation from perfect scalability is observed at 9000 client connections. However, that is caused by the bandwidth limitation rather than by the server hitting a bottleneck: 9000 clients correspond to a payload bandwidth of 450 MB. Taking the AsyncMsg, TCP, IP and Etherent overheads in account, that throughput roughly corresponds to the saturation of the network links. The good performance comes at a price in terms of CPU usage. As shown in the second plot in figure 3.16, the server uses as 78% of the available CPU cycles. However, one must take into account that this synthetic benchmark pushes the AsyncMsg library well beyond the requirements: the operating point for the HLTSV–DCM communication is at 100 kHz, where the server has the much lower CPU usage of 35%. The memory usage is proportional to the number of connections, and in this case it is generally negligible. Even with this difficult workload, the AsyncMsg library leaves abundant processing power for the application logic.

The results for the DCM–Data Logger messaging pattern are shown in figure 3.17. In this case, the clients were configured to send 1 MB messages at a fixed rate of 0.2 Hz, to which the server replied with a 10 B payload. Also in this scenario the server scalability is perfect until the bandwidth saturation is reached. As expected, with bigger messages and a lower message rate, the server does not need as many CPU cycles as in the previous scenario. Indeed, even at the network saturation, the server handles the traffic with a CPU utilisation of 15%. With big messages, the memory usage is more significant, but not worrying. Indeed the server needs at least 1 MB per connection to receive the mes-

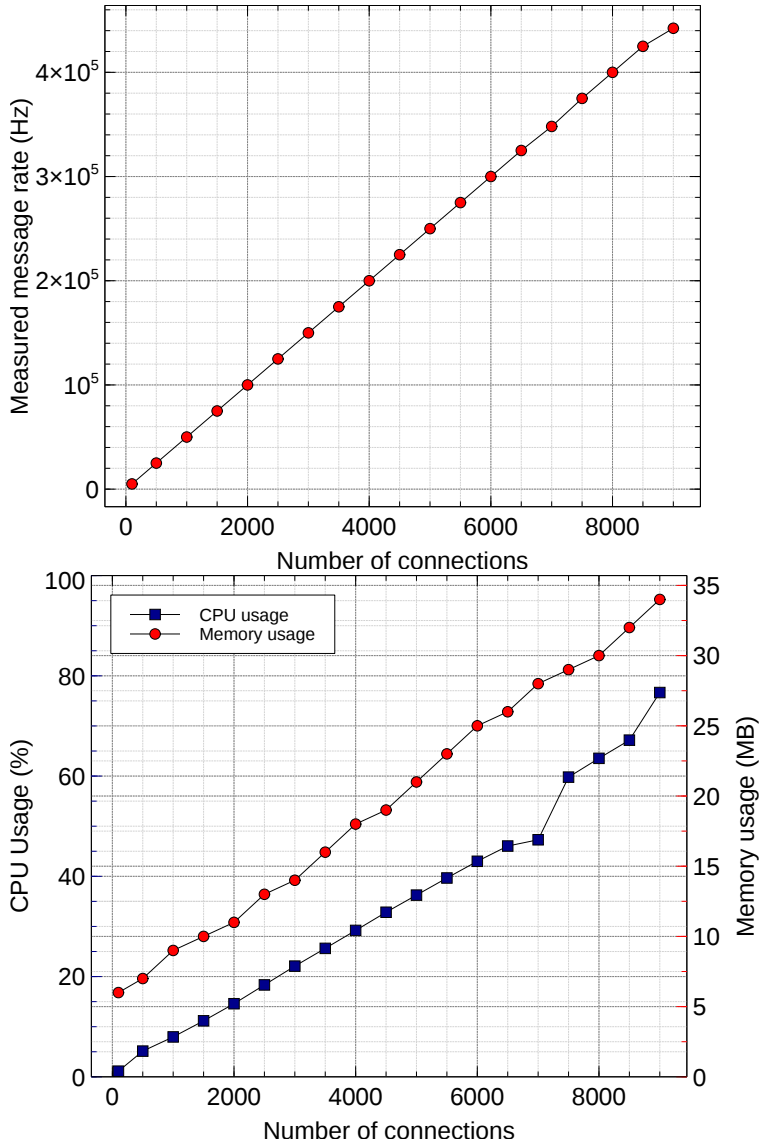


Figure 3.16 Benchmark results for the HLTSV–DCM messaging pattern.

sage. Therefore it is not surprising that the memory usage is roughly 1 MB multiplied by the number of connections.

The measurements presented prove that the AsyncMsg library is suitable for use as the messaging layer of the ATLAS DAQ system, and that it will not represent a bottleneck for future increases in performance requirements.

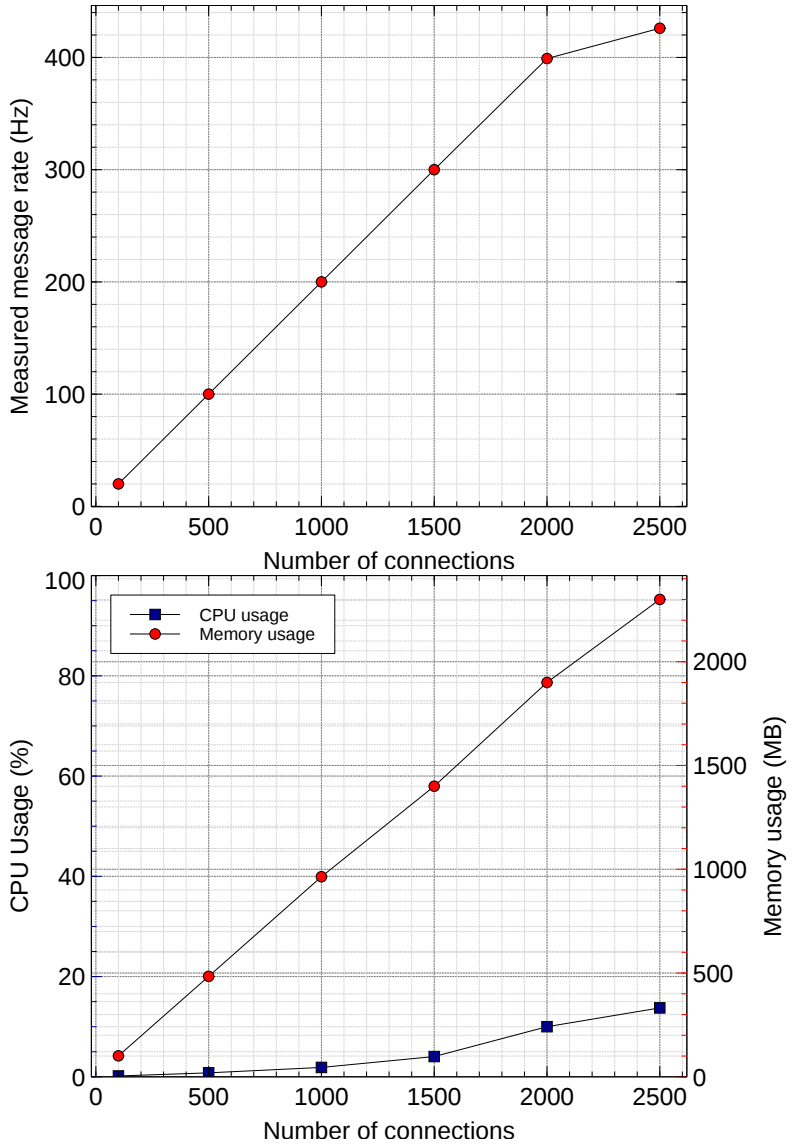


Figure 3.17 Benchmark results for the DCM–Data Logger messaging pattern.

Static traffic shaping for current data-acquisition systems

4

The effects of the incast traffic pattern described in general terms in chapter 2 are shown here in action, using measurements performed on the current data-acquisition system of the ATLAS experiment, introduced in the previous chapter. The measurements quantify the impact of incast on the system's performance metrics in with different network loads and with two different switch buffer layouts. A simple application-layer traffic-shaping algorithm is shown to be effective in mitigating the incast effect and thus increasing the network usage efficiency.

4.1 Performance issues in data-acquisition networks

As explained in section 2.1.1, the throughput of a data-acquisition system is crucial to the performance of the experiment as a whole. If the throughput is insufficient, interesting and valuable experimental data is lost. Worse, if the data losses are caused by the experiment generating more data because it is observing a particular phenomenon, a bias in the acquired data set is introduced. Depending on the nature of the experiment this might make all of the acquired data untrustworthy.

Data-acquisition throughput essentially depends on two quantities: the data-processing and data-collection times. The data-processing time depends on the efficiency of algorithms specific to the experiment and there is no generic approach to reducing it. The data-collection time, instead, is largely determined by the effectiveness of the data-acquisition infrastructure.

The typical data-acquisition traffic pattern is the bursty, many-to-one communication described in section 2.1.2.

In the case of ATLAS, each of the HLT processing units operates exclusively on the single event assigned to it, with the HLT selection proceeding iteratively starting from the regions-of-interest identified by the Level-1, collecting data fragments incrementally as needed. A processing unit is blocked while it waits for the data fragments to be collected, which means that the data-collection latency effectively translates to wasted CPU time. ATLAS event data are striped over all the Readout Systems. As explained in section 3.5 and section 3.7.2, each ROS is connected with point-to-point links (ROLs) with RODs pushing data from a specific region of a specific sub-detector. On the other hand, a single HLT processing unit analysing an event normally requests fragments from multiple regions and multiple sub-detectors at the same time. Since the fragments are already buffered there, the ROS response to a fragment request is almost immediate. If multiple requests for fragments reach different Readout Systems at roughly the same time (as is the case when fragments from multiple ROS nodes are requested together), many nodes will start sending them at the same time to the same destination, thus creating instantaneous network congestion. In the ATLAS data-acquisition network this can affect both the core, where packets with the same destination will arrive at roughly the same time from multiple ROS links, and the rack switches, where a burst of packets arriving from the high speed link from the core needs to be sent over a slower link to the destination HLT node (see figure 4.1). This instantaneous congestion can lead to occurrences of the incast pathology, when all the packets in some of the bursts are discarded due to buffer overflows. As discussed in section 2.3, the discarded packets will only be retransmitted after their TCP retransmission timeout (RTO) expires. Given that the RTO has a default minimum of 200 ms on Linux, and that the average per-event processing time in ATLAS is of the order of 100 ms, waiting for just one TCP RTO can potentially triple the time it takes to collect and process an event!

4.2 Evaluation of the impact of the incast pathology on data-acquisition performance

The following sections present a series of measurements which demonstrate the impact of the incast pathology in a data-acquisition network.

Detecting packet drops in an Ethernet network is a relatively simple task: most switch

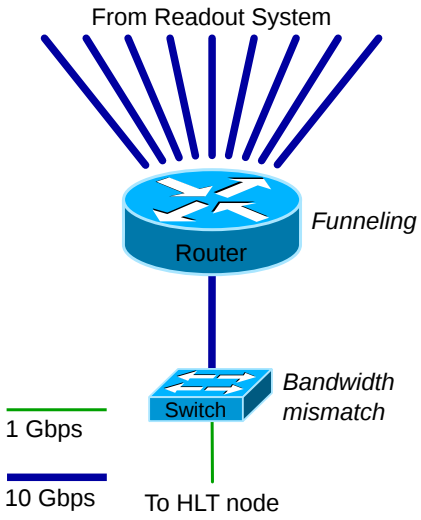


Figure 4.1 Visualization of the potential network congestion issues.

models publish cumulative counts of dropped packets per port via SNMP and the Linux kernel provides the total number of packet retransmissions that have occurred in a TCP connection. However, in real-world scenarios, the relationship between these counts and the overall performance of the system under study is rather complex, making it difficult to analyse the problem in detail.

A more fruitful approach is to generate synthetic traffic patterns, which can be tightly controlled and are known in advance. This way, it is possible to disregard packet drop counts and instead study the consequences of incast on the main performance metric of interest: the data-collection time.

4.2.1 Measurement set-up

The measurements were performed using some of the hardware available in the ATLAS Data-Acquisition system. At the time of these tests, the infrastructure was still under consolidation, so the network layout was slightly different from the one described in section 3.7.6. In particular, Readout Systems were not directly connected with 10 Gb/s Ethernet links to the core routers. Instead, they were connected with 1 Gb/s Ethernet links to an intermediate aggregation switch with a 10 Gb/s Ethernet uplink to each core. The ROS uplinks were kept under-subscribed so that no congestion could appear in the aggregation switch.

Given these constraints, the following test set-up, shown in Figure 4.2, was chosen:

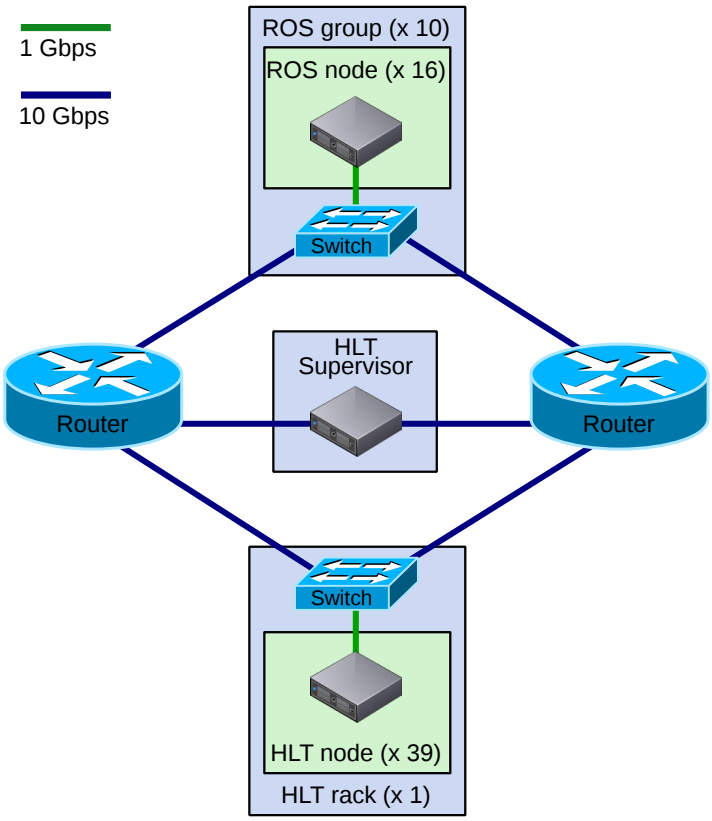


Figure 4.2 Measurement set-up.

- 10 Readout System groups:
  - Each group consists of 16 ROS nodes connected to one switch (total: 160).
  - 12 event fragments with a constant size of 1.1 kB are served by each node (total fragments: 1920, full event size: 2112 kB).
- 1 HLT rack:
  - It consists of 39 PCs connected to one switch.
  - Each PC hosts 24 HLT processing units (936 total).

This configuration was chosen because it provides a realistic model of the expected network buffer usage in the final system topology. The fact that just one HLT rack is in use instead of the ~50 racks of the complete system does not prejudice the usefulness of this set-up: since Ethernet drops packets in case of network congestion, no head-of-line blocking is possible and congestion does not propagate from switch to switch. Therefore, in output-queued switches, flows directed to different output ports do not affect each other. The obtained results should scale reliably with the higher number of HLT racks in the complete system.

All PCs were running Scientific Linux 6.5. They are equipped with two Intel Xeon X5650 processors, 24 GB of RAM, and two Intel 82576 Ethernet network interfaces. The operating system's network stack was kept in its default configuration.

The core routers are Brocade MLXe-32 [15] modular switches with 10 Gb/s line cards. They use input-buffering with a peculiar implementation of switch-level virtual output queuing: input ports are grouped in modules of 8 ports at most and each module maintains multiple, distinct queues to every output port on the router. As these routers are geared towards Internet Service Providers, they are equipped with very deep buffers: each module has a packet memory of 1.5 GB.

Different models of top-of-rack switches were tested. For clarity the results using two representative models are presented here:

- Switch A (HP ProCurve 6600-48G-4XG [33]): a switch with 48 1 Gb/s ports and four 10 Gb/s ports. Each 1 Gb/s port has an output queue limited to 786 kB. Each 10 Gb/s port has an output queue limited to 4.8 MB.
- Switch B (Brocade VDX 6740T-1G [16]): a switch with 48 1 Gb/s ports and two 40 Gb/s ports. The first 32 1 Gb/s share an input buffer of 12.6 MB. The remaining 16 1 Gb/s ports and the two 40 Gb/s ports share an input buffer of 12.6 MB. Each port

also has an input queue limit of up to 8 MB and an output queue limit of up to 8 MB.

Switch A represents the class of switches with a fixed amount of buffer space dedicated to each port. Switch B represents the class of switches with a shared memory pool from which each port can "borrow" buffer space as needed. The bulk of the data-collection traffic enters the switch from the two 10 Gb/s uplinks from the core routers and exits from the 1 Gb/s ports connected to a PC in the rack. As a consequence, for Switch A, the parameter that limits the system's performance is the 786 kB limit on the output queues of the 1 Gb/s ports. For switch B, instead, the limiting parameter is the 12.6 MB input buffer shared by the 40 Gb/s ports. The switches also reserve a portion of the buffers for internal meta-data and for quality-of-service purposes. Therefore, the actual buffer space available for normal priority packets is lower than the limits quoted above.

In these measurements, the synthetic traffic pattern used is as follows:

- Events are assigned by the HLT supervisor to HLT processing units at a configurable constant rate.
- The processing units immediately collect a configurable fraction of the event's fragments, process them for a fixed amount of time (100 ms in these measurements), and ask for a new assignment.

The total data-collection throughput is therefore given by

$$throughput = HLTSV\ rate \times DC\ size$$

where the size of each data collection is

$$DC\ size = collected\ fraction \times event\ size$$

By varying the number of collected fragments and adjusting the HLTSV rate so as to keep their product unchanged, one can vary the traffic pattern while keeping the throughput constant.

For both top-of-rack switch models two scenarios were tested:

- Medium throughput: the $HLTSV\ rate \times collected\ fraction$ product was chosen to give a data-collection throughput of around 1.06 GB/s (8.45 Gb/s). Including the overhead, the rack uplinks utilisation was 45%.
- High throughput: the $HLTSV\ rate \times collected\ fraction$ product was chosen to give

a data-collection throughput of around 2.11 GB/s (16.9 Gb/s). Including the overhead, the rack uplinks utilisation was 90%.

4.2.2 Results

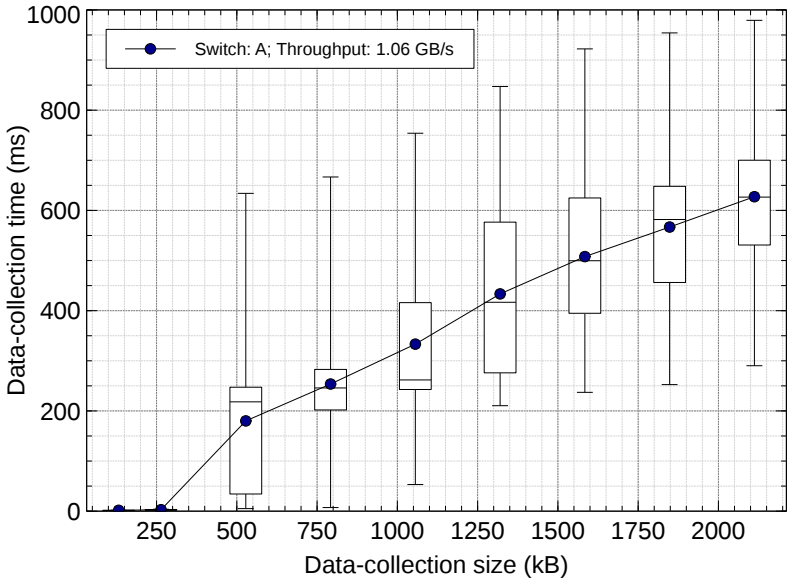
The effect of the varying traffic patterns on the data-collection time is shown in figure 4.3 for switch A and figure 4.4 for switch B.

In ideal conditions, the data-collection time would be determined by the HLTPU–ROS round-trip time (RTT) and the data transmission delay:

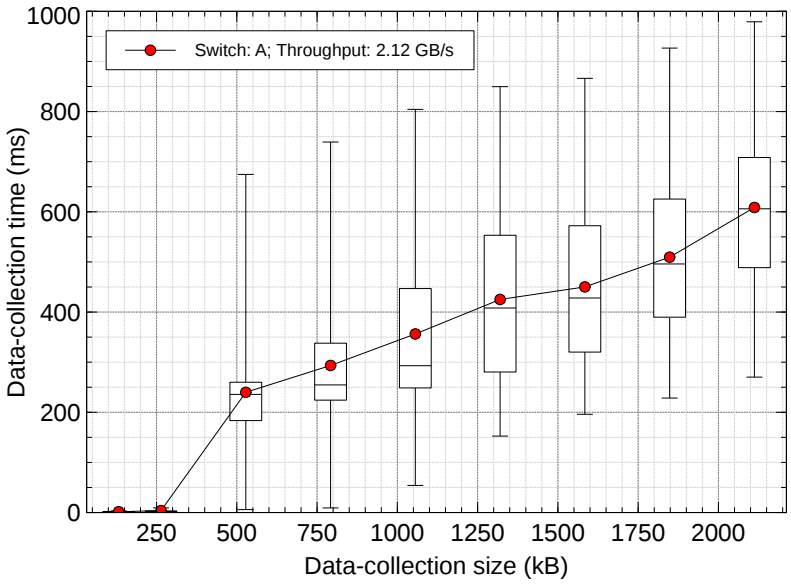
$$DC\,time = RTT + \frac{DC\,size}{bandwidth}$$

The RTT here is the sum of two components: the actual node-to-node RTT, and the ROS application response time. In the test system this sum is 400 µs on average. For accuracy, the data-collection size used here must take into account the overhead of the TCP, IP, and Ethernet encapsulations, shown in section 2.2.2 to be about 5.3% of the application-level data-collection size. Given that the bandwidth of the slowest link in the path is 1 Gb/s, the ideal data-collection time should be between 1.5 ms for the smallest data-collection size and 18.2 ms for collecting the full event. The measurements however reveal that these predictions are only valid for small data-collection sizes. Over a certain size threshold, that depends on the amount of traffic and on the switch model being used, the actual data-collection time is bigger than the ideal time by more than one order of magnitude. This is caused by the incast phenomenon: the Readout Systems inject fragments in bursts. The size of the burst is obviously the data-collection size. In this test system, the huge buffers in the core routers are sufficient to absorb any kind of congestion that might arise from having multiple inputs sending to a single output port in the router. In the HLT top-of-rack switches, however, buffering space is more limited. The data bursts enter the switch at a combined 20 Gb/s, but they exit through a single 1 Gb/s port connected to the requesting HLT node. Once the switch buffers are full, packets corresponding to entire transmissions from a ROS will be all dropped, thus triggering the incast pathology as explained in section 2.3. This effect is clearly shown in figure 4.3a and figure 4.3b.

In switch A, each output port has its own dedicated buffer. If the burst size is larger than the buffer, the incast pathology will always be triggered. Indeed, the onset point of incast does not depend on the total amount of traffic going through the switch, as shown by the similarity between figure 4.3a and figure 4.3b.



(a)



(b)

Figure 4.3 Data-collection time as a function of the data-collection size, using switch A, at a constant bandwidth of (a) 1.06 GB/s and (b) 2.12 GB/s. The test conditions are detailed in section 4.2.1. The bullets represent the average values. The horizontal box lines represent the first quartile, the median, and the third quartile. The box whiskers represent the first and the 99th percentile.

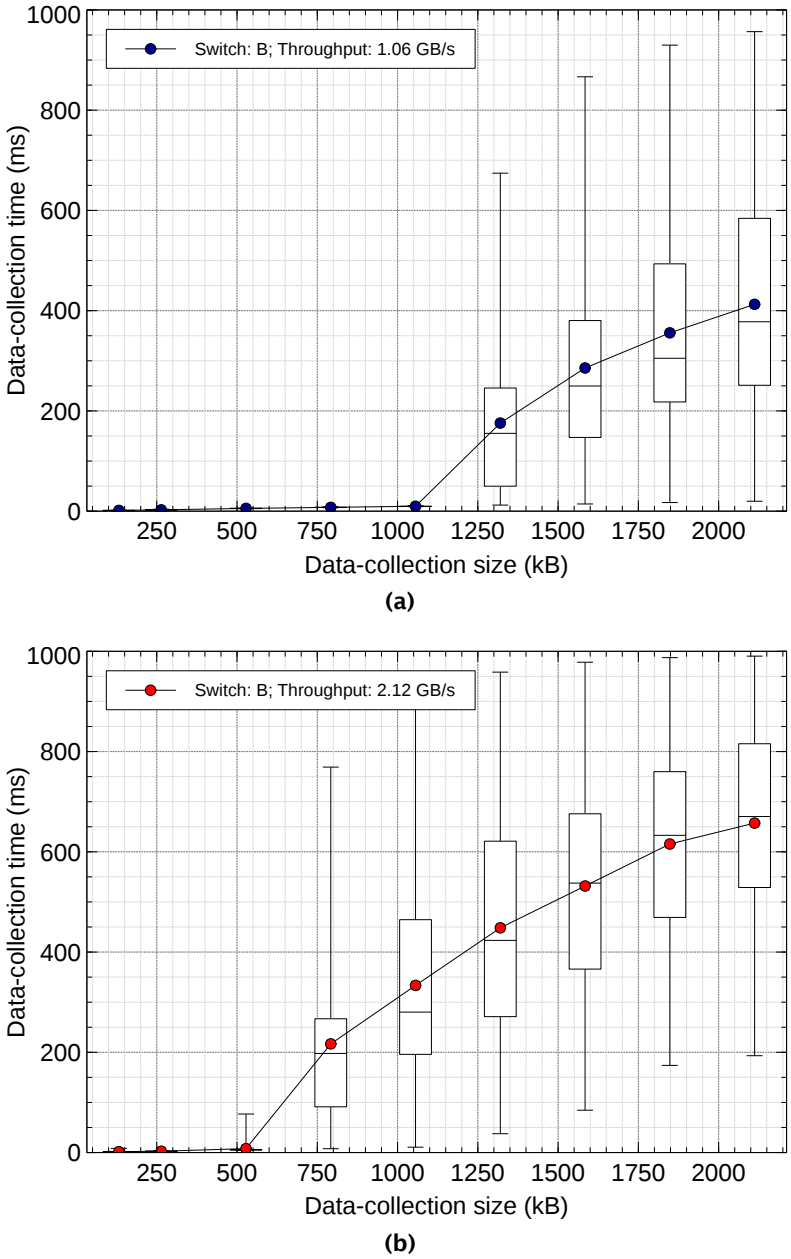
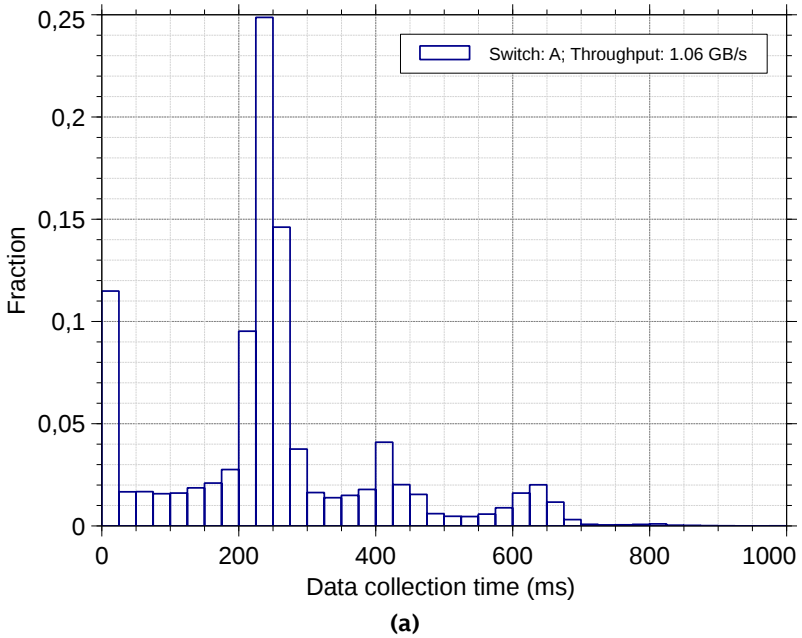
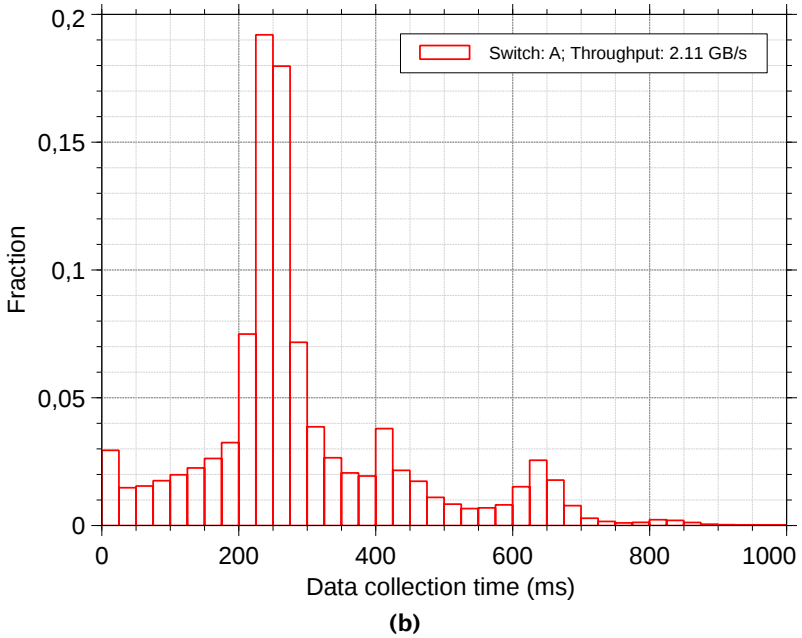


Figure 4.4 Data-collection time as a function of the data-collection size, using switch B, at a constant bandwidth of (a) 1.06 GB/s and (b) 2.12 GB/s. The test conditions are detailed in section 4.2.1. The bullets represent the average values. The horizontal box lines represent the first quartile, the median, and the third quartile. The box whiskers represent the first and the 99th percentile.

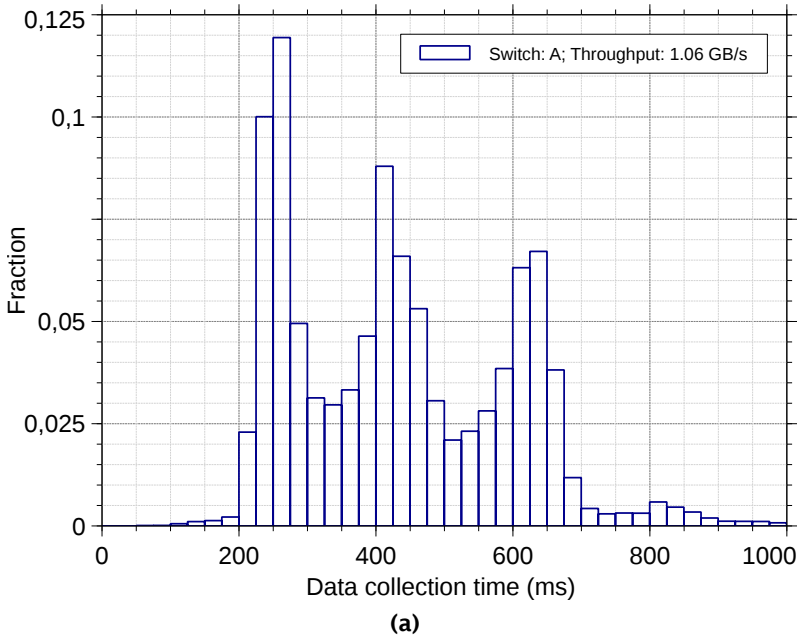


(a)

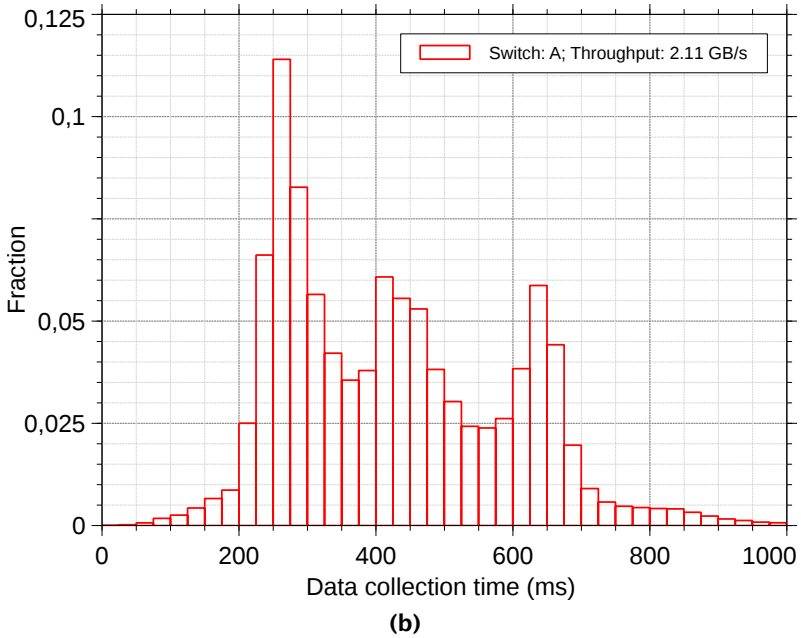


(b)

Figure 4.5 Distribution of data-collection times for a data-collection size of 792 kB, using switch A, at a constant bandwidth of (a) 1.06 GB/s and (b) 2.12 GB/s.

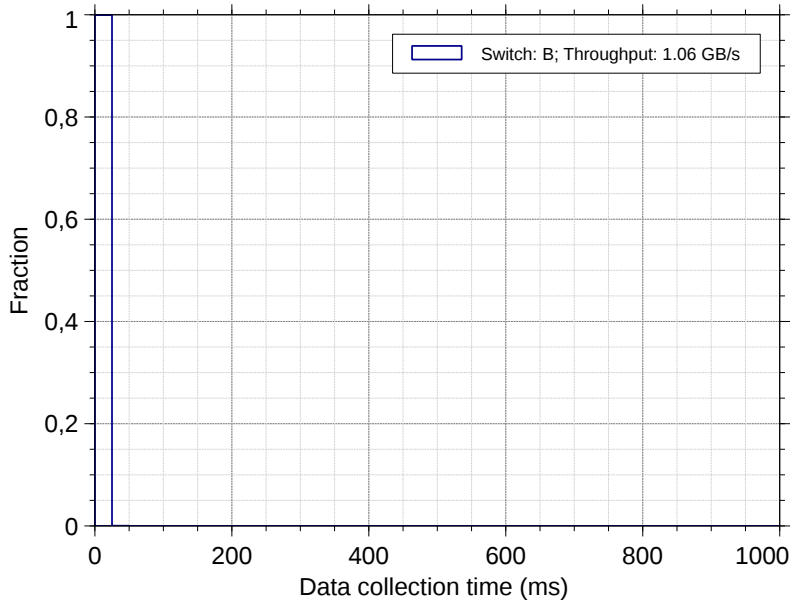


(a)

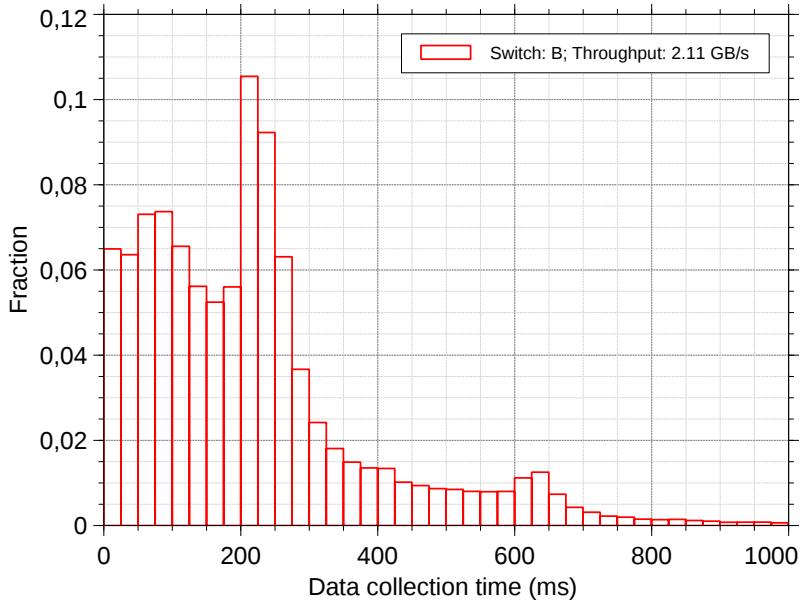


(b)

Figure 4.6 Distribution of data-collection times for a data-collection size of 1320 kB, using switch A, at a constant bandwidth of (a) 1.06 GB/s and (b) 2.12 GB/s.

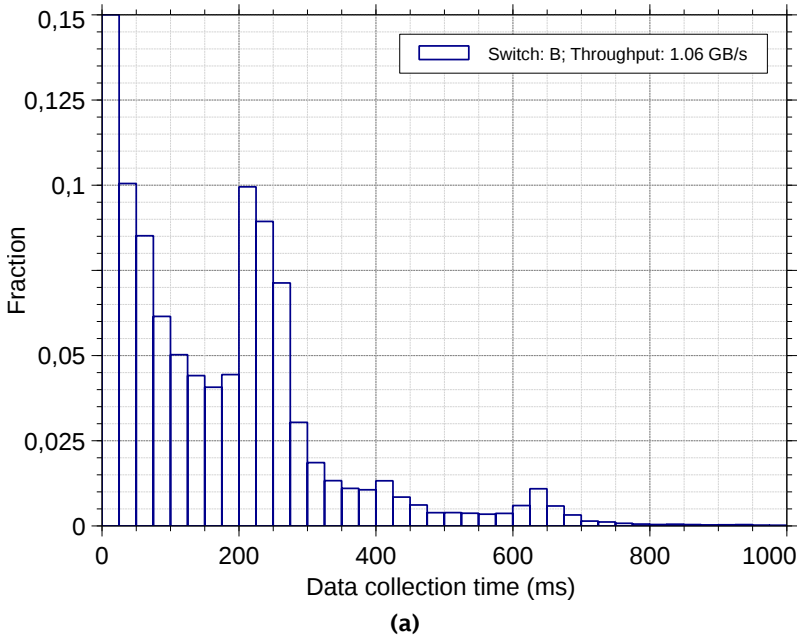


(a)

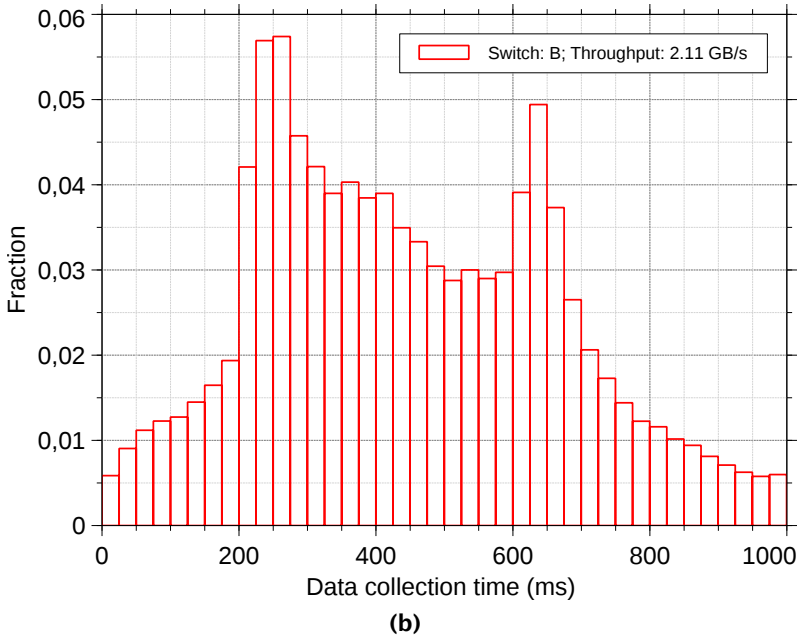


(b)

Figure 4.7 Distribution of data-collection times for a data-collection size of 792 kB, using switch B, at a constant bandwidth of (a) 1.06 GB/s and (b) 2.12 GB/s.



(a)



(b)

Figure 4.8 Distribution of data-collection times for a data-collection size of 1320 kB, using switch B, at a constant bandwidth of (a) 1.06 GB/s and (b) 2.12 GB/s.

In switch B, instead, the buffer is one order of magnitude larger than the maximum burst size. However, it is shared among many output ports. The average buffer occupancy grows with the amount of traffic passing through the switch. As a consequence, the burst size that the buffer can handle shrinks, explaining the difference between figure 4.4a and figure 4.4b: at lower throughput, bigger bursts are necessary to cause incast to appear. At higher throughput, the incast onset point corresponds to a significantly lower data-collection size.

The effect of incast can be clearly seen examining the distributions of the data-collection times at fixed data-collection sizes. Figures 4.5 and 4.6 show the data-collection times measured using switch A for a data-collection size of 792 kB and 1320 kB respectively. Not coincidentally, the three peaks in the distributions roughly correspond to 200, 400 and 600 ms, i.e. to multiples of the minimum TCP retransmission time-out on Linux. With a data-collection size of 792 kB, most data-collections are affected by at least one retransmission. With a bigger data-collection burst of 1320 kB, a non-insignificant fraction of the data-collections is affected by multiple retransmissions. This means that the first batch of retransmissions is itself subject to enough packet drops to trigger incast again. In some cases even the second and third retransmissions are affected. Figures 4.7 and 4.8 show the same data measured using switch B. In this case, when the throughput and data-collection size are low enough, no data-collections are affected by retransmissions. However, increasing the throughput (and hence the average occupancy of the shared switch buffer) or the data-collection size causes enough packets to be dropped and the incast pathology appears.

4.3 Request-side traffic shaping

The incast problem is well studied in literature, with many different solutions being proposed (see section 2.4). In general the solutions focus on preventing the senders from sending too many packets onto a congested path, so that the switches will not have to resort to dropping an entire burst of packets. This is usually achieved with special hardware support (e.g. switches with hardware flow control or explicit congestion notification) possibly in conjunction with alterations of the TCP implementation. Alternatively, solutions tailored to specific traffic patterns, such as the one described here, can be developed, with the goal of saving on hardware cost and complexity.

4.3.1 Incast mitigation

A prime example of a solution to the incast problem that is easily applicable to data-acquisition systems is application-level traffic shaping. Incast arises from the fact that multiple sources send data at the same time to a destination, without coordinating among themselves. In a data-acquisition system however, the destination is normally aware of the fact that it is requesting data from multiple sources together. By spreading the requests over time, it can therefore limit the number of simultaneous data bursts in response. Obviously such a mechanism imposes a trade-off: excessive spreading of the requests can increase the data-collection time by unnecessarily delaying the requests for data, whereas insufficient smoothing will not eliminate packet drops.

In the specific case of ATLAS, the Data-Collection Manager application oversees the traffic in and out of every worker node. No other program on the node has access to the data network. Therefore, the DCM, thanks to its privileged position, can effectively limit the maximum amount of concurrent data requests to the Readout System. This is implemented with a credit-based traffic shaping algorithm [24]. Its basic rules are as follows.

- Each DCM has a fixed number of credits available. All the HLT processing units on its node share these credits.
- Each data request from a processing unit to a ROS consumes as many credits as the number of data fragments it asks for.
- Each response from a ROS returns the credits used by the corresponding request.
- If all available credits are used, further requests are blocked until the necessary credits become available.

The number of fragments in a request gives a rough estimation of the size of the corresponding response. Therefore, this algorithm effectively limits the maximum burst size of data transfers directed to the same HLT node. However, it relies on the assumptions that all event fragments are similar in size, and that this size is known beforehand. These assumptions are reasonable for ATLAS under normal operating conditions, but not necessarily for other systems or scenarios. However, the algorithm can easily be extended to cover more use cases.

4.3.2 Effectiveness evaluation

The effectiveness of the traffic shaping algorithm outlined in the previous section was evaluated using the test set-up described in section 4.2.1. The same two different top-of-rack switches were used. The results presented here correspond to the following synthetic traffic pattern:

- Events are assigned by the HLT supervisor to HLT processing units at a configurable constant rate.
- The processing units immediately collect all of the event's fragments, process them for a fixed amount of time, and ask for a new assignment.

In other terms this is the traffic pattern that corresponds to the data points in section 4.2.1 with the largest data-collection size. Obviously this pattern is the harshest in terms of generated traffic bursts: since the data fragments corresponding to an event are collected all at once, the maximum burst size is equal to the event size. On the other hand, the fixed-size fragments enable the simple traffic-shaping algorithm to operate without inefficiencies caused by it lacking knowledge of the expected size of the responses. In analogy with the tests section 4.2.1, many different throughput levels were tested: the results reported here correspond to HLT supervisor event assignment rates of 500 Hz and 1000 Hz, giving a rack uplink utilisation of 45% and 90% respectively.

As explained in the previous section, by varying the amount of traffic shaping credits assigned to the Data-Collection Managers, one can control the maximum size of the burst of data sent by the Readout Systems. The results of such a scan are shown in figure 4.9 for switch A and figure 4.10 for switch B. The total data-collection time per event is influenced both by the network conditions and by the traffic-shaping mechanism. With too few traffic-shaping credits available (i.e. with a small maximum allowed burst size), the data-collection time is larger due to data-collection inefficiency: the HLT nodes cannot fully utilise the network bandwidth. This is analogous to a TCP connection with a congestion window smaller than the bandwidth-delay product. With too many traffic-shaping credits the algorithm fails to prevent overflowing the buffers of the top-of-rack switch, triggering the TCP incast pathology.

Using top-of-rack switch A the minimum data-collection latency is found with a traffic shaping limiting the maximum burst size to 528 kB (not including overhead). As expected, this value is close to the switch's maximum output buffer size of 786 kB and exceeding it triggers incast, regardless of the total throughput handled by the rack. Since

top-of-rack switch B is instead equipped with a shared buffer, the optimal traffic shaping configuration is not so clearly defined, as it depends on the total throughput as well.

With fixed-size fragments, the traffic-shaping algorithm effectively behaves in the same way as a window-based receiver-side congestion control. As such, its optimal window size can be determined as follows:

$$\frac{window}{bandwidth} = RTT + \frac{block}{bandwidth}$$

where *bandwidth* is the data rate of the slowest link in the path, *RTT* is the application-to-application round-trip time, and *block* is the smallest amount of data that is transmitted. In this case, given that each ROS serves 12 fragments of 1.1 kB each and that the traffic-shaping algorithm does not fragment requests to a ROS, *block* is equal to 13.2 kB. As already mentioned, the measured average RTT in the test system is 400 µs and the bottleneck link runs at 1 Gb/s, yielding an optimal window size of 88.2 kB. The measurements, however, paint a different picture: the optimal working point corresponds to a much larger window. This is most likely caused by a suboptimal event assignment policy. Due to performance constraints, the HLT supervisor assigns events to HLT processing units on a first-come first-served (FCFS) basis, i.e. events are assigned to processing units in order of arrival of their assignment requests. It is therefore possible that multiple processing units running on the same node request data at the same time, thus competing for the same pool of traffic-shaping credits. This is further confirmed by the observation that the data-collection time increases with the throughput: higher throughput means higher assignment rate, which in turn increases the probability of consecutive event assignments to the same node. Indeed, as shown in figure 4.11 and figure 4.12, the data-collection time can only reach its minimum at extremely low throughput, with HLT nodes essentially idle.

In conclusion, while the traffic-shaping algorithm is certainly effective in mitigating the effects of the incast pathology, it does not allow to recover the full ideal performance of the system. For that to happen, further improvements are necessary.

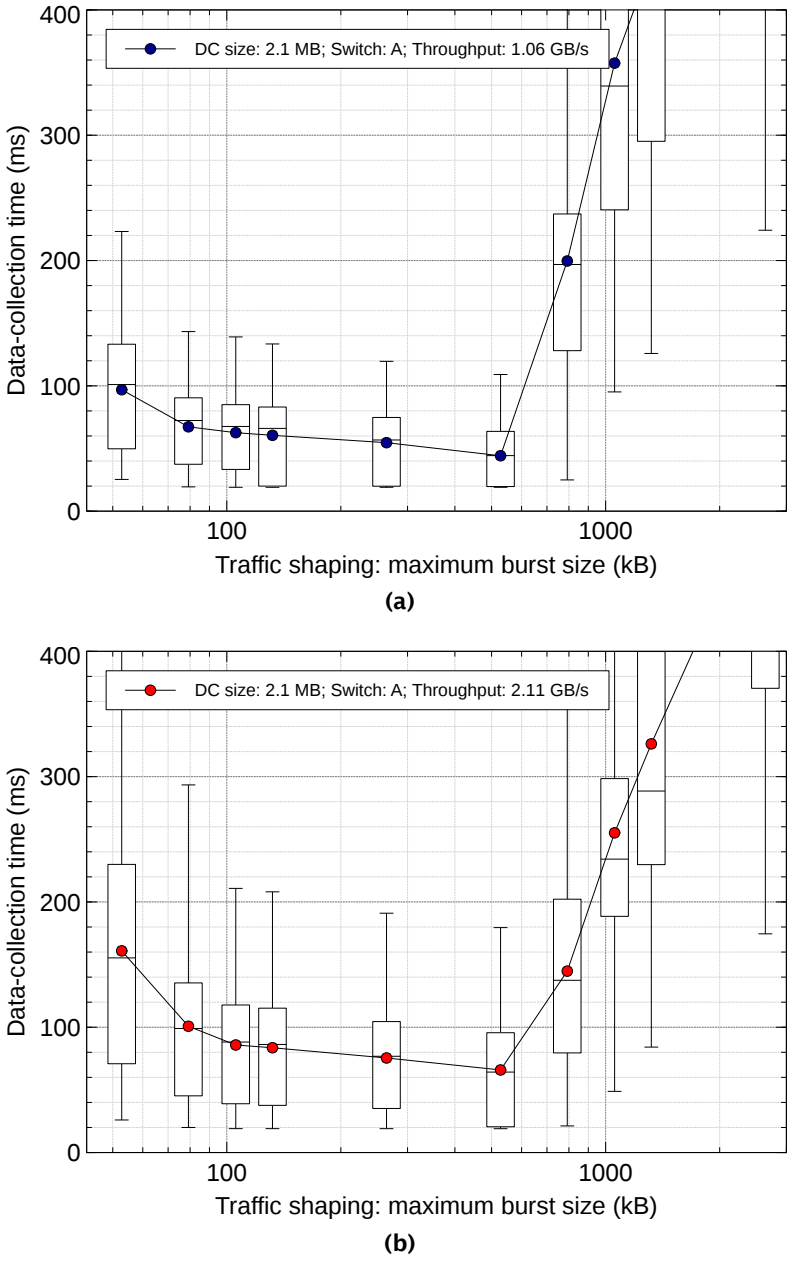


Figure 4.9 Data-collection time as a function of the maximum burst size allowed by the traffic shaping algorithm described in section 4.3.1, using switch A, at a constant bandwidth of (a) 1.06 GB/s and (b) 2.12 GB/s. The bullets represent the average values. The horizontal box lines represent the first quartile, the median, and the third quartile. The box whiskers represent the first and the 99th percentile.

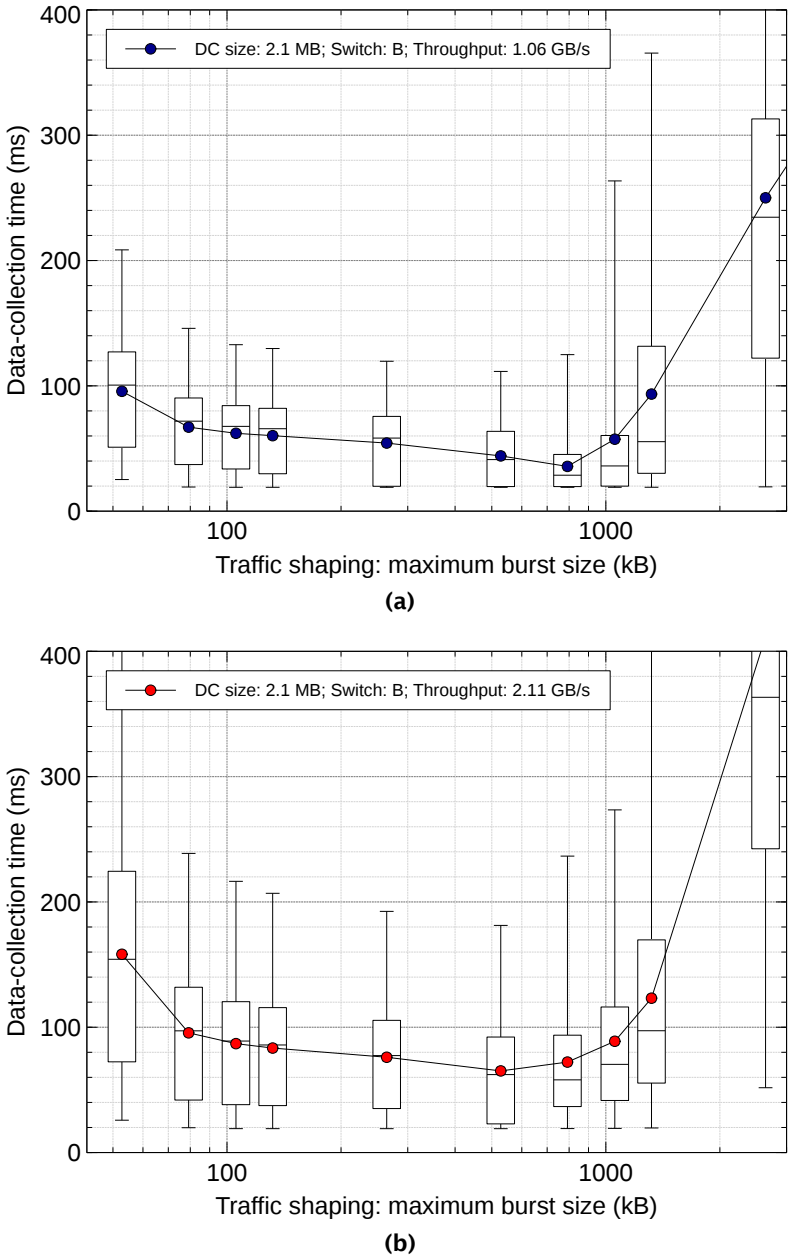


Figure 4.10 Data-collection time as a function of the maximum burst size allowed by the traffic shaping algorithm described in section 4.3.1, using switch B, at a constant bandwidth of (a) 1.06 GB/s and (b) 2.12 GB/s. The bullets represent the average values. The horizontal box lines represent the first quartile, the median, and the third quartile. The box whiskers represent the first and the 99th percentile.

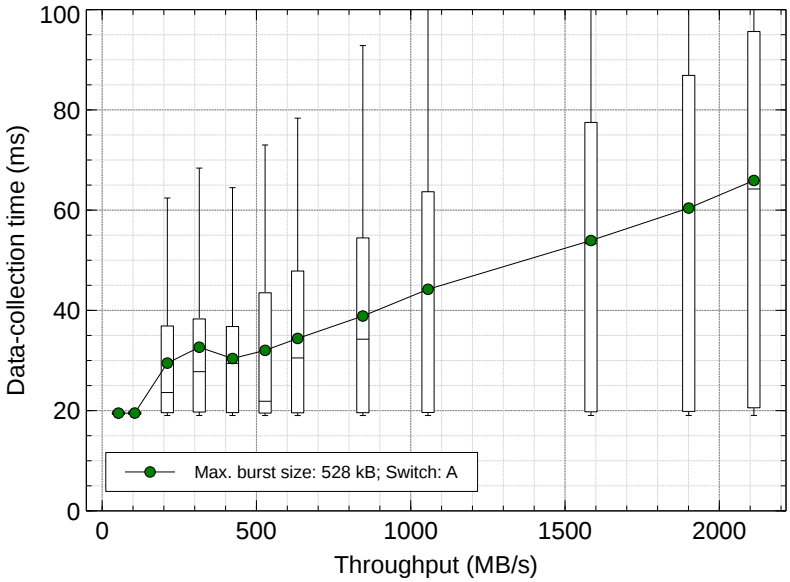


Figure 4.11 Data-collection time as a function of the rack throughput, using switch A, with a maximum burst size limited by traffic shaping to 528 kB.

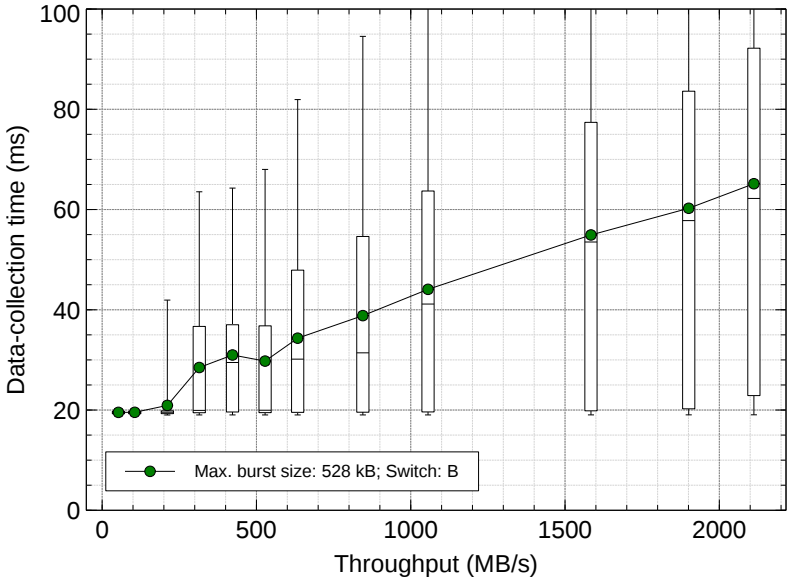


Figure 4.12 Data-collection time as a function of the rack throughput, using switch B, with a maximum burst size limited by traffic shaping to 528 kB.

5 Simulation model

The kind of measurements presented in chapter 4 can reveal a great deal of information on the behaviour of the data-acquisition system. However, they are not a practical method for further research. In general, data-acquisition systems are mission-critical components of scientific experiments. Therefore, a systematic study of their performance envelope is often impeded by operational constraints: once the design of the systems is finalised, the systems are rarely available for performance measurements and the opportunities of performing hardware or software modifications are limited.

A simulation model can instead be a worthwhile alternative, assuming that it is accurate enough to reliably reproduce the key traits of the system. The following sections describe a model that aims at reproducing the measurements shown in chapter 4. After establishing its accuracy, the simulation can be used to explore various factors influencing the performance of the data collection.

5.1 Model development

The simulation presented in this chapter is based on the OMNeT++ discrete event simulation[1] framework [64]. OMNeT++ is free for non-commercial use and its source code is available. It was chosen for two main reasons: its wide acceptance in the academic community and its ease of use for the purpose of modelling different types of networks, such as computer networks, wireless networks, sensor networks, interconnection networks of High-Performance Computing (HPC) systems, or interconnection networks supporting massive-storage or big-data systems [36, 41, 19, 68, 47].

[1] In this particular sentence, the word "event" refers to simulation events. To avoid confusion, throughout the rest of the thesis, "event" will only be used as in chapter 3, i.e., to mean "collision event".

In OMNeT++ simulations are composed of modules, defined in the declarative NED language, which communicate exchanging messages via module-to-module channels. So-called simple modules are the active components of the simulation and are implemented in C++, leveraging the class hierarchy provided by the simulation framework. Modules can be grouped together to form compound modules and networks. The development of network simulations is aided by built-in support for physical channels, with latency, transmission delay, and message loss properties.

The model described here makes use of the INET Framework for OMNeT++ [36]. INET is a library of protocol models which includes detailed implementations of all network layers, from the data-link layer upwards. In particular, its models of the Ethernet and IP protocols are employed.

5.1.1 Hosts

Data-acquisition application models are implemented on top of the standard INET host model which includes a complete network stack, from the transport layer down to the physical layer, as shown for example in figure 5.1.

The transport layer model, i.e. the TCP model, has obviously the biggest influence on the overall accuracy of the simulation. It is unfortunately also the most complex. As a consequence, similarly to other protocol model libraries, only simplified versions of the "traditional" TCP variants (Tahoe, Reno, Vegas and New Reno) are implemented natively in INET [59]. A useful feature of INET's TCP models is that they support a data transfer mode that preserves application-level message boundaries: for example, if an application sends a 10 kB message, the receiver application will receive the same message, after TCP has completely finished simulating the transmission of 10 kB over the connection. This greatly simplifies the modelling of message-based applications, like those in the ATLAS data-acquisition software. Unfortunately, INET's TCP implementations are not free of bugs and, more importantly, no implementation of the TCP flavour actually used in Linux, TCP CUBIC [32], is available. However, INET can integrate with the Network Simulation Cradle (NSC) [39], a wrapper for the network stacks of real-world operating systems. It is thus possible to use a real-world TCP implementation in the network simulator. In particular, NSC supports the TCP stack from Linux 2.6.26, which is used in this simulation. NSC does not preserve application-level message boundaries: if an application sends a message that is bigger than TCP's maximum segment size, the receiver application will not receive a single message, but rather multiple TCP segments.

It is then up to the receiver application to re-assemble the original message. Moreover, the higher accuracy of NSC comes at the price of significantly higher computational cost. Therefore, INET's simplified TCP models were used during the development of the simulation. They were then replaced by NSC as the model matured.

The network layer model is composed by INET's fairly complete implementations of the IP, ICMP and ARP protocols. This simulation does not require multicast transmission or advanced routing, so just a small part of the network-layer models is actually needed.

The data-link layer model is subdivided, following the Ethernet standard, into an upper layer that handles the encapsulation of packets into Ethernet frames, and a lower media access control (MAC) layer that schedules the transmission of data on the physical link. Given that all the links in the simulation are point-to-point and full-duplex, the MAC layer complexity can be kept at a minimum.

No performance model for the network stack is implemented. Packet processing times are assumed to be small enough that no significant queuing effects appear. The application models are instead charged with including some additional delay in their response times to simulate the host latency. This is a justifiable simplification: all the hosts in the real systems are powerful enough to handle a throughput of several Gb/s without performance limitations.

5.1.2 Applications

In order to simulate the test system presented in section 4.2.1, four applications need to be modelled: the HLT supervisor, the Readout System application, the Data-Collection Manager, and the HLT processing unit.

In the real system, the HLT supervisor, which assigns events to processing units, and the Readout System applications, which serve event data fragments, are data-driven applications: their behaviour is dependent on what phenomena the experiment is measuring and what events the Level-1 trigger is selecting. In principle, this would require models of those applications to follow traces recorded on the real system. However, this is not necessary when trying to reproduce the synthetic traffic patterns described in section 4.2.1. The Readout System becomes a trivial server application, responding to fragment requests with configurable delay and response size. The supervisor instead is reduced to a periodic scheduler. The processing units send a message to the supervisor when they are available, i.e. when they are ready to start processing another event. The

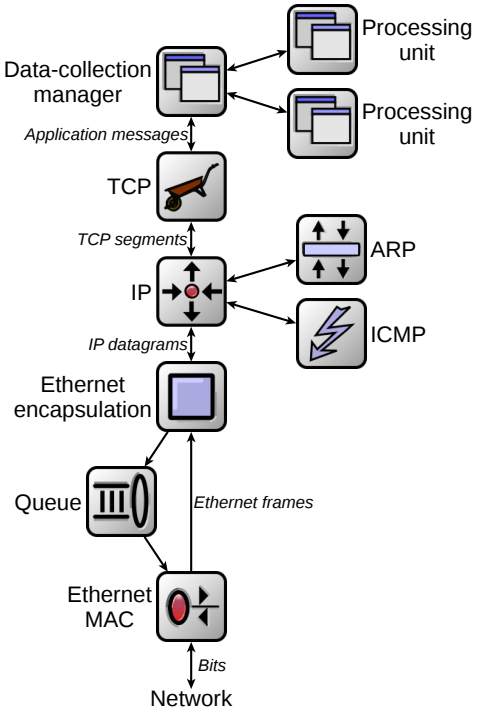


Figure 5.1 Example of a model a host: HLT working node with 2 processing units.

supervisor stores this information and uses it to assign events to processing units at a configurable global rate, by sending assignment messages.

OMNeT++ models are implemented using the C++ programming language. Since the same language is used throughout the ATLAS data-acquisition software, the application-level code that is relevant to the simulation model can be ported to the simulation environment with minimal changes. This ensures a bug-for-bug compatible reproduction of the applications' behaviour within the model. This approach is used for modelling the processing units, which generate the data requests for an event, and the per-node data-collection managers, which act as proxies between the node's processing units and the readout systems. In particular, a processing unit can simulate per-event iterative collection and processing of data: after it receives an event assignment, it can request data from the Readout System with a configurable pattern, emulate processing by waiting for a configurable amount of time, and repeat this process several times before considering the event fully processed and asking for another one from the supervisor. Just like in the real system, the processing units running on a HLT worker node do not interface directly with the TCP module: their communications are mediated by the Data-Collection Manager. Its most relevant functions in the context of this model are: the mapping of

each data request from the HLT processing units to messages to multiple ROS nodes and the enforcement of the traffic-shaping algorithm described in section 4.3.1.

5.1.3 Network switches

For the purposes of this simulation, the most relevant aspects of the network hardware are packet buffering and queuing. The internal architecture of the switches is not modelled in detail. It is assumed that the switch speedup is high enough to prevent input head-of-line blocking, and to make the packetisation delay of the switch cells negligible. This is obviously an oversimplification with respect to the actual architecture of a switch. Unfortunately, Ethernet switch manufacturers divulge very few details on the actual architecture of their products, so a more sophisticated model would involve a huge amount of guesswork. Nevertheless, some mitigating factors reduce the impact of this oversimplification. First of all, the system being modelled does not make use of quality-of-service features: all packets have the same priority. As a consequence, a detailed model of the switch arbitration mechanism would not have a significant effect on the accuracy of the switch model. Moreover, a realistic model of the switch latency would not significantly impact the overall simulation results: the most important performance metric, the data-collection time, is dominated by the delays caused by packet drops and retransmissions, which are several orders of magnitude larger than the typical switch latency.

With the above assumptions, switches are modelled as follows. For each switch port, an INET Ethernet MAC module acts as the interface between the physical transmission channel and the switch. It sends incoming frames to an ideal frame relay unit, which maintains the switch's MAC address table and instantaneously forwards frames towards their destination port (or broadcasts them if the destination is not yet present in the address table). One or more "packet droppers" intercept frames between the relay unit and the switch output queues. These modules decide whether to forward or drop a frame, depending on the total size of the packets stored in the queues that are connected to their outputs, effectively defining the switch buffering scheme. Frames that are not dropped are stored in the switch's output queues, waiting for the Ethernet MAC to pull them from the queue when the transmission channel is ready.

Two basic buffering schemes are considered: dedicated and shared. In the dedicated buffers model, shown in figure 5.2a, there is a dropper for every output port, effectively modelling tail-drop output queues. In the shared buffer model, shown in figure 5.2b, a

single dropper guards all the output ports, so packets are dropped when the the total size of the queued packets reaches a set threshold. The two basic schemes can naturally be extended to model more complex architectures, e.g. with buffer space limits both on a per-switch basis and on a per-port basis, or with different groups of ports using different buffer pools.

5.1.4 Complete model

The components described in the previous sections are assembled to create a model of the test system described in section 4.2.1, with one significant difference. As explained section 3.7.6, in the real system the two core routers are in an active-active redundant configuration, with traffic equally distributed over the two in normal operating conditions. This is possible because the core routers share some internal information, such as forwarding tables, using a vendor-specific protocol, which is not easily reproducible in the switch model described above. However, a closer look at the implementation of this configuration in the real system enables to reproduce it in the simulation without additional effort. As already mentioned in section 3.7.6, hosts and switches connected to both core routers use a hash of the packets' data-link and network addresses to choose to which core router the packets are sent. Therefore, the traffic between a given ROS host and a given HLT worker node will always flow through the same core router. If the configuration is balanced, roughly half of the ROS hosts will send data to the HLT worker nodes via the first core router, while the other half will communicate via the second router. Thus, the real system configuration can be approximated by splitting each ROS group of 16 hosts into two groups of 8 hosts, with the first group connected to just the first core router, and the second group connected to just the second core router. Figure 5.3 shows the complete model.

5.1.5 Parameters

The model parameters are chosen with the goal of reproducing the measurements presented in chapter 4. Most parameters can simply be set to the same values of their counterparts in the real system, as described in section 4.2.1. These include:

- topology of nodes and network links (see figure 5.3),
- event fragment size (1.1 kB),
- number of event fragments (1920, each ROS node serves 12 fragments),

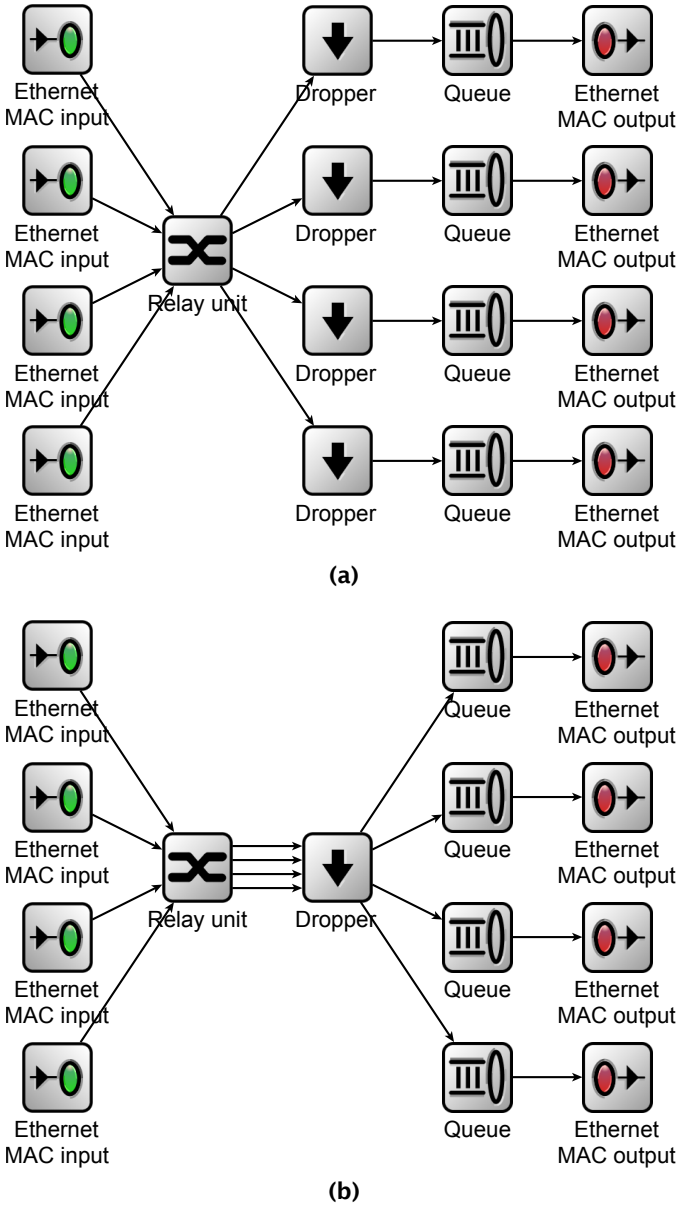


Figure 5.2 Models of output buffered switches with (a) dedicated per-port buffers and (b) shared buffers.

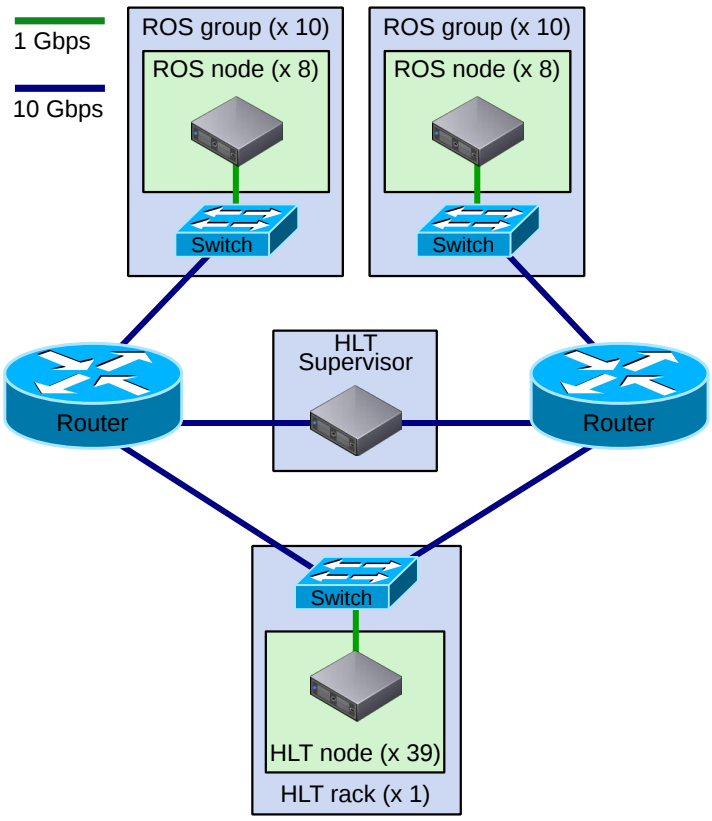


Figure 5.3 Layout of the simulated system.

- HLT supervisor event assignment rate (500 Hz and 1000 Hz for the results shown here),
- HLT processing unit behaviour (full data collection followed by 100 ms of processing time),
- configuration of the DCM traffic shaping algorithm.

Other parameters cannot be chosen a priori, and must be measured, either directly or indirectly. These include:

- round-trip time between a data request sent by an HLT processing unit and the corresponding response from a ROS application,
- size limits of switch queues.

The round-trip time is the sum of many components: application latency, node network stack latency, and network latency, which cannot be easily measured independently. However, the total application-to-application round-trip time can be measured directly on an unloaded system. This results in an average time of 400 μs, with a standard deviation of 100 μs. The actual size limits of switch queues are harder to determine: while the total amount of packet buffer memory available on a switch is usually found in the switch's documentation, that figure alone is not sufficient to estimate the limits of packet queues. Normally, a fraction of the packet buffer is reserved for quality-of-service purposes (i.e. reserved for high-priority packets), so not all the advertised memory is actually usable by normal-priority packets. Moreover, the actual amount of memory used by each packet while stored in the switch is also unknown: the packets traverse the switch accompanied by some internal meta-data which is stored together with them in the packet buffer. Therefore, the size limits of switch queues must be determined indirectly, by analysing the switch behaviour under controlled conditions. The measurements presented in chapter 4 are in fact taken under controlled conditions, so this approach can be used without requiring additional ad-hoc measurements, as explained in section 5.2.2.

5.1.6 Runtime

The observed simulation run-times are roughly proportional to the number of generated packets, and hence on the simulated data-collection throughput. Empirically, the relation between simulated time t and simulation run-time T is approximately given by: $T = t \cdot r/R$, where r is the data-collection throughput, and R is a parameter that depends

on the CPU core running the simulation. As an example, $R \approx 2$ MB/s for an Intel Xeon E5645 processor, and $R \approx 1$ MB/s for an older Intel Xeon E5420 processor. This gives run-times between 4 and 8 hours for simulating a configuration with a data-collection bandwidth of ~2 GB/s for 30 s.

The component with the biggest impact on the run-time is the TCP model. Indeed, when using INET's simplified TCP models instead of the Linux TCP stack provided by NSC, the run-time is reduced by ~75%.

5.2 Model validation

5.2.1 Goals

The model outlined in section 5.1 has some obvious approximations, the most important ones being the simplified switch architecture and the lack of a performance model of the nodes. Before the simulation is used to draw conclusions on scenarios that cannot easily be tested in practice, it is necessary to validate the model against the known behaviour of the system.

As explained in section 2.1.1, the key application performance metric is the average data-collection time per event. The measurements described in section 4.2.2 demonstrate that the data-collection time is dramatically affected by the occurrence of the TCP incast pathology, so much so that, without countermeasures, the time spent transferring event data can become bigger than the time needed to process that event. The aim of the simulation model is to be a useful tool in exploring new ways to improve the performance of data-acquisition systems. Therefore, two features of the measurements need to be reproduced with particular accuracy:

- the impact on the data-collection times of the incast-avoidance mechanism (i.e. the traffic shaping algorithm described in section 4.3.1),
- the conditions that trigger the incast phenomenon.

The focus of this section is on comparing the measured data presented in section 4.3.2, which includes both features mentioned above, with the simulated results.

5.2.2 Analysis and comparison of measured and simulated results

The average measured and simulated data-collection times are shown in figure 5.4 for Switch A, and in figure 5.4 for Switch B. As usual, both the "medium throughput" (1.06 GB/s: 45% rack uplinks utilisation) and "high throughput" (2.11 GB/s: 90% rack uplink utilisation) scenarios are shown.

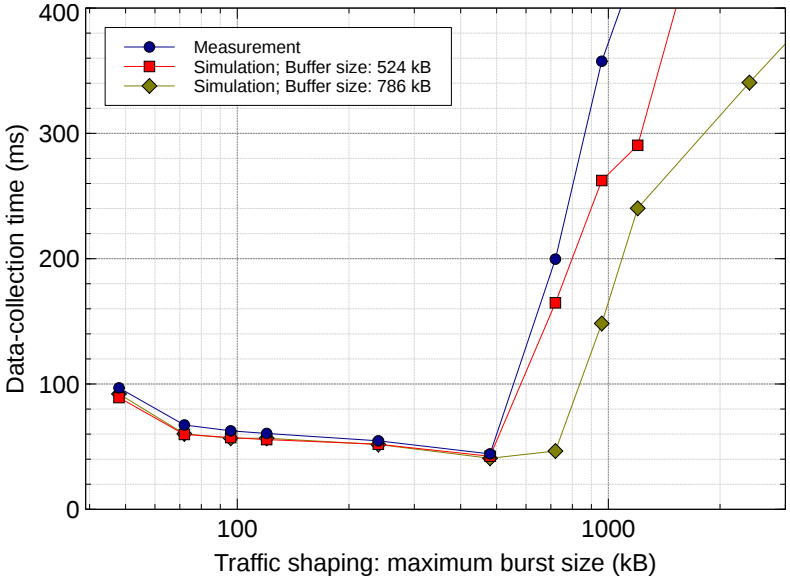
The model of the traffic-shaping algorithm benefits from the code sharing with its actual implementation. As a consequence, the simulation is particularly accurate in the region of the parameter space where the data-collection time is more heavily influenced by the traffic-shaping algorithm, i.e. for maximum burst sizes lower or equal to the value that minimises the data-collection latency.

The burst size threshold over which the incast pathology is triggered is obviously heavily dependent on the sizes of the switch buffers, which, as explained above, cannot be known with absolute certainty. Therefore, the results corresponding to two different simulated buffer sizes are shown[2]. The higher size corresponds to the manufacturer-specified buffer size, without taking into account that a fraction of it is used for purposes other than storing packets. The lower size corresponds to the value that yields the best reproduction of the distribution of the data-collection times when the traffic-shaping algorithm is completely disabled (i.e. when the data-collection time is dominated by the behaviour of the switches and of TCP). As expected, with the higher buffer sizes, the simulated incast thresholds are close to but larger than the measured ones. With the lower buffer sizes, instead, the incast thresholds are accurately reproduced.

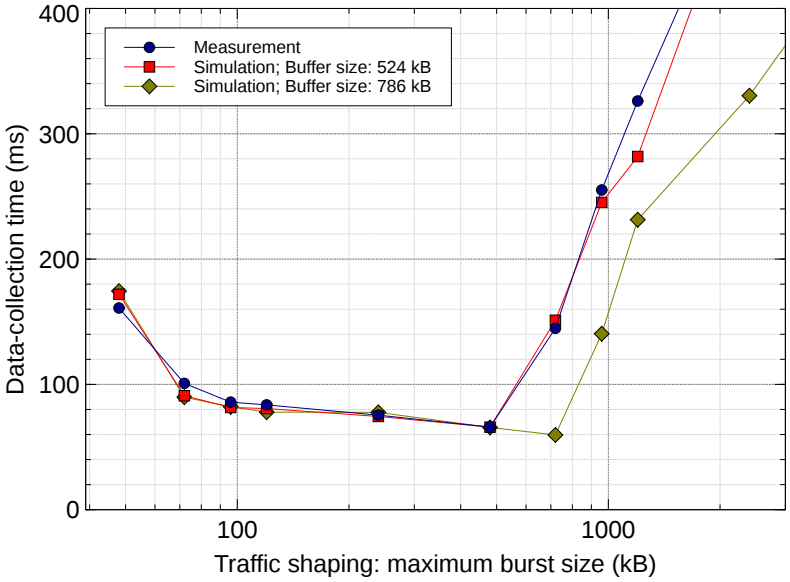
Beyond the incast threshold, the data-collection latency is determined by a complex interplay between the traffic shaping algorithm and the TCP retransmission mechanism. Unsurprisingly, the simulation loses accuracy in this region. Nevertheless, it still manages to reproduce the qualitative behaviour of the system. In general, reproducing the exact values of the data-collection time in this region is not particularly important: the interest of perfectly reproducing the pathological combination of ineffective traffic-shaping and TCP incast is limited. Correctly predicting the conditions under which this pathological combination occurs is a more valuable goal.

The data presented so far demonstrates that the simulation can reproduce the main performance metric of a data-acquisition system in terms of average values. The accuracy of the model can be further confirmed comparing the distributions of the data-collection

[2]Note that in the case of Switch A, the buffer size refers to the buffer on each individual port; in the case of Switch B, the buffer size refers to the total amount of memory shared by all switch ports.

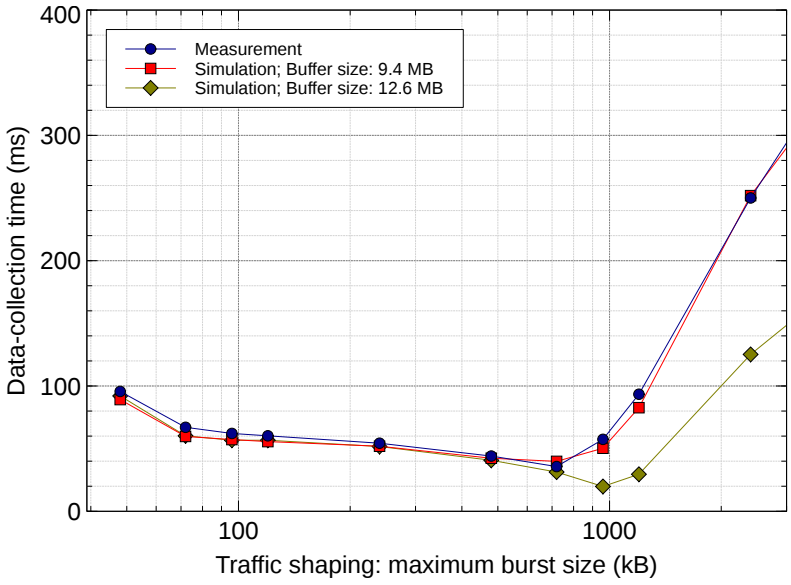


(a)

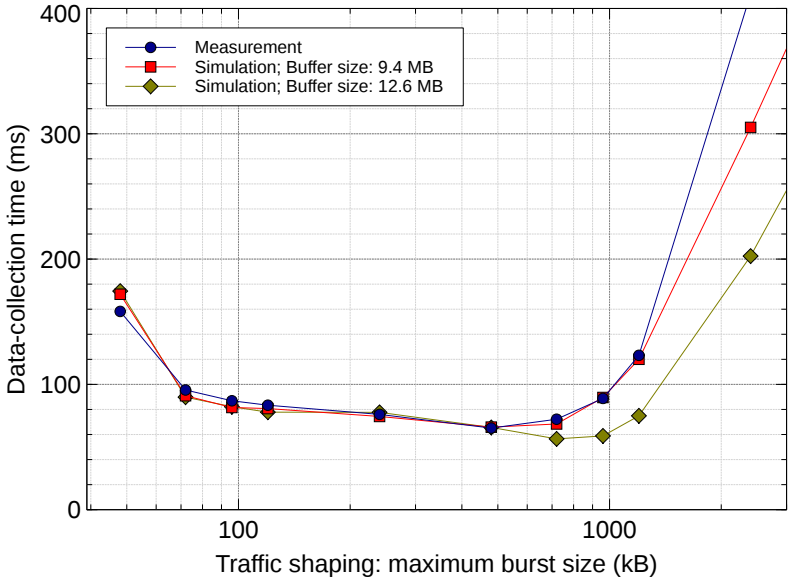


(b)

Figure 5.4 Comparison of measured and simulated average data-collection times as a function of the maximum burst size allowed by the traffic shaping algorithm described in section 4.3.1, using switch A, at a constant bandwidth of (a) 1.06 GB/s and (b) 2.12 GB/s.



(a)



(b)

Figure 5.5 Comparison of measured and simulated average data-collection times as a function of the maximum burst size allowed by the traffic shaping algorithm described in section 4.3.1, using switch B, at a constant bandwidth of (a) 1.06 GB/s and (b) 2.12 GB/s.

times, rather than just their mean values. For Switch A, this comparison is shown in figure 5.6 in the "medium throughput" scenario and in figure 5.7 in the "high throughput" scenario. The comparisons are shown for Switch B in figure 5.8 and figure 5.9. Three settings of the traffic-shaping algorithm are shown in each figure: with a very small maximum burst size (52.8 kB), with a maximum burst size just below the lowest incast threshold of all scenarios (528 kB), and with no traffic-shaping at all.

The simulation-generated distributions are generally in good agreement with the measured distributions, as long as the simulation is configured with the right switch buffer size. The histograms are useful to illustrate the effects of the traffic-shaping algorithm. In this scenario 2.1 MB events are sent to HLT worker nodes connected via a 1 Gb/s link. Therefore, as already shown in section 4.2.2, the minimum data-collection time is around 18.2 ms. When the traffic-shaping algorithm is too restrictive (see figures 5.6a, 5.7a, 5.8a, and 5.9a), almost none of the events are fully collected in less than 20 ms and, although a significant portion is collected within 40 ms, the distributions have long tails of events that take up to 300 ms (in the "medium" throughput scenario) or even 500 ms (in the "high" throughput scenario) to be fully collected. With a larger traffic-shaping window (see figures 5.6b, 5.7b, 5.8b, and 5.9b), a sizeable portion of the events is fully collected within 20 ms, which is compatible with the minimum calculated above, presumably because all their fragments were collected when no other events were competing for the same traffic-shaping credits. Other events, however, need to wait for enough credits to become available, therefore the distribution still presents a tail.

With traffic-shaping disabled, the distributions of data-collection times show the consequences of the incast pathology. In this case, when Switch A is in use (see figures 5.6c and 5.7c) no events are fully collected within the first 200 ms, which means that during most data-collections at least one data-transfer from a ROS suffers enough packet drops to cause a TCP retransmission time-out. This is expected, given that the event size, and hence the burst size, is 2.1 MB and the top-of-rack switch port output buffer is at best 786 kB. The majority of the events are collected between 200 ms and 700 ms, i.e. they incur between one and three retransmission time-outs, but a non-negligible fraction of them has an even higher data-collection time, with this fraction being higher for the higher throughput scenario. With the shared buffer in Switch B, the distributions are even more dependent on the total throughput. In the "medium" throughput scenario (see figure 5.8c), some events are collected without TCP retransmission time-outs, but the vast majority incurs one or more retransmission time-out. In the "high" throughput scenario (see figure 5.9c), the average data-collection increases noticeably, due to higher

occupancy of the shared buffer.

In the (a) and (b) histograms, the switch buffer size parameter has a limited impact on the distributions. This is expected, since in those scenarios the traffic-shaping algorithm is effective in ensuring that data bursts do not overrun the switch buffers, both with the lower and higher buffer sizes. The (c) histograms, instead, highlight the difference between a too optimistic estimation of the available buffer space and a correct one. For both switch models, the wrong choice of parameter produces a data-collection time distribution that is significantly different from the measured one.

All the comparisons discussed in this section indicate that, with the right choice of parameters, the simulation model presented in section 5.1 is accurate enough to reproduce the real-world behaviour of the system under study. The model can than be used to simulate scenarios that could not easily be enacted in practice.

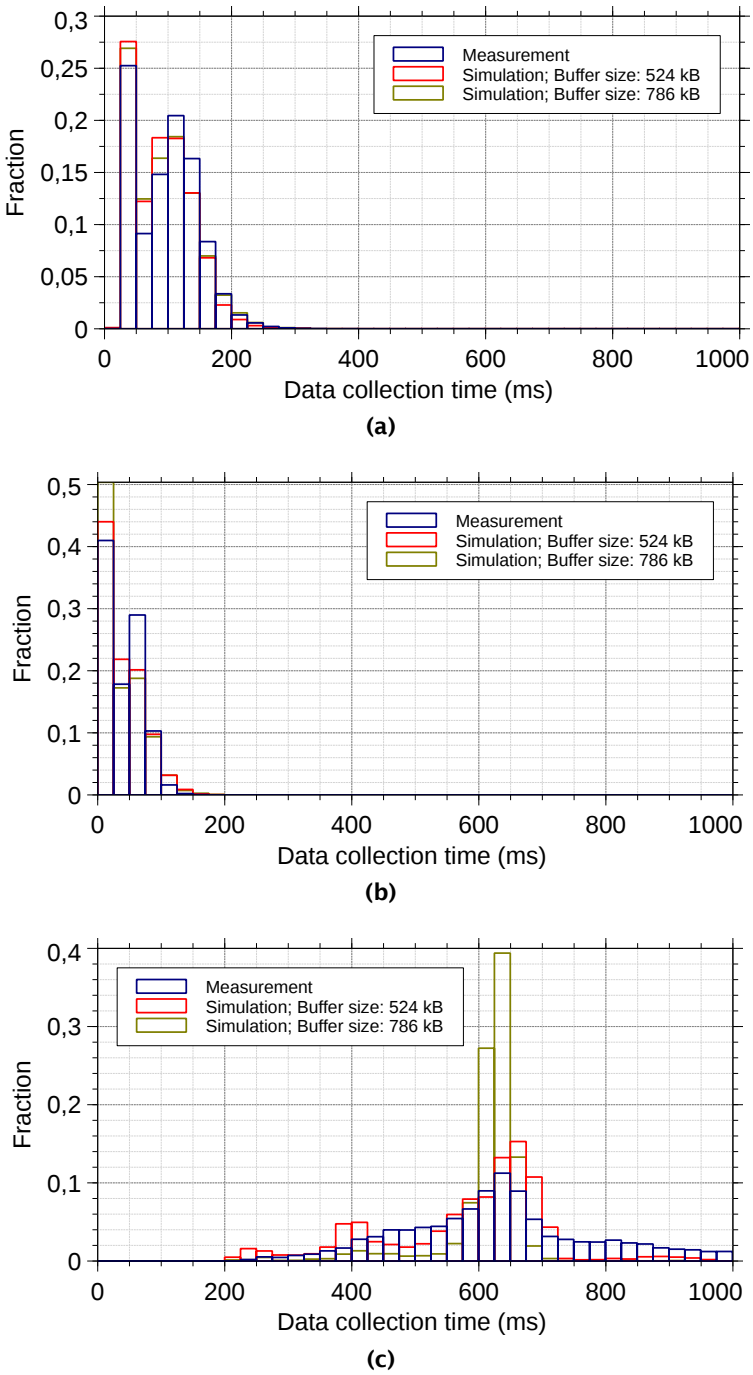


Figure 5.6 Distributions of measured and simulated data-collection times, using switch A, at a constant bandwidth of 1.06 GB, with the maximum burst size limited by the traffic shaping algorithm to (a) 52.8 kB, (b) 528 kB, (c) unlimited.

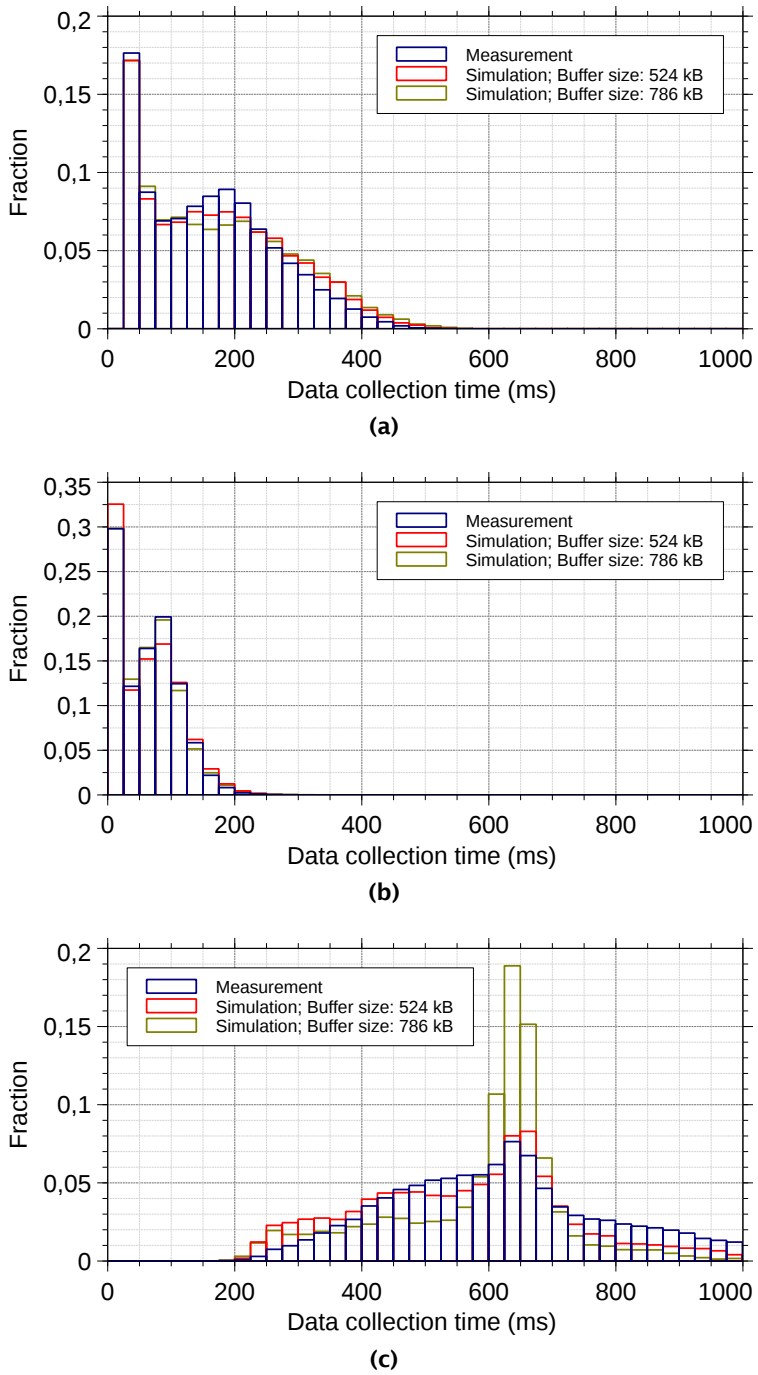


Figure 5.7 Distributions of measured and simulated data-collection times, using switch A, at a constant bandwidth of 2.06 GB, with the maximum burst size limited by the traffic shaping algorithm to (a) 52.8 kB, (b) 528 kB, (c) unlimited.

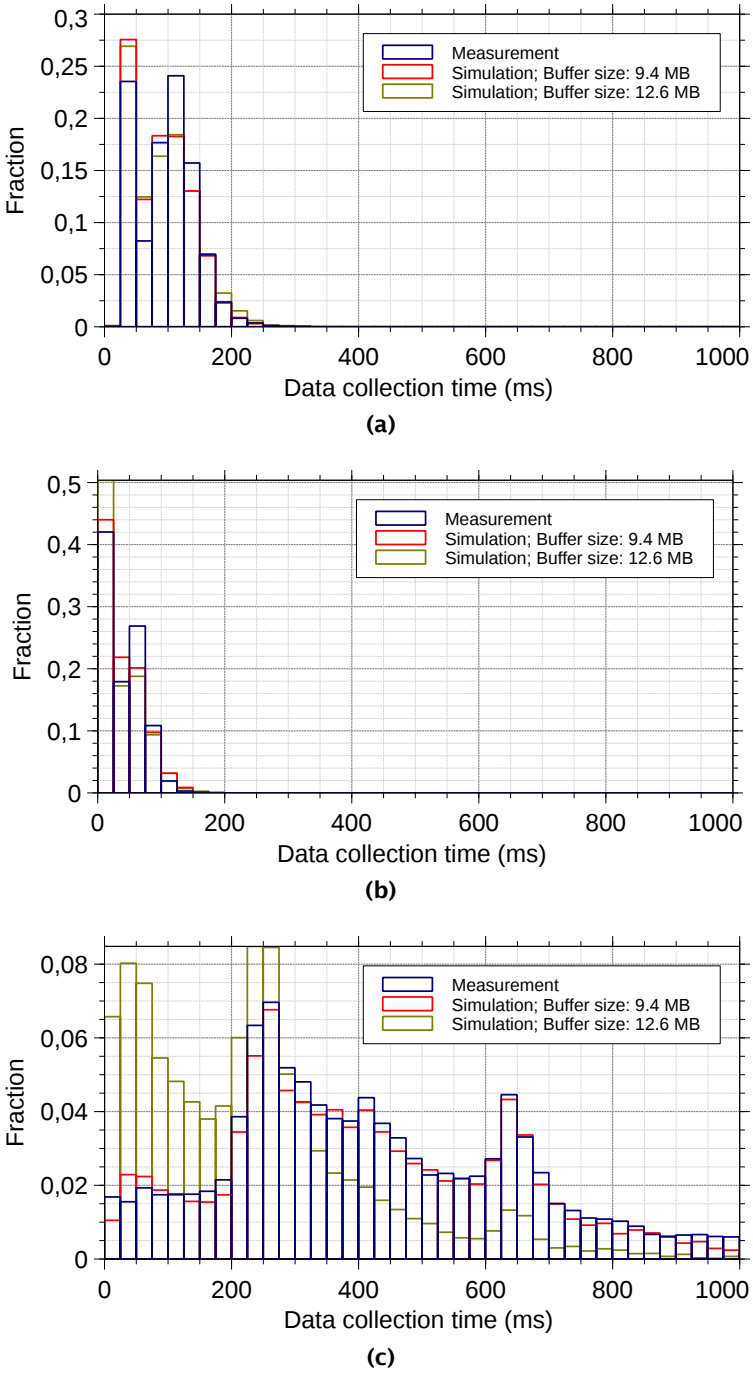
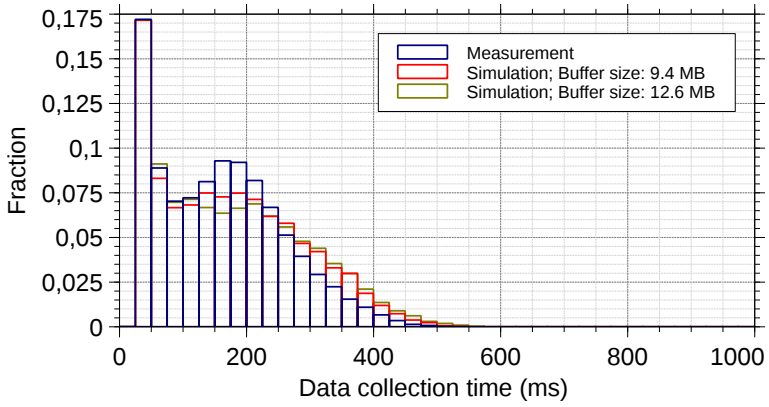
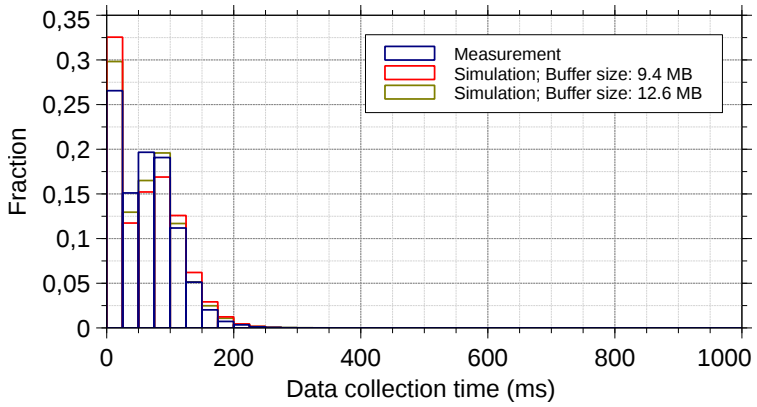


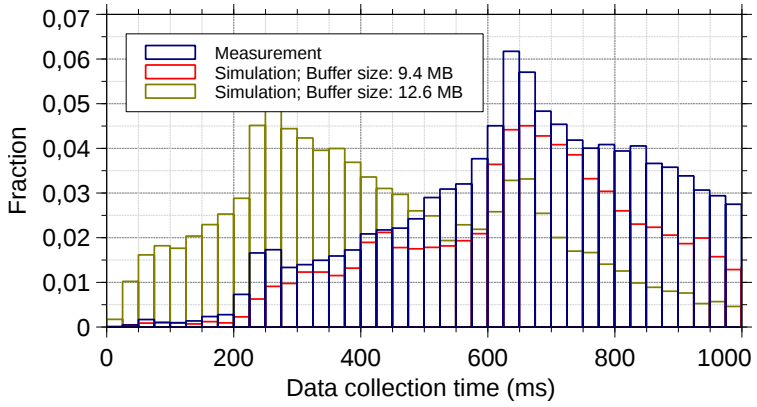
Figure 5.8 Distributions of measured and simulated data-collection times, using switch B, at a constant bandwidth of 1.06 GB, with the maximum burst size limited by the traffic shaping algorithm to (a) 52.8 kB, (b) 528 kB, (c) unlimited.



(a)



(b)



(c)

Figure 5.9 Distributions of measured and simulated data-collection times, using switch B, at a constant bandwidth of 2.11 GB, with the maximum burst size limited by the traffic shaping algorithm to (a) 52.8 kB, (b) 528 kB, (c) unlimited.

Enhancements for next-generation data-acquisition systems

6

The simulation model presented in chapter 5 was validated by showing that it yields results that reproduce the behaviour of the real system. Therefore, it can be used as a tool to evaluate the effectiveness of different solutions aimed at improving the system performance, i.e. reducing the data-collection time. Clearly, the possible modifications to the system that can be implemented in a simulation are almost endless. This chapter focuses on non-invasive solutions that can be readily deployed on an existing system. Therefore, solutions requiring specialised hardware or unsupported modifications to the operating system are not considered. The various modifications are evaluated by simulating their effects on the tests described in section 4.2.1 and section 4.3.1.

6.1 Work assignment policies

As mentioned in section 4.3.2, the HLT supervisor assigns events to HLT processing units on a first-come first-served (FCFS) basis, i.e. events are assigned to processing units in order of arrival of their assignment requests. Performance limitations in the HLT supervisor software currently prevent event-assignment policies more complex than FCFS from being implemented. However, it is reasonable to expect that more suitable policies reduce the contention for traffic-shaping credits and therefore improve the system performance. Naturally, the simulation is not affected by the performance issues mentioned above, so alternative policies can be easily implemented. Besides FCFS,

four event-assignment policies were modelled:

- Random: the supervisor chooses a random processing unit out of all of those that requested an event assignment, without regard for the order in which they did so.
- Node round-robin: the supervisor chooses a processing unit running on the next node in a fixed sequence of nodes.
- Node load-balancing: the supervisor chooses a processing unit running on the node with the most idle units.
- Processing unit round-robin: the supervisor chooses the next processing unit in a fixed sequence of units, where units running on the same worker node are consecutive.

The simulation reproduces the configuration described in section 4.3.2. The results are shown in figure 6.1 for Switch A and in figure 6.2 for Switch B. As usual, both the "medium throughput" (1.06 GB/s: 45% rack uplinks utilisation, figures 6.1a and 6.2a) and "high throughput" (2.11 GB/s: 90% rack uplink utilisation, figures 6.1b and 6.2b) scenarios are shown. As explained in section 4.2.2, for these tests the theoretical minimum data-collection time is around 18.2 ms. With the FCFS policy the average time is significantly higher than that in all scenarios: at medium throughput it is never below 40 ms, and at high throughput it is never below 60 ms.

The change in policy from FCFS to random leads to a very significant reduction of the data-collection latency, especially in the region of the parameter space where the traffic-shaping algorithm is most effective. The explanation for this difference lies in the mapping of processing units to nodes. While the traffic-shaping credits limit is enforced on a per-node basis, events are assigned on a per-processing-unit basis. In the particular set-up used, there are 24 units per node (see section 4.2.1). If events are assigned in quick succession to processing units running on the same node, those units will collect data at the same time, thus competing for the same pool of traffic-shaping credits. The random policy reduces the probability that two or more events will be assigned in a short interval to units hosted by the same node. Units on different nodes do not compete for the same credit pool, ultimately resulting in a lower average data-collection time. While the random event-assignment policy does enable reaching an average data-collection time of around 20 ms at medium throughput, it does not perform as well at full throughput.

The node load-balancing policy further reduces the chance of two events being assigned to the same node, which translates to further reductions of the data-collection time in the region where traffic-shaping is effective. The node round-robin policy yields similar

results when the traffic-shaping mechanism is effective. On top of that, it performs better than all the others when the nodes have very few credits available. It also has a slight advantage over the others in the incast-affected region. With this policy, once a supervisor has assigned an event to one node, it will have to assign events to all the other nodes before coming back to the first. Thus, the overlap between data collections initiated by different units on the same node is minimised. As a result, the policy yields optimal results both at medium and full throughput, reaching average data-collection times of 20 ms, close to the theoretical minimum mentioned above.

Finally, the processing unit round-robin policy is an example of the consequences of a bad policy choice: by concentrating the data-collection on the same node at the same time, all 24 processing units on a node compete for the same credits, resulting in abysmal performance.

As evidenced by comparing figures 6.1a and 6.2a with figures 6.1b and 6.2b, the performance differences among the event-assignment policies are higher when the data-acquisition throughput is higher.

6.2 Variable fragment sizes

Both the measurements described in chapter 4 and the simulation results presented so far use a single fixed size for all the data fragments (1.1 kB) of an event. This choice was made in order to reduce the complexity of the tests so that the measurements could be more easily interpreted. In more realistic conditions, however, the size of fragments depends on the characteristics of the particular phenomenon observed by the detectors. In high-energy physics, the event size distribution is generally quite asymmetric with a long tail: "ordinary" events tend to be small and appear rather frequently; events corresponding to rare interesting phenomena in the detector tend to have bigger fragments. Moreover, within an event, fragment sizes are not all identical: they depend on a multitude of factors, including the observed phenomenon and the region of the detector they correspond to. Given that the traffic-shaping algorithm uses the number of fragments in a data request as an estimate of the size of the corresponding response, its performance is expected to worsen when the fragment sizes are not always the same.

In order to simulate this, the fragment sizes are modelled as the sum of two components. The first component changes for every event, while the second component changes for every fragment of an event. The first component is determined as follows. A value

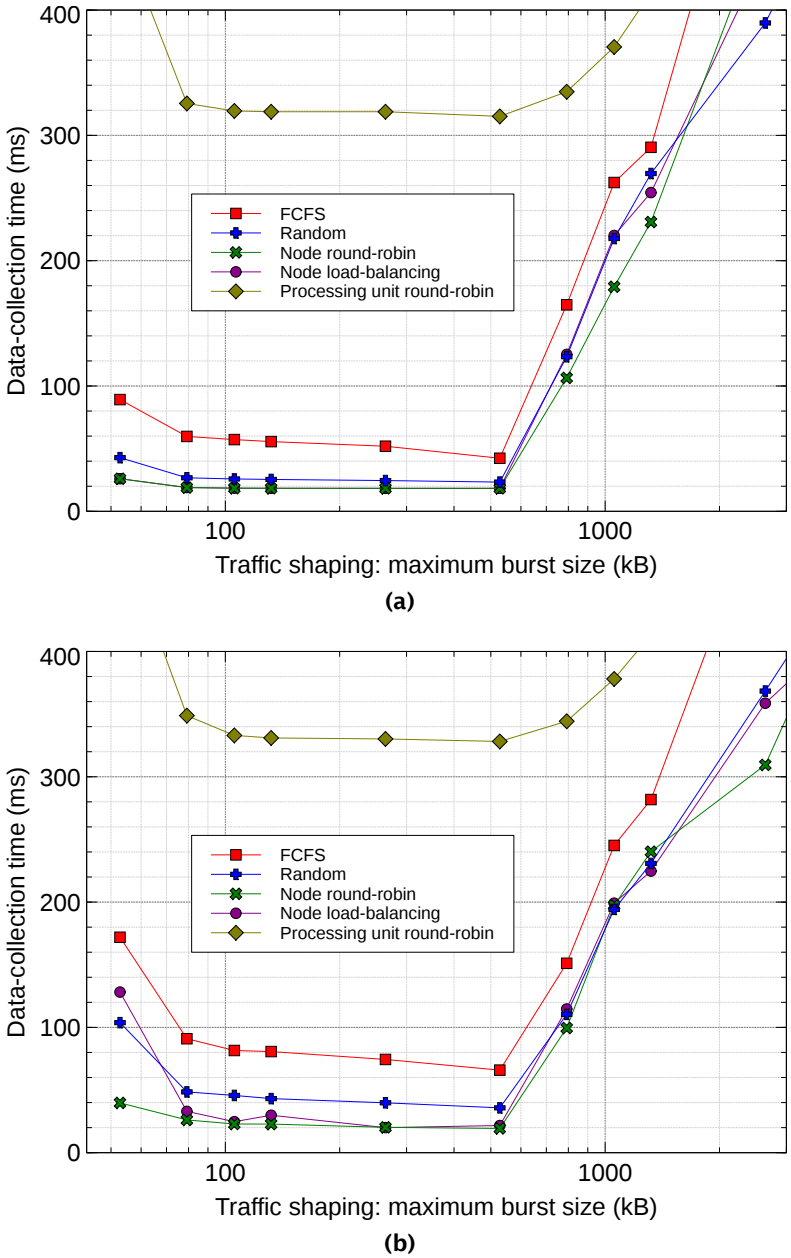


Figure 6.1 Average data-collection times as a function of the maximum burst size allowed by the traffic-shaping algorithm, for different event assignment policies, using switch A, at a constant bandwidth of (a) 1.06 GB/s and (b) 2.12 GB/s.

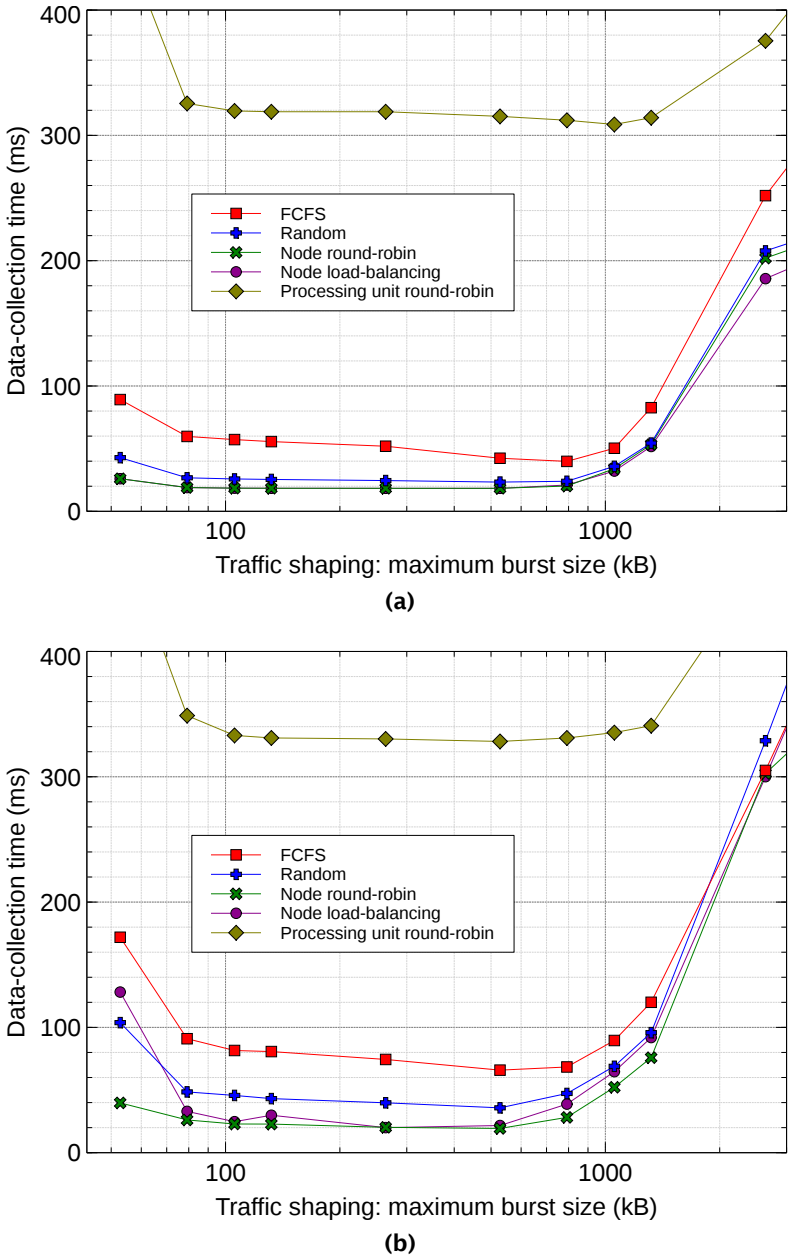
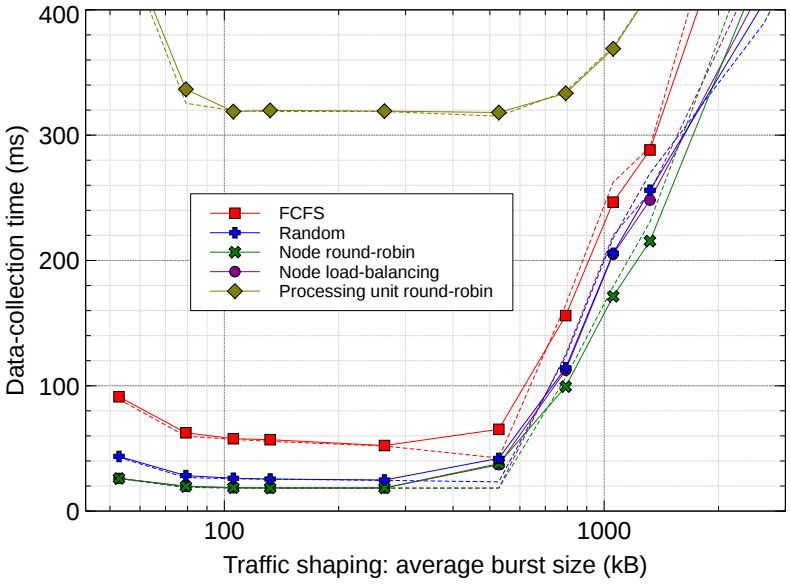
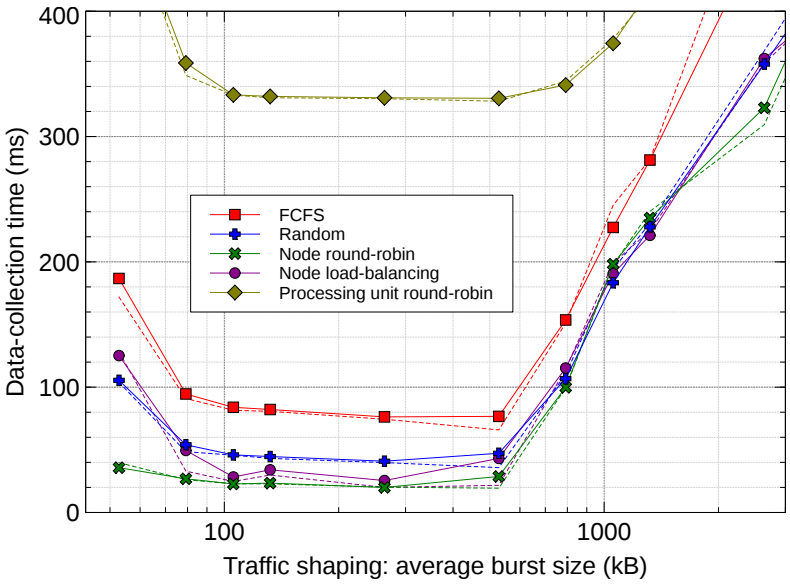


Figure 6.2 Average data-collection times as a function of the maximum burst size allowed by the traffic-shaping algorithm, for different event assignment policies, using switch B, at a constant bandwidth of (a) 1.06 GB/s and (b) 2.12 GB/s.

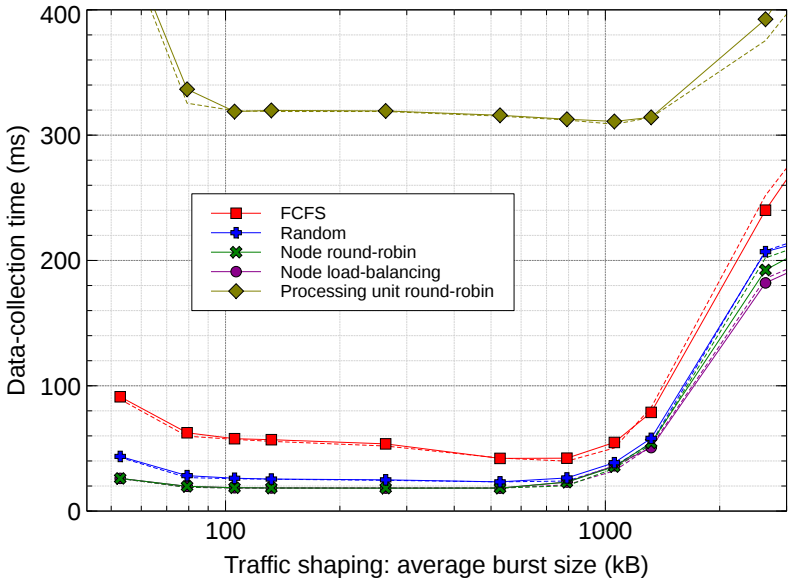


(a)

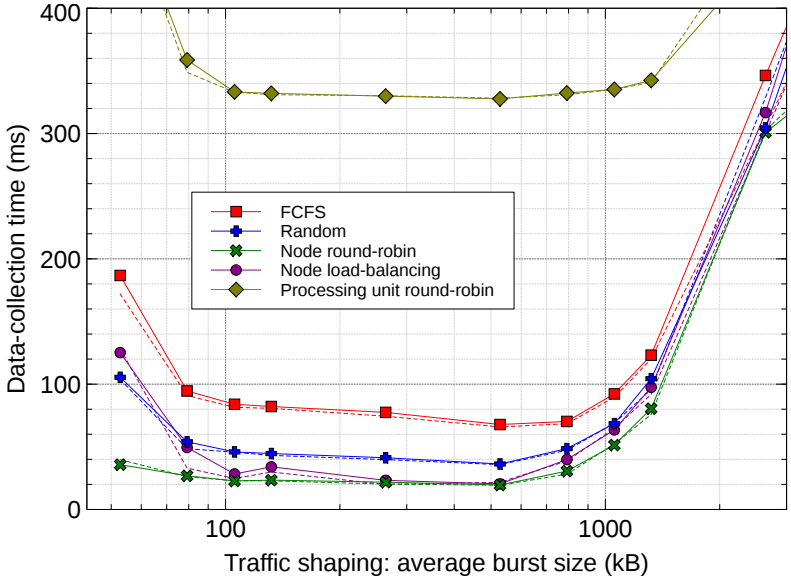


(b)

Figure 6.3 Average data-collection times as a function of the average burst size allowed by the traffic-shaping algorithm, for different event assignment policies, using switch A, at a constant bandwidth of (a) 1.06 GB/s and (b) 2.12 GB/s. The solid lines represent the data-collection latencies with with variable event sizes, as described section 6.2. For comparison, the dashed lines represent the data-collection latencies with fixed event sizes.



(a)



(b)

Figure 6.4 Average data-collection times as a function of the average burst size allowed by the traffic-shaping algorithm, for different event assignment policies, using switch B, at a constant bandwidth of (a) 1.06 GB/s and (b) 2.12 GB/s. The solid lines represent the data-collection latencies with with variable event sizes, as described section 6.2. For comparison, the dashed lines represent the data-collection latencies with fixed event sizes.

representing the event size is extracted from a log-normal distribution with location parameter 0 and scale parameter 0.25, modelling the long-tail distribution described above. The distribution is scaled so that the average event size is equal to the fixed event size used previously (2112 kB). The extracted event size is equally subdivided among the event fragments. The second component instead models the size variability among the various fragments of the same event. It is extracted from a normal distribution with mean 0 B and standard deviation 200 B.

The results are presented in figure 6.3 for the simulation using Switch A. The incast onset point is reached with fewer available credits than with fixed-size fragments: the traffic-shaping algorithm cannot prevent packet drops as effectively as with fixed-size fragments. This is expected, since Switch A has a dedicated fixed-size output buffer for each port (see section 4.2.1). With fixed-size fragments, the size of traffic bursts directed to a worker node cannot exceed the product of the node's traffic-shaping credits and the fragment size. With variable fragment sizes, instead, the traffic-shaping mechanism can only ensure that the average size of traffic bursts corresponds to that product. When the average burst size is close to the switch's buffer size (~500 kB), some traffic bursts might overrun the buffer, leading to packet drops. However, an optimal operating region can still be reached. In particular, if the chosen event assignment policy effectively spreads the events on all the available nodes, like in the case of the node load-balancing and round-robin policies, the minimum data-collection latency value is identical to the value that can be reached with fixed-size fragments.

The results for Switch B are shown in figure 6.3. In this case, the variable event sizes have little impact on the performance of the traffic-shaping algorithm. This is due to the switch's shared buffer architecture. Since all traffic bursts are absorbed by the same common buffer, above-average bursts are compensated by below-average bursts. Thus, buffer overruns are much rarer with respect to the scenario using Switch A.

Since in the real system the fragments sizes do in fact vary, all the other simulation results presented in this chapter will be based on the fragment size model described above.

6.3 Reducing TCP's minimum retransmission time-out

As explained in section 2.3, one TCP parameter, the minimum retransmission time-out (RTO), has a large impact on the system's throughput when the traffic pattern triggers

the TCP incast pathology. TCP's passive packet loss recovery mechanism retransmits packets that were not acknowledged within the RTO. The RTO value is based on the estimated round-trip time of the connection:

$$RTO = SRTT + 4 \times RTTVAR$$

where $SRTT$ and $RTTVAR$ are TCP's smoothed estimations of the round-trip time and its variation (see [60] for more details). However, the TCP specification also sets a minimum value for the RTO. The reason for defining a minimum is to prevent spurious retransmissions caused by [56]:

- RTO smaller than the operating system's timer granularity,
- RTO smaller than the TCP delayed acknowledgement time-out.

Besides needlessly increasing the injected traffic, spurious retransmissions have a more serious consequence: TCP uses dropped packets to detect network congestion and adjust its sending window. As a consequence, spurious retransmission time-outs cause spurious decreases of the sending window, lowering the connection's throughput.

Based on the analysis found in [5], the original specification sets the minimum RTO to 1 s. In Linux, the usual timer granularity is 1 ms. However, the acknowledgement time-out varies between 40 ms and 200 ms. Thus, to prevent spurious retransmissions, the minimum RTO is set to 200 ms. Data-centre networks such as ATLAS's have typically sub-millisecond round-trip times, meaning that the default minimum RTO is completely inadequate. In addition, delayed acknowledgements provide little benefits in a data-centre network: the reduction in acknowledgement packet rate is only useful when the path from the receiver to the sender is congested; the reduction in acknowledgement-processing overhead is insignificant given the available processing power. In Linux, delayed acknowledgements can be explicitly disabled either by the application itself or with a configuration parameter.

Lowering the minimum RTO was proven effective in mitigating the effects of incast in storage networks suffering from incast [65], so it might provide the same benefits to data-acquisition systems. In an unmodified Linux kernel, the minimum RTO can be configured as low as the timer granularity, i.e. 1 ms. However, since also $SRTT$ and $RTTVAR$ are measured with the same 1 ms granularity, the minimum is effectively 5 ms ($RTO = SRTT + 4 \times RTTVAR$ with $SRTT = RTTVAR = 1\,\text{ms}$). This is still one order of magnitude larger than the typical round-trip time of the ATLAS network. As shown in [65], lower RTO values can be achieved by modifying the Linux source code to use

higher-resolution clocks to measure the round-trip time. However, these modifications are experimental and rather invasive. Therefore, they impose a significant maintenance burden and cannot be easily deployed on an existing system, so they are not considered here.

The effectiveness of a lower TCP minimum RTO was evaluated first as a stand-alone solution to the incast problem, i.e. without enabling the traffic-shaping mechanism. The simulation was used to record the average data-collection time for values of the minimum RTO parameter between 5 ms and 200 ms. As already mentioned, minimum RTOs lower than 200 ms can result in unnecessary time-outs due to the delayed acknowledgements mechanism. In order to evaluate the performance impact of this phenomenon, the simulations were run with delayed acknowledgements both enabled and disabled.

The results are shown in figure 6.5 for Switch A and in figure 6.6 for Switch B. In all cases, lowering the minimum RTO successfully lowers the average data collection time. As expected, the best results are obtained when the lowest minimum RTO (5 ms) is configured and the "best" event-assignment policies ("node round-robin" and "node load-balancing) are used. In particular, in the "medium throughput" scenarios (figure 6.5a and figure 6.6a), the average data-collection time is around 26 ms with Switch A and 38 ms with Switch B. However, in the "high throughput" scenarios (figure 6.5b and figure 6.6b), the data-collection time does not drop below 95 ms using Switch A and 55 ms using Switch B. These times represent a huge decrease from the hundreds of milliseconds corresponding to the default minimum RTO. Enabling or disabling the delayed acknowledgement mechanism (refer to the dotted data series in the figures) leads to curious results: with Switch A, data-collection times are generally lower when delayed acknowledgements are disabled. Unexpectedly, with Switch B the opposite is true.

The results described above show that lowering the minimum RTO does mitigate the consequences of the incast pathology in all simulated scenarios. However, as a stand-alone solution, it does not achieve data-collection times as low as the traffic-shaping algorithm. As shown in section 6.2, the client-side traffic-shaping algorithm can lower the data-collection times to 20 ms. The optimal operation of the algorithm, however, requires that its "average burst size" parameter be in a rather narrow range. The optimal range depends on a multitude of factors, including: the average buffer occupancy and round-trip time of the network path between client and servers; the variance of the size of fragments sent by the servers. Outside of this range, the data-collection time

rapidly increases to hundreds of milliseconds. Combining client-side traffic shaping with a lowered minimum RTO should mitigate this problem, thus making the system more resilient against unforeseen operating conditions and misconfiguration. In order to evaluate the effectiveness of this, the simulation was used to record the average data-collection time for different average burst sizes in three different configurations:

- minimum RTO lowered to 5 ms, delayed acknowledgements disabled (LowRto-QuickAck);
- minimum RTO lowered to 5 ms, delayed acknowledgements enabled (LowRto-DelAck);
- minimum RTO set to the default 200 ms, delayed acknowledgements enabled (Default).

The results are shown in figure 6.7 for Switch A and in figure 6.8 for Switch B. In all simulated scenarios, the LowRto-QuickAck configuration (refer to the solid data series in the figures) yields the same data-collection times as the Default configuration (dashed data series) when the average burst size is lower than the incast onset threshold, i.e. when the traffic-shaping algorithm completely prevents packet drops (average burst size $\leq$300 kB for Switch A, $\leq$500 kB for Switch B). In the same region, the LowRto-DelAck configuration (dotted data series) yields consistently higher latencies than the Default configuration. This is due to the disruption to TCP operations caused by spurious retransmits. More importantly, in both LowRto configurations the data-collection times beyond the incast onset threshold are greatly improved. The optimal operating range of the traffic-shaping algorithm is significantly extended in all scenarios and, even where the data-collection latencies are not minimised, they are greatly reduced with respect to the Default configuration.

6.4 Centralised traffic scheduling

The incast pathology is caused by the lack of coordination among the nodes sending data to the same destination. A centralised scheduler of the data transfers would completely eliminate this problem. By guaranteeing that data transfers are only scheduled when the whole network path from the source to the destination is available, it could drastically reduce the use of buffers in the network devices. Naturally, this comes at a price: a central scheduler is a single point of failure and its maximum performance limits the scalability of the system. The consequences of failures can be mitigated or

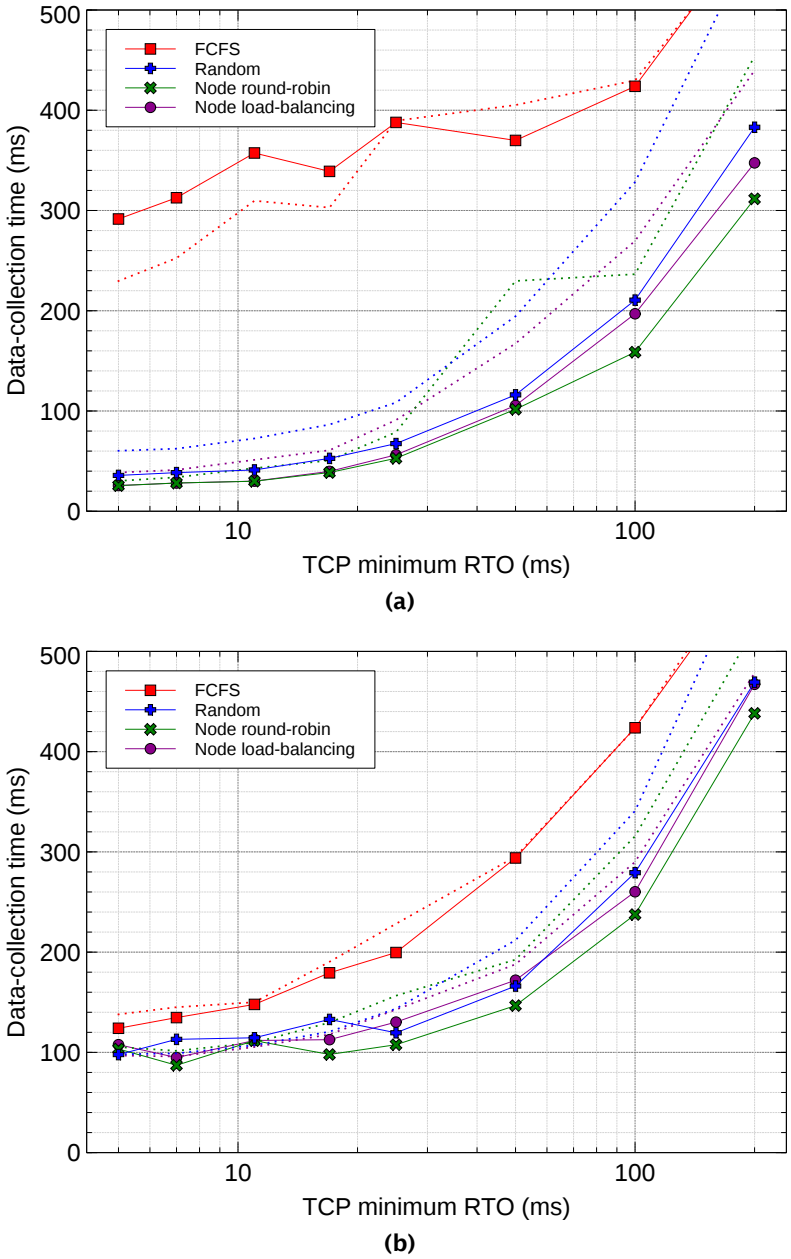


Figure 6.5 Average data-collection times as a function of the minimum TCP retransmission time-out (RTO), using Switch A, at a constant bandwidth of (a) 1.06 GB/s and (b) 2.12 GB/s. The solid lines correspond to simulations with TCP delayed acknowledgements disabled; the dotted lines correspond to simulations with TCP delayed acknowledgements enabled.

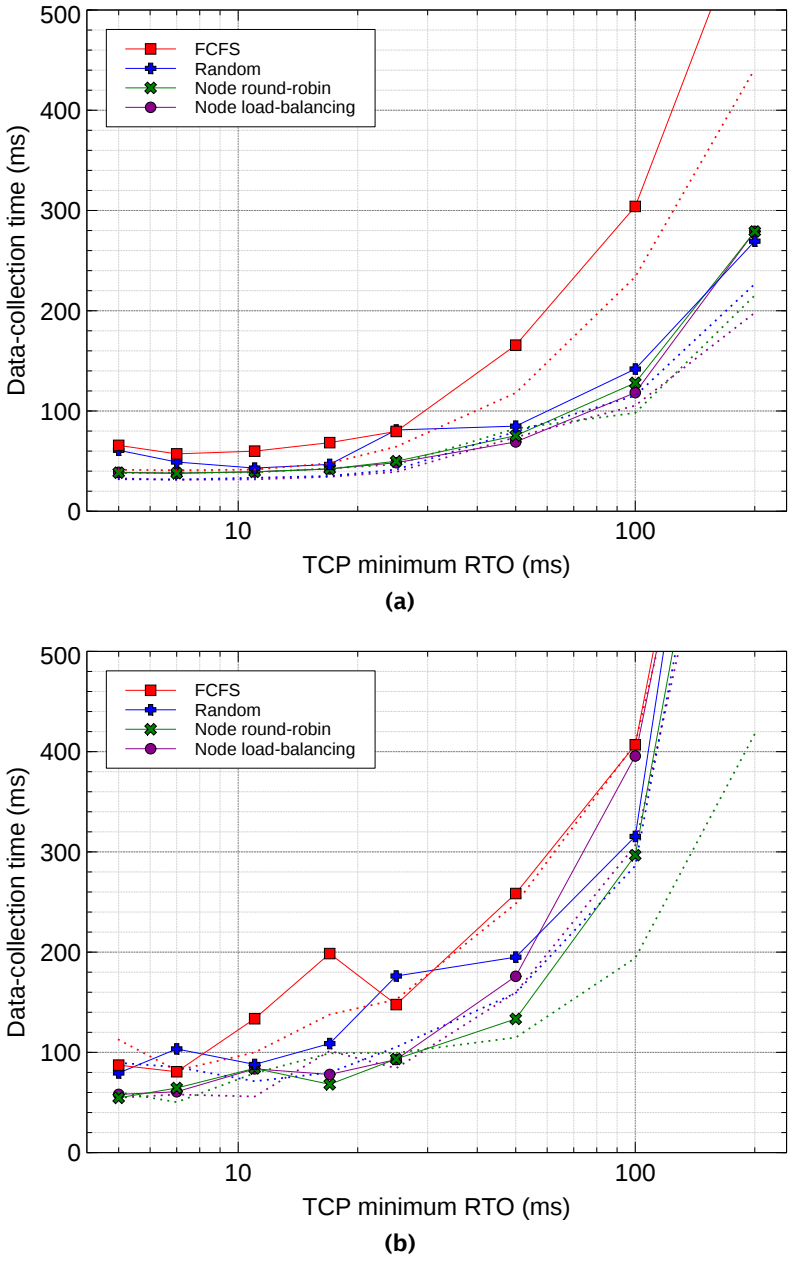
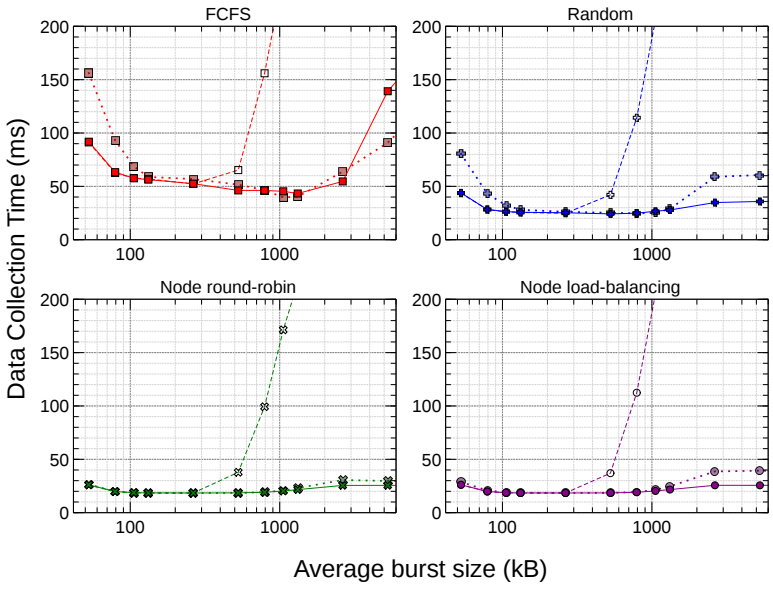
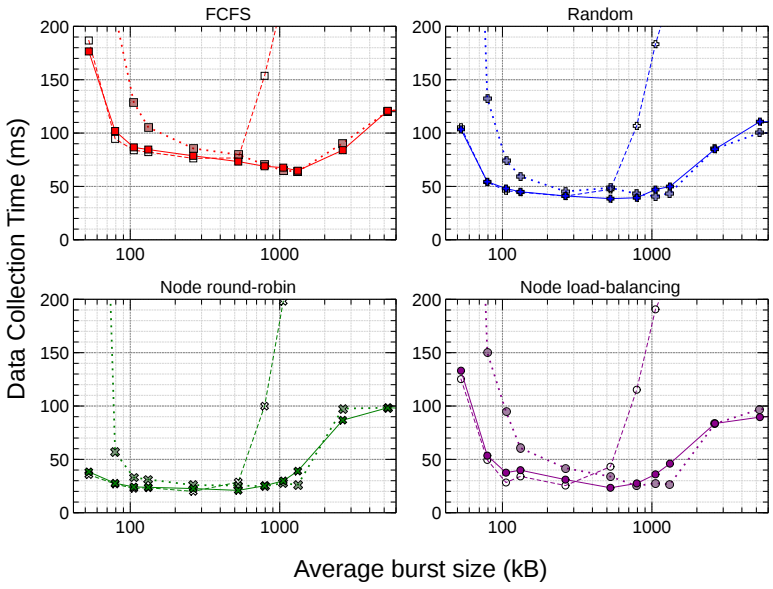


Figure 6.6 Average data-collection times as a function of the minimum TCP retransmission time-out (RTO), using Switch B, at a constant bandwidth of (a) 1.06 GB/s and (b) 2.12 GB/s. The solid lines correspond to simulations with TCP delayed acknowledgements disabled; the dotted lines correspond to simulations with TCP delayed acknowledgements enabled.

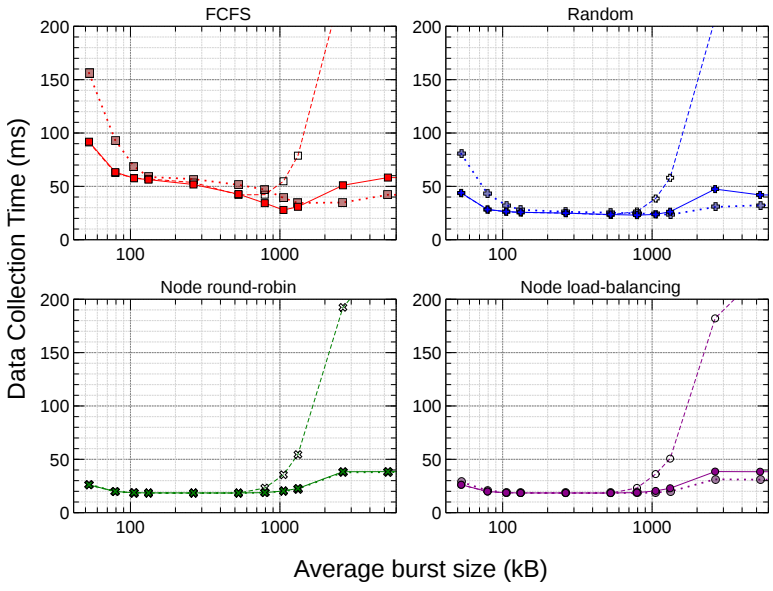


(a)

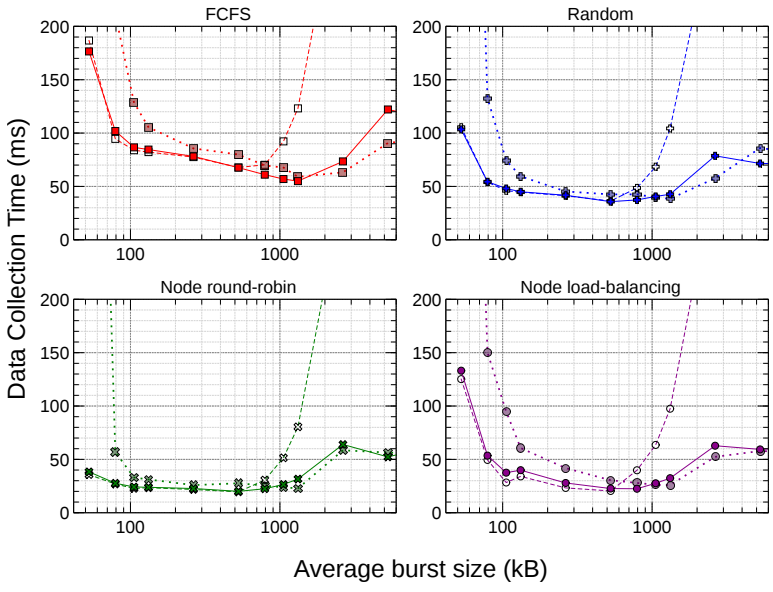


(b)

Figure 6.7 Average data-collection times as a function of the average burst size allowed by the traffic-shaping algorithm, using Switch A, at a constant bandwidth of (a) 1.06 GB/s and (b) 2.12 GB/s. The solid lines correspond to a minimum RTO of 5 ms and delayed acknowledgements disabled; the dotted lines correspond to a minimum RTO of 5 ms and delayed acknowledgements enabled; the dashed lines correspond to a minimum RTO of 200 ms and delayed acknowledgements enabled.



(a)



(b)

Figure 6.8 Average data-collection times as a function of the average burst size allowed by the traffic-shaping algorithm, using Switch B, at a constant bandwidth of (a) 1.06 GB/s and (b) 2.12 GB/s. The solid lines correspond to a minimum RTO of 5 ms and delayed acknowledgements disabled; the dotted lines correspond to a minimum RTO of 5 ms and delayed acknowledgements enabled; the dashed lines correspond to a minimum RTO of 200 ms and delayed acknowledgements enabled.

avoided with a reliable failover strategy (e.g. multiple schedulers in an active-standby set-up). The scalability limits cannot be easily worked around. Therefore, a centralised scheduler should only be considered if it can actually handle the maximum scale of the system in terms of number of nodes, number of connections, and total throughput. As a consequence, discussing and simulating an abstract scheduler without referencing a real-world implementation would be rather pointless. Instead, this section evaluates an existing solution, called Fastpass, proposed in [52].

With Fastpass, senders delegate control of when each packet is transmitted to a centralised scheduler[1]. When a node has packets ready to send, the operating system sends this demand in a message to the scheduler, specifying the packets' destinations and total sizes. The scheduler decides the time when each packet can be transmitted. It operates with the granularity of a "timeslot". A timeslot is the time taken to transmit a single maximum-size data-link packet (the so-called maximum transmission unit, MTU) over the fastest link connecting a computer to the network. The scheduler is aware of the network topology. For each timeslot, it selects a set of source-destination pairs that can communicate, ensuring that the traffic will not exceed the bandwidth of any link in the path from the source to the destination. This minimises the length of the queues in the network switches, making buffer overflows essentially impossible. The scheduler chooses the order in which demands are processed. For fairness, it selects source-destination pairs on a "least recently scheduled first" basis.

On the nodes, Fastpass is deployed as a plug-in to Linux's traffic control framework[2]. It queues outgoing packets before sending them to the network interface and sends the allocation demands in a request to the scheduler. The scheduler eventually replies with the timeslots allocated to each destination. The scheduler runs on a Linux PC, but it is implemented on top of Intel DPDK, a framework that enables direct access to the network interface queues, bypassing the operating system. Stress tests described in [52] show that the scheduler implementation can handle a system with 10 Gb/s links and an aggregate traffic of over 2 Tb/s. As reported in section 3.7, the ATLAS data-acquisition system operates at an input throughput of up to 200 GB/s, i.e. 1.6 Tb/s. The Fastpass scheduler is therefore suitable for handling this volume of traffic.

The simulations presented in this section are aimed at quantifying the impact of various operational parameters of the Fastpass system on the overall data-acquisition performance. The goal is to verify that Fastpass (or a similar centralised traffic scheduler) can

[1] Fastpass also includes centralised network path selection, which is not considered here.
[2] More specifically, a Linux traffic control queuing discipline (*qdisc*).

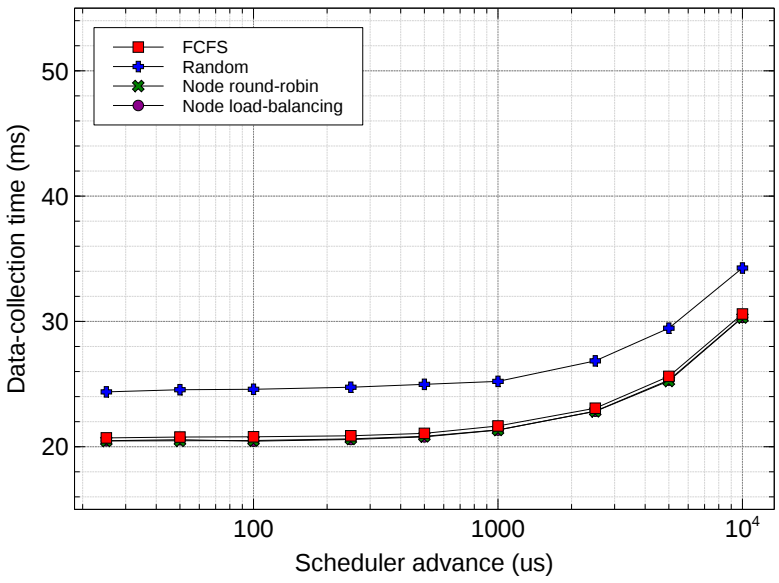
operate effectively in the environment of the ATLAS data-acquisition system. With centralised packet scheduling, the switch buffer occupancies are minimised and, without packet drops, TCP's congestion control is effectively bypassed. Therefore, in contrast with the other sections in this chapter, simulating different switch buffer models yields identical results. Fastpass demand and allocation messages share the same network resources as the scheduled traffic. For the purposes of these simulations, it is assumed that the network switches are configured to prioritise Fastpass packets over normal traffic. Unless specified, the one-way delay between a sender's traffic control plug-in and the scheduler application is on average 100 µs with a standard deviation of 50 µs. This configuration is intended to be representative of a possible deployment of the centralised scheduler within the existing ATLAS data-acquisition network.

6.4.1 Scheduler timing

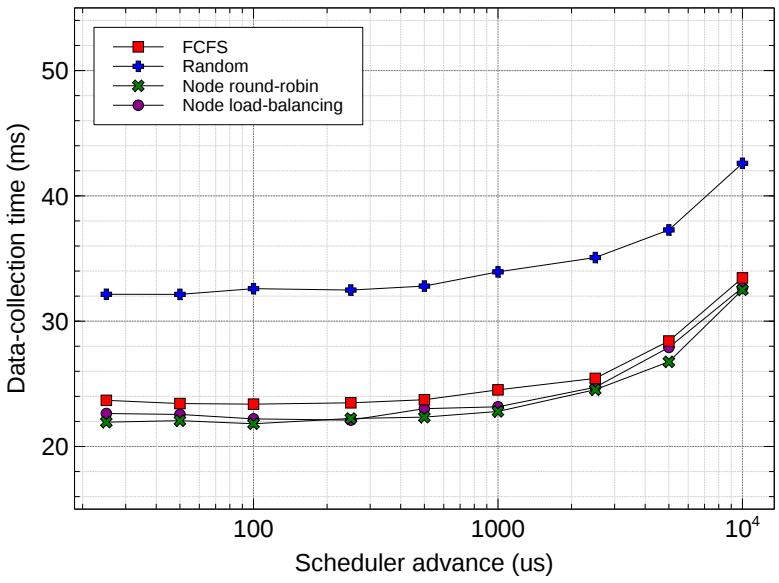
Once the scheduler has decided which source-destination pairs are allowed to communicate during a given timeslot, it sends this information as network messages to the relevant senders. Naturally these messages incur a certain delay before they reach and are processed by the senders. Therefore, the scheduler must conclude the allocation of a timeslot some time before that timeslot, so that the messages have time to reach the senders while they are still valid. In practice, the deadline for scheduling the timeslot starting at $t_{timeslot}$ is $t_{deadline} = t_{timeslot} - \Delta t_{advance}$ where $\Delta t_{advance}$ is a configurable time interval. On one hand, if $\Delta t_{advance}$ is too small, some scheduler messages might reach the senders when the timeslot has already passed. On the other hand, if $\Delta t_{advance}$ is too large, some demands that could have been satisfied in timeslot starting at $t_{timeslot}$ might have to wait for the next timeslot because they reached the scheduler after it concluded the allocation.

The goal of the simulation presented here is to determine the largest value of $\Delta t_{advance}$ that can be configured without significantly increasing the average data-collection time. The results are shown in figure 6.9. As expected, the lowest data-collection times are obtained, in conjunction with the node round-robin and node load-balancing policies, when the lowest $\Delta t_{advance}$ (25 µs) is configured. For $\Delta t_{advance} \leq 500$ µs, the data-collection times do not significantly increase, remaining around 21 ms and 22 ms respectively in the "medium throughput" and "high throughput" scenarios. Even with $\Delta t_{advance} = 1$ ms, the data-collection time only increases ~2% with respect to the minimum.

End-to-end delays in data-centre networks (and in the ATLAS data-acquisition network



(a)



(b)

Figure 6.9 Average data-collection times as a function of the scheduler advance $\Delta t_{advance}$, for different event assignment policies, at a constant bandwidth of (a) 1.06 GB/s and (b) 2.12 GB/s.

in particular) are of the order of 100 µs. These results show that a central scheduler can deal with those delays without a significant performance impact.

6.4.2 Clock synchronisation

A system with centralised traffic scheduling requires that nodes transmit packets exactly at the times chosen by the scheduler. Otherwise, multiple senders might use the same network resources at the same time, which results in queuing at the switches. Computer clocks are not perfect: many factors can alter their frequency in different ways, so that even initially synchronised clocks drift apart after some time. The well-established network time protocol (NTP) keeps computer clocks synchronised to a reference clock. On a local-area network, NTP can achieve millisecond or even sub-millisecond accuracy if the network delay jitter is small enough [44]. As already explained, with centralised packet scheduling, the network queues are very short, so clock synchronisation with millisecond accuracy is certainly possible. Greater accuracies (of the order of the microsecond) can be achieved with the precision time protocol (PTP), which requires hardware support in the network interface cards of the nodes and, optionally, for maximum accuracy, in the network switches. Consequently, it cannot be readily deployed on an existing system without invasive modifications.

In the worst case, the queue length will be $q = \Delta t_{clock}/\Delta t_{timeslot}$ packets, where Δt_{clock} is the maximum difference between any two clocks and $\Delta t_{timeslot}$ is the duration of a timeslot. A timeslot corresponds to the time needed to transmit a single maximum-size packet over the fastest link connecting a computer to the network. For Ethernet, that is ~12 µs if the link is 1 Gb/s or ~1.2 µs if the link is 10 Gb/s. With a 1 ms maximum clock discrepancy, this results in large worst-case queue sizes: up to 82 packets (123 kB) at 1 Gb/s and 820 packets (1.23 MB) at 10 Gb/s. Therefore, it would seem that high-accuracy clock synchronization is absolutely required for the scheduling mechanism to work as promised. However, there are a few mitigating factors. First of all, this worst-case scenario is extremely unlikely in practice: for maximum queuing, the clock differences must be uniformly distributed between 0 and Δ_{clock}. Natural clock drift does not lead such a distribution. Moreover, even in the worst case, the queued packets would not overrun the buffers of a low-end data-centre switch with shared buffers (such as switch B): packet drops are still prevented. Finally, in data-acquisition workloads, the number of senders is relatively small (in the ATLAS system modelled in this thesis there are 160 ROS nodes). This also reduces the likelihood of hitting the worst-case scenario.

The simulation was used to confirm these assertions. Each sender was configured with a fixed clock difference with respect to the scheduler. The time differences were extracted from an uniform distribution. The width of the distribution is the maximum difference between any two clocks Δt_{clock}. The average data-collection time was measured with Δt_{clock} set to 10, 100, 1000, and 10000 µs. The results are shown in figure 6.10. The data-collection time is basically unaffected by Δt_{clock} up to 1 ms. Only when Δt_{clock} is set to 10 ms a significant increase is seen. However, that increase is not due to queuing, but rather to the fact that some senders might start transmitting up to 10 ms later than others, and the data-collection is not completed until then.

These results show that a centralised packet scheduler can be successfully deployed in a data-acquisition system even without the added cost and complexity of a hardware-assisted clock synchronisation solution.

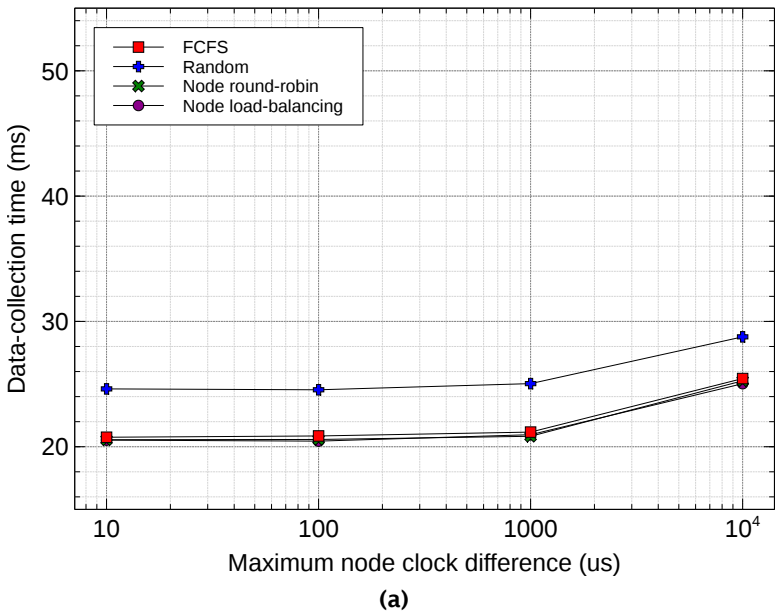
6.4.3 Scheduler granularity

The Fastpass scheduler operates at the granularity of one MTU. This makes it rather efficient in utilising the available network bandwidth. The only inefficiency occurs when nodes have less than a full MTU worth of data to send. In that case, some network bandwidth is left unused.

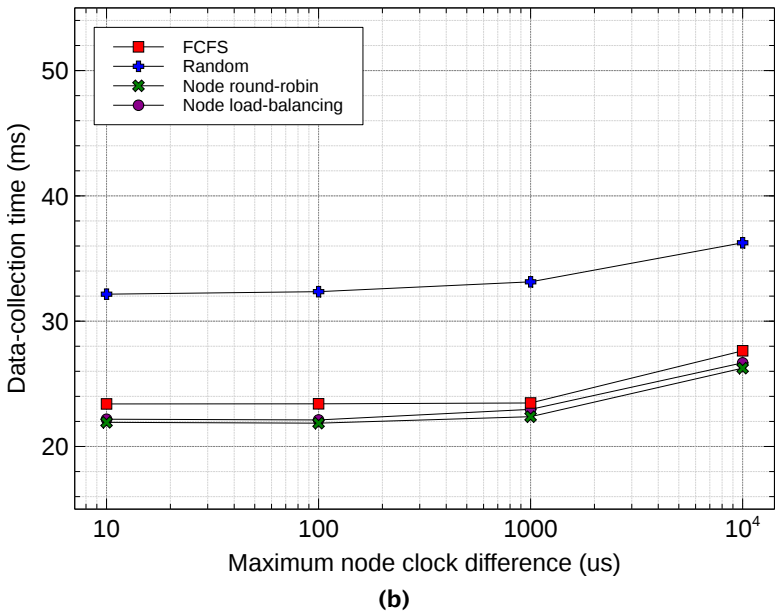
The scheduler granularity determines its performance requirements: a one-MTU timeslot lasts ~12 µs with 1 Gb/s edge links or ~1.2 µs with 10 Gb/s edge links. That means that, in order to keep up with the demands, the scheduler must allocate ~81000 timeslots per second or ~810000 timeslots per second respectively. An obvious way to reduce the load on the scheduler is thus to define longer timeslots. For example, using two-MTUs timeslots would halve the required allocations per second. Naturally, this increases the inefficiency due to partially unused timeslot.

The simulation presented measures the impact of multi-MTU timeslots on the overall system performance, i.e. on the data-collection time. The simulated timeslot durations correspond to granularities going from one to twelve MTUs. The results are shown in figure 6.11. The data-collection time grows roughly linearly with the timeslot duration: a one-MTU increase of the timeslot duration corresponds to a ~4% increase of the data-collection time with respect to the minimum.

As explained in section 6.4, the Fastpass scheduler is fast enough to handle the traffic volume of the ATLAS data-acquisition system as it is. However, this might not be the case for other systems or for future evolutions of the ATLAS system. If the scheduler

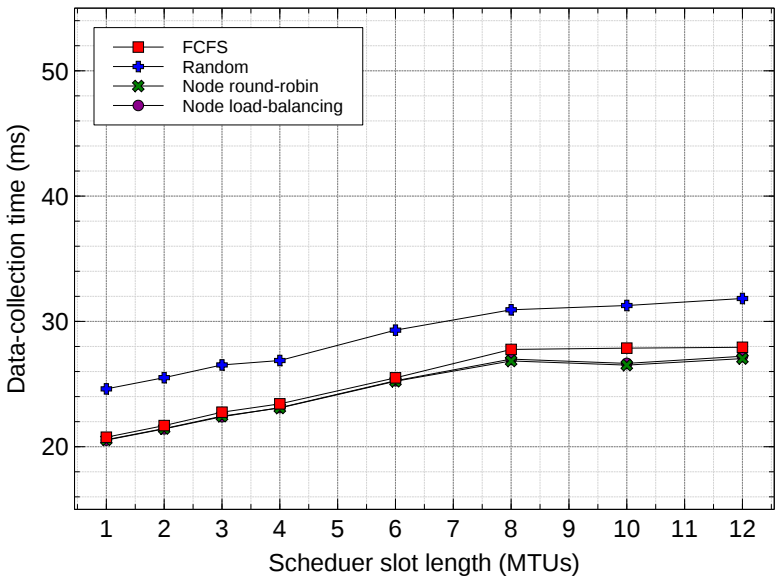


(a)

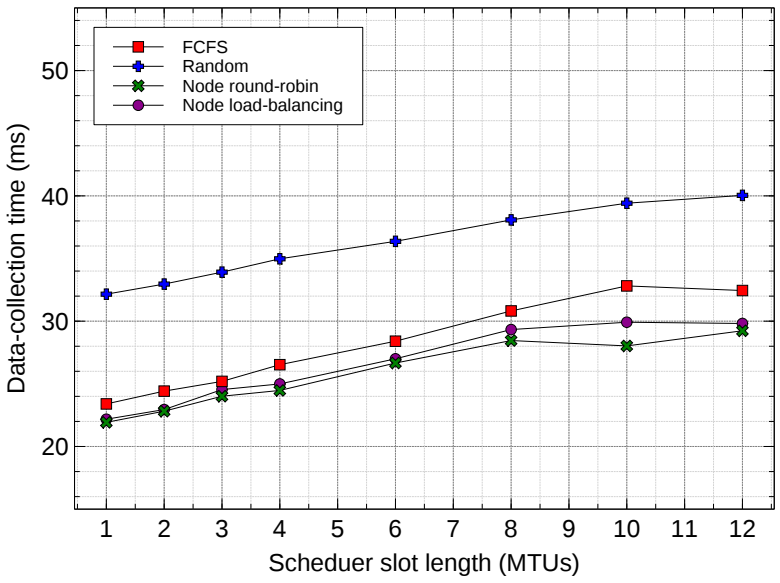


(b)

Figure 6.10 Average data-collection times as a function of the arbitration slot size, for different event assignment policies, at a constant bandwidth of (a) 1.06 GB/s and (b) 2.12 GB/s.



(a)



(b)

Figure 6.11 Average data-collection times as a function of the arbitration slot size, for different event assignment policies, at a constant bandwidth of (a) 1.06 GB/s and (b) 2.12 GB/s.

does not scale to the required performance, the available options are limited: either the implementation is improved, or centralised traffic scheduling must be abandoned altogether. As a work-around, increasing the timeslot duration can significantly reduce the load on the scheduler. As shown, the price for this reduction is an increase in the average data-collection latency. Depending on the performance objectives of the system in question, this might be an acceptable trade-off.

6.5 Switch buffer space

While the goal of this thesis is to evaluate enhancements that can be readily deployed on existing systems, the simulation model can also prove useful for guiding future network hardware purchases. In particular, it can be used to determine the relation between the size of the packet buffers in the top-of-rack switches and the data-collection time. It is possible to determine the minimum size of the packet buffers that completely prevents packet drops (and thus eliminates the incast phenomenon for the chosen traffic pattern) without having to limit the maximum burst size at the application level.

The results for a switch with dedicated per-port memory are shown in figure 6.12. When employing the FCFS assignment policy, buffers of at least 4 MB in the "medium throughput" scenario and 5 MB in the "high throughput" scenario effectively prevent all packet drops. The latency can then reach its lowest value (slightly lower than 20 ms, compatible with the theoretical minimum calculated in section 5.2.2). However, with the random assignment policy in the high-throughput scenario, the lowest latency isn't reached even with 5 MB buffers. This discrepancy, and in general the better performance of FCFS in this configuration, can be explained. If two events are consecutively assigned to two processing units on the same worker node, the data for the second event incurs a larger delay, since the buffer of the worker node's switch port still contains data from the first event, leading to queuing delays or overflow. This effect changes the order of the supervisor's FCFS queue: over time, entries in the queue referring to different units on the same worker node distance themselves. The available switch buffer memory is therefore used more efficiently, and queuing delays are avoided. As expected, the node load-balancing and round-robin algorithms outperform both FCFS and random, reaching the lowest latency with just 3 MB buffers in the medium-throughput scenario and 3.5 MB in the high-throughput scenario. As a reminder, the per-port memory of Switch A is ~750 kB. Even with the optimal event-assignment policy, the needed buffer sizes are 5 times larger than that.

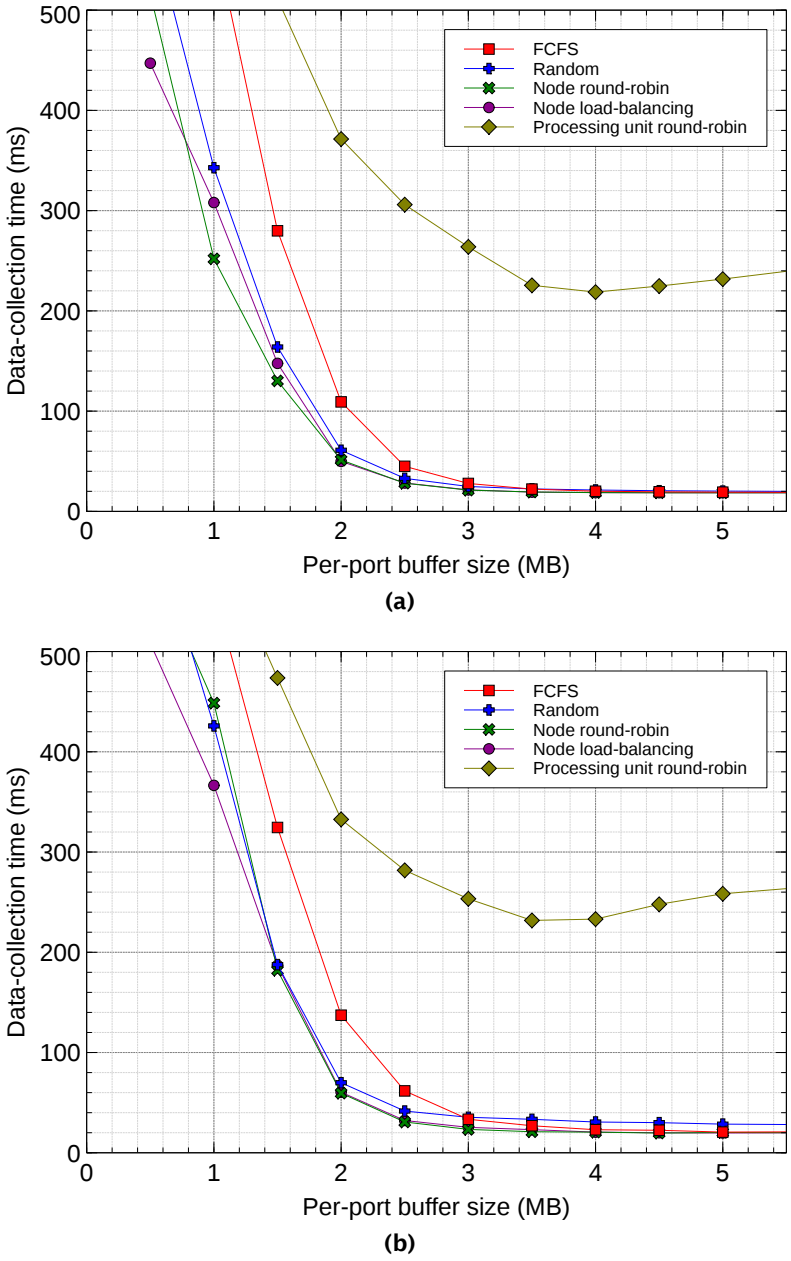


Figure 6.12 Average data-collection times as a function of the size of the per-port buffers in the top-of-rack switch, for different event assignment policies, at a constant bandwidth of (a) 1.06 GB/s and (b) 2.12 GB/s.

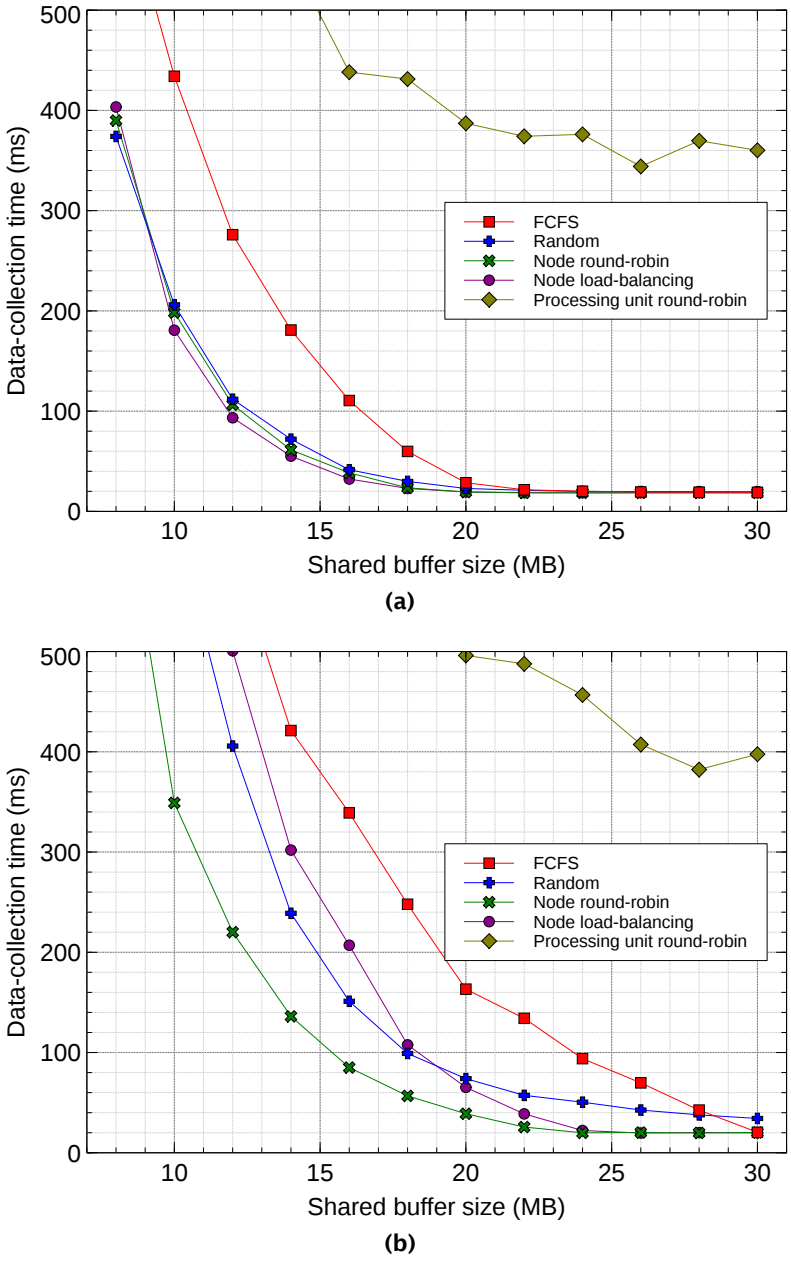


Figure 6.13 Average data-collection times as a function of the size of the shared buffer in the top-of-rack switch, for different event assignment policies, at a constant bandwidth of (a) 1.06 GB/s and (b) 2.12 GB/s.

The results for a switch with shared memory are shown in Figure figure 6.13. In the medium-throughput scenario, both with the FCFS and random assignment policies, the minimum data-collection time is reached with a buffer size of at least 24 MB. However, in the high-throughput scenario a 30 MB buffer size is needed to get to the minimum data-collection time with the FCFS policy. As with the previous simulation, the random policy does not reach the minimum latency in the high-throughput scenario. With the node load-balancing and round-robin policies, the minimum latency is reached with a shared buffer of at least 20 MB in the medium-throughput scenario and 24 MB in the high-throughput scenario. Interestingly, in the high-throughput scenario, for smaller buffer sizes, the node round-robin policy performs better than the node load-balancing policy. This shows the importance of spreading the traffic over all the available nodes and therefore over all the switch's output ports, so that it can be consumed rather than queued. With the node round-robin policy two events are never consecutively assigned to the same worker node. The node load-balancing policy does not offer that guarantee and therefore can cause higher buffer occupancy.

6.6 Discussion

All the solutions examined in this chapter improve the data-collection performance to various extents. The main findings are resumed here. By its very definition, the incast problem can be avoided if the buffers in the network switches are large enough to absorb the biggest bursts of data directed to a single destination, as shown in the previous section. If that is not the case, various strategies can be employed to reduce the effects of incast. As an alternative to the general but somewhat impractical solutions mentioned in section 2.4, the following possibilities were evaluated:

- a network-aware work-assignment algorithm,
- the receiver-side traffic shaping algorithm discussed in chapter 4,
- lowering the TCP retransmission time-out,
- centralised packet scheduling.

A good work-assignment algorithm (such as node round-robin or node load-balancing) spreads the traffic over all the available nodes and network links, ensuring an optimal utilisation of the available resources. Its implementation is necessary to obtain the best results with any of the solutions presented.

The receiver-side traffic shaping algorithm discussed in chapter 4:

- ✔ completely prevents network buffer overruns;
- ✔ combined with a good event-assignment policy, achieves the lowest data-collection time (~20 ms in the simulated scenarios, close the the theoretical minimum of 18.2 ms);
- ✗ has a narrow, traffic-pattern-dependent optimal operation range;
- ✗ requires the data fragments sizes to be somewhat predictable.

Lowering the TCP minimum retransmission time-out to 5 ms:

- ✔ is effective independently of the network traffic-pattern;
- ✗ performs significantly worse than traffic-shaping, especially when the network is heavily utilised (the lowest data-collection time achieved is ~40 ms in the high-traffic scenarios).

Receiver-side traffic-shaping and a lowered TCP minimum RTO (with delayed acknowledgements disabled) are not mutually exclusive. In fact, their combination, together with a good event-assignment algorithm, is a very effective technique to prevent and mitigate the consequences of the TCP incast pathology with data-acquisition traffic. The main drawback of this combination is that obtaining the best performance requires tuning the traffic-shaping algorithm to match the systems' configuration and traffic pattern. Given that in data-acquisition systems neither change often, this is not a serious impediment to its adoption.

A completely alternative solution, centralised packet scheduling:

- ✔ is effective independently of the network configuration and traffic-pattern;
- ✔ offers acceptable performance even if the simple FCFS policy is used for event assignment;
- ✗ limits the scalability and reliability of the whole system due to the centralised architecture;
- ✗ is not as effective as receiver-side traffic shaping (the lowest data-collection time achieved is ~22 ms in the high-traffic scenarios, 10% higher than when using a properly configured traffic-shaping).

7 Conclusion

Large-scale data-acquisition systems must aggregate data fragments from millions of sources into one coherent output. In the networked part of a data-acquisition system, communications are inherently many-to-one, bursty, and synchronous. This combination triggers the well-known incast pathology, which dramatically reduces the system's throughput. Many-to-one bursts are a natural feature of data acquisition: when an experiment observes a certain phenomenon, all its sensors produce data at the same time. The data-acquisition system must gather and collate these data fragments in a single location for further processing and analysis.

The goal of this work was to search for and evaluate effective ways of mitigating or eliminating the impact of incast in data-acquisition systems. In particular, the focus was on practical solutions, that can be deployed on existing systems without disrupting their operation. In order to do that, a real large-scale system was studied: the ATLAS experiment's trigger and data-acquisition system.

The most important metric of a data-acquisition system's performance was identified: the data-collection time, i.e. the time spent gathering data from the sources to the processing nodes. In systems where data is collected on-demand, a larger than necessary data-collection time effectively means wasted CPU time. Moreover, in incast-affected networks, resources are needlessly wasted by retransmitting dropped packets.

The measurements presented in chapter 4 showed that the ATLAS data-acquisition network is indeed vulnerable to incast. With no incast-avoidance measures in place, its data-collection time can get as high as 35 times larger than its ideal minimum. The measurements clearly show why: as long as the total amount of collected data can fit into the buffers of the switches between sources and receiver, the data-collection time stays low. When the data cannot be totally buffered, it becomes possible that all of the

transmission from one source is dropped. If this happens, the source can only detect that packets must be retransmitted by means of a relatively slow timeout mechanism. When many sources encounter this issue, it may even become necessary to retransmit packets a third or even a fourth time. Waiting for these timeouts and retransmissions causes the data-collection time to explode. These observations suggest a simple but effective mitigation strategy: a receiver-side traffic-shaping mechanism that reduces the effects of incast by only allowing a small number of senders to transmit data to the same receiver at the same time. Effectively, this smooths the bursts of many-to-one transmissions. In ATLAS this mechanism can reduce the data-collection time by a factor 10 in the worst case mentioned above. Even so, this approach, is not, by itself, capable of returning the system to its ideal performance levels, especially when the network utilisation is close to the maximum.

To further improve on the results obtained with traffic shaping, a simulation model was necessary: data-acquisition systems are mission-critical components of their experiments. Therefore, once they are commissioned, they are rarely available for performance measurements and implementing even small hardware or software modifications is often impossible. The discrete event simulation developed in chapter 5 models the ATLAS data-acquisition system at the granularity of a data-link packet. Applications, and network protocols were given the most realistic implementations. Switch and end-node hardware latencies were not modelled in detail. The underlying assumption is that data-collection latency is dominated by the delays caused by packet drops and retransmissions, which are several orders of magnitude larger than the typical switch or end-node hardware latency. Before putting it to use, it was validated extensively by comparing its results with measurements taken on the real system. As expected, the crude hardware latency model did not meaningfully impact the results. On the other hand, switch buffer sizes must be chosen carefully, as they are essential in determining packet drops. In general, the model's predictions were shown to be in good agreement with the measurements mentioned above, thereby validating the model and enabling its use to study further improvements.

The most important result of the simulations presented in chapter 6 shows that it is possible to undo the effects of incast in data-acquisition systems with pure software solutions, which do not require invasive and expensive hardware and operating system modifications. In particular, a combination of:

- a traffic-shaping algorithm that allows only a small number of senders to transmit data to the same receiver at the same time,

- a work-scheduling policy that spreads the load evenly on the network, and
- the tuning of two TCP parameters (lowering the minimum retransmission timeout and disabling delayed acknowledgements)

results in data-collection times that are within 10% of the ideal minimum, even with a high network traffic load. This combination is even more effective than more sophisticated approaches, like a somewhat impractical centralised packet-scheduling system.

The main drawback of that combination is that the traffic-shaping algorithm is not generic: it must be tuned to match the system configuration and traffic pattern. However, in data-acquisition systems these are rather static and known in advance, rendering this limitation less relevant. The simulation model developed for these studies can also double as a tool to estimate the best configuration of the traffic-shaping algorithm and the best work-assignment policy for a given system. Nevertheless, the development of an automatic discovery algorithm to find the optimal traffic-shaping parameter is a natural extension of this work.

These results enable the ATLAS data-acquisition system to operate at high efficiency, without losing network throughput to incast. More generally, they demonstrate that it is possible to build data-acquisition systems using low-cost best-effort networks without necessarily incurring in the catastrophic loss of throughput brought about by incast. More expensive lossless network technologies are not needed.

Given their negligible cost, low invasiveness, and relative ease of implementation, the solutions presented in this thesis can be recommended to currently operating and future data-acquisition systems based on best-effort network technologies.

Acknowledgements

First and foremost, I want to thank Prof. Dr. Holger Fröning for his guidance and directions as my doctoral advisor, and Prof. Dr. Pedro Javier García for his precious collaboration and his insights on computer networks.

I am grateful to the Faculty of Mathematics and Computer Science of Heidelberg University for the opportunity to pursue this doctorate.

I would like to thank Dr. Wainer Vandelli for his constant support and supervision during my time as a member of the ATLAS team at CERN. I am deeply indebted to Prof. Dr. Andrea Negri, who introduced me to the world of data acquisition, large-scale experiments, and CERN.

I owe most of my understanding of event-based network simulations to Dr. Pedro Yébenes Segura. His tips have been invaluable to me.

Special thanks to Eukeni Pozo Astigarraga, Dr. Giuseppe Avolio, Dr. Reiner Hauser, and Dr. Giovanna Lehmann Miotto. Working, discussing, and designing data-acquisition software side-by-side with them was an incredible learning experience.

I would not have ever finished writing this thesis without the constant comforting and encouragement of the amazing Margherita Boselli. Her in-depth reviews were instrumental in helping me make this work much better, and I look forward to return the favour soon.

Last but absolutely not least, I want to thank my family: my parents Alfredo and Bianca, and my sister Camilla. I will never have enough of their unconditional support and loving advice.

Bibliography

[1] R. Achenbach et al. "The ATLAS Level-1 Calorimeter Trigger". In: *Journal of Instrumentation* 3.3 (2008), P03001. ISSN: 1748-0221. DOI: 10.1088/1748-0221/3/03/P03001.

[2] ALICE Collaboration. "The ALICE experiment at the CERN LHC". In: *Journal of Instrumentation* 3.8 (2008), S08002. ISSN: 1748-0221. DOI: 10.1088/1748-0221/3/08/S08002.

[3] M. Alizadeh, A. Javanmard, and B. Prabhakar. "Analysis of DCTCP: Stability, Convergence, and Fairness". In: *Proceedings of the ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*. SIGMETRICS. New York: ACM, 2011, pp. 73–84. ISBN: 978-1-4503-0814-4. DOI: 10.1145/1993744.1993753.

[4] M. Alizadeh et al. "Data Center TCP (DCTCP)". In: *Proceedings of the ACM SIGCOMM 2010 Conference*. SIGCOMM. New York: ACM, 2010, pp. 63–74. ISBN: 978-1-4503-0201-2. DOI: 10.1145/1851182.1851192.

[5] M. Allman and V. Paxson. "On Estimating End-to-end Network Path Properties". In: *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. SIGCOMM. New York: ACM, 1999, pp. 263–274. ISBN: 978-1-58113-135-2. DOI: 10.1145/316188.316230.

[6] A. dos Anjos, H. P. Beck, and B. Gorini. *The raw event format in the ATLAS Trigger & DAQ*. Engineering Specification ATL-D-ES-0019 v.4.0e. Geneva: CERN, 2011. URL: http://edms.cern.ch/document/445840/4.0e.

[7] F. Anulli et al. "The Level-1 Trigger Muon Barrel System of the ATLAS experiment at CERN". In: *Journal of Instrumentation* 4.4 (2009), P04010. ISSN: 1748-0221. DOI: 10.1088/1748-0221/4/04/P04010.

[8] S. Ask et al. “The ATLAS central level-1 trigger logic and TTC system”. In: *Journal of Instrumentation* 3.8 (2008), P08002. ISSN: 1748-0221. DOI: 10.1088/1748-0221/3/08/P08002.

[9] ATLAS Collaboration. “Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC”. In: *Physics Letters B* 716.1 (2012), pp. 1–29. ISSN: 0370-2693. DOI: 10.1016/j.physletb.2012.08.020.

[10] ATLAS Collaboration. “The ATLAS Experiment at the CERN Large Hadron Collider”. In: *Journal of Instrumentation* 3.8 (2008), S08003. ISSN: 1748-0221. DOI: 10.1088/1748-0221/3/08/S08003.

[11] S. Baron. *TTC*. 2012. URL: http://ttc.web.cern.ch/ttc/ (visited on 03/26/2012).

[12] S. Bensley et al. *Datacenter TCP (DCTCP): TCP Congestion Control for Datacenters*. Internet Draft draft-ietf-tcpm-dctcp-03. Internet Engineering Task Force, 2016. URL: https://tools.ietf.org/html/draft-ietf-tcpm-dctcp-03.

[13] E. Blanton, M. Allman, and V. Paxson. *TCP Congestion Control*. Internet Request for Comments 5681. RFC Editor, 2009. URL: http://www.rfc-editor.org/rfc/rfc5681.txt (visited on 02/19/2016).

[14] L. S. Brakmo, S. W. O’Malley, and L. L. Peterson. “TCP Vegas: New Techniques for Congestion Detection and Avoidance”. In: *Proceedings of the ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*. SIGCOMM. New York: ACM, 1994, pp. 24–35. ISBN: 978-0-89791-682-0. DOI: 10.1145/190314.190317.

[15] Brocade Communications Systems. *MLX Series Routers*. URL: http://www.brocade.com/en/products-services/routers/mlx-series.html (visited on 02/21/2016).

[16] Brocade Communications Systems. *VDX 6740 Switches*. URL: http://www.brocade.com/en/products-services/switches/data-center-switches/vdx-6740-switches.html (visited on 02/21/2016).

[17] J. M. Campbell, J. W. Huston, and W. J. Stirling. “Hard interactions of quarks and gluons: a primer for LHC physics”. In: *Reports on Progress in Physics* 70.1 (2007), pp. 89–193. ISSN: 0034-4885, 1361-6633. DOI: 10.1088/0034-4885/70/1/R02. arXiv: hep-ph/0611148.

[18] J.-L. Caron. *Layout of the LEP tunnel including future LHC infrastructures*. 1997. URL: http://cdsweb.cern.ch/record/841560.

[19] *Castalia Wireless Sensor Network Simulator*. URL: https://castalia.forge.nicta.com.au/ (visited on 03/23/2016).

[20] CMS Collaboration. "Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC". In: *Physics Letters B* 716.1 (2012), pp. 30–61. ISSN: 0370-2693. DOI: 10.1016/j.physletb.2012.08.021.

[21] CMS Collaboration. "The CMS experiment at the CERN LHC". In: *Journal of Instrumentation* 3.8 (2008), S08004. ISSN: 1748-0221. DOI: 10.1088/1748-0221/3/08/S08004.

[22] T. Colombo and W. Vandelli. "Novel, highly-parallel software for the online storage system of the ATLAS experiment at CERN: Design and performances". In: *2012 18th IEEE-NPSS Real Time Conference*. RT. New York: IEEE, 2012, pp. 1–6. DOI: 10.1109/RTC.2012.6418361.

[23] T. Colombo et al. "Modeling a Large Data-Acquisition Network in a Simulation Framework". In: *2015 IEEE International Conference on Cluster Computing*. CLUSTER. New York: IEEE, 2015, pp. 809–816. DOI: 10.1109/CLUSTER.2015.137.

[24] T. Colombo. "Data-flow Performance Optimisation on Unreliable Networks: the ATLAS Data-Acquisition Case". In: *Journal of Physics: Conference Series* 608.1 (2015), p. 012005. ISSN: 1742-6596. DOI: 10.1088/1742-6596/608/1/012005.

[25] T. Colombo et al. "Optimizing the data-collection time of a large-scale data-acquisition system through a simulation framework". In: *The Journal of Supercomputing* 72.12 (2016), pp. 4546–4572. ISSN: 0920-8542, 1573-0484. DOI: 10.1007/s11227-016-1764-1.

[26] P. Devkota and A. Reddy. "Performance of Quantized Congestion Notification in TCP Incast Scenarios of Data Centers". In: *2010 IEEE International Symposium on Modeling, Analysis Simulation of Computer and Telecommunication Systems*. MASCOTS. New York: IEEE, 2010, pp. 235–243. DOI: 10.1109/MASCOTS.2010.32.

[27] "LHC Machine". In: *Journal of Instrumentation* 3.8 (2008). Ed. by L. Evans and P. Bryant, S08001. ISSN: 1748-0221. DOI: 10.1088/1748-0221/3/08/S08001.

[28] K. R. Fall and W. R. Stevens. *TCP/IP Illustrated, Volume 1: The Protocols*. Second. Addison-Wesley Professional, 2011. 1056 pp. ISBN: 978-0-13-280820-0.

[29] A. Firoozshahian et al. "Efficient, Fully Local Algorithms for CIOQ Switches". In: *26th IEEE International Conference on Computer Communications*. INFOCOM. New York: IEEE, 2007, pp. 2491–2495. DOI: 10.1109/INFCOM.2007.307.

[30] S. Floyd et al. *The NewReno Modification to TCP's Fast Recovery Algorithm*. Internet Request for Comments 6582. RFC Editor, 2012. URL: http://www.rfc-editor.org/rfc/rfc6582.txt (visited on 04/15/2016).

[31] S. Ha and I. Rhee. "Taming the elephants: New TCP slow start". In: *Computer Networks* 55.9 (2011), pp. 2092–2110. ISSN: 1389-1286. DOI: 10.1016/j.comnet.2011.01.014.

[32] S. Ha, I. Rhee, and L. Xu. "CUBIC: A New TCP-friendly High-speed TCP Variant". In: *SIGOPS Oper. Syst. Rev.* 42.5 (2008), pp. 64–74. ISSN: 0163-5980. DOI: 10.1145/1400097.1400105.

[33] Hewlett-Packard Development Company. *ProCurve 6600 Manuals*. URL: http://www.hp.com/rnd/support/manuals/6600dc.htm (visited on 02/21/2016).

[34] *IEEE Standard for Ethernet*. IEEE Std 802.3-2015. New York: IEEE, 2016. ISBN: 978-1-5044-0078-7.

[35] *IEEE Standard for Local and metropolitan area networks–Bridges and Bridged Networks*. IEEE Std 802.1Q-2014. New York: IEEE, 2014. ISBN: 978-0-7381-9433-2.

[36] *INET Framework*. URL: https://inet.omnetpp.org/ (visited on 03/23/2016).

[37] *Internet Protocol*. Internet Request for Comments 791. RFC Editor, 1981. URL: https://www.rfc-editor.org/rfc/rfc791.txt.

[38] V. Jacobson. "Congestion Avoidance and Control". In: *Symposium Proceedings on Communications Architectures and Protocols*. SIGCOMM. New York: ACM, 1988, pp. 314–329. ISBN: 978-0-89791-279-2. DOI: 10.1145/52324.52356.

[39] S. T. Jansen. "Network Simulation Cradle". Thesis. The University of Waikato, 2008. URL: http://researchcommons.waikato.ac.nz/handle/10289/3287 (visited on 03/23/2016).

[40] G. Jereczek et al. "A lossless switch for data acquisition networks". In: *2015 IEEE 40th Conference on Local Computer Networks*. LCN. New York: IEEE, 2015, pp. 552–560. DOI: 10.1109/LCN.2015.7366370.

[41] A. Köpke et al. "Simulating Wireless and Mobile Networks in OMNeT++ the MiXiM Vision". In: *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*. Simutools. Brussels: ICST, 2008, 71:1–71:8. ISBN: 978-963-9799-20-2. URL: http://dl.acm.org/citation.cfm?id=1416222.1416302 (visited on 03/23/2016).

[42] C. Lee et al. “DX: Latency-Based Congestion Control for Datacenters”. In: *IEEE/ACM Transactions on Networking* 25.1 (2017), pp. 335–348. ISSN: 1063-6692. DOI: 10.1109/TNET.2016.2587286.

[43] LHCb Collaboration. “The LHCb Detector at the LHC”. In: *Journal of Instrumentation* 3.8 (2008), S08005. ISSN: 1748-0221. DOI: 10.1088/1748-0221/3/08/S08005.

[44] D. L. Mills. *Executive Summary: Computer Network Time Synchronization*. 2012. URL: https://www.eecis.udel.edu/~mills/exec.html (visited on 09/12/2016).

[45] R. Mittal et al. “TIMELY: RTT-based Congestion Control for the Datacenter”. In: *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. SIGCOMM. New York: ACM, 2015, pp. 537–550. ISBN: 978-1-4503-3542-3. DOI: 10.1145/2785956.2787510.

[46] D. Nagle, D. Serenyi, and A. Matthews. “The Panasas ActiveScale Storage Cluster: Delivering Scalable High Bandwidth Storage”. In: *Proceedings of the 2004 ACM/IEEE Conference on Supercomputing*. SC. Washington: IEEE Computer Society, 2004, p. 53. ISBN: 978-0-7695-2153-4. DOI: 10.1109/SC.2004.57.

[47] A. Núñez et al. “SIMCAN: A SIMulator Framework for Computer Architectures and Storage Networks”. In: *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*. Simutools. Brussels: ICST, 2008, 73:1–73:8. ISBN: 978-963-9799-20-2. URL: http://dl.acm.org/citation.cfm?id=1416222.1416304 (visited on 03/23/2016).

[48] J. Pequenao. *Computer generated image of the ATLAS calorimeter*. 2008. URL: http://cdsweb.cern.ch/record/1095927.

[49] J. Pequenao. *Computer generated image of the ATLAS inner detector*. 2008. URL: http://cdsweb.cern.ch/record/1095926.

[50] J. Pequenao. *Computer generated image of the ATLAS Muons subsystem*. 2008. URL: http://cdsweb.cern.ch/record/1095929.

[51] J. Pequenao. *Computer generated images of the Pixel, part of the ATLAS inner detector*. 2008. URL: http://cdsweb.cern.ch/record/1095925.

[52] J. Perry et al. “Fastpass: A Centralized ”Zero-queue” Datacenter Network”. In: *Proceedings of the 2014 ACM Conference on SIGCOMM*. SIGCOMM. New York: ACM, 2014, pp. 307–318. ISBN: 978-1-4503-2836-4. DOI: 10.1145/2619239.2626309.

[53] A. Phanishayee et al. “Measurement and Analysis of TCP Throughput Collapse in Cluster-based Storage Systems”. In: *Proceedings of the 6th USENIX Conference on File and Storage Technologies*. FAST’08. Berkeley: USENIX Association, 2008, 12:1–12:14. URL: http://dl.acm.org/citation.cfm?id=1364813.1364825 (visited on 02/19/2016).

[54] R. van der Pol. *TRILL and IEEE 802.1aq Overview*. LHCONE Architecture Document. SARA, 2012.

[55] M. E. Pozo Astigarraga. “Evolution of the ATLAS Trigger and Data Acquisition System”. In: *Journal of Physics: Conference Series* 608.1 (2015), p. 012006. ISSN: 1742-6596. DOI: 10.1088/1742-6596/608/1/012006.

[56] I. Psaras and V. Tsaoussidis. “The TCP Minimum RTO Revisited”. In: *NETWORKING 2007. Ad Hoc and Sensor Networks, Wireless Networks, Next Generation Internet*. Ed. by I. F. Akyildiz et al. Lecture Notes in Computer Science 4479. Berlin Heidelberg: Springer, 2007, pp. 981–991. ISBN: 978-3-540-72605-0 978-3-540-72606-7. DOI: 10.1007/978-3-540-72606-7_84.

[57] K. K. Ramakrishnan, S. Floyd, and D. L. Black. *The Addition of Explicit Congestion Notification (ECN) to IP*. Internet Request for Comments 3168. RFC Editor, 2001. URL: https://www.rfc-editor.org/rfc/rfc3168.txt.

[58] *Requirements for IP Version 4 Routers*. Internet Request for Comments 1812. RFC Editor, 1995. URL: https://www.rfc-editor.org/rfc/rfc1812.txt.

[59] T. Reschka et al. “Enhancement of the TCP module in the OMNeT++/INET framework”. In: *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*. SIMUTools ’10. Brussels: ICST, 2010, 24:1–24:1. ISBN: 978-963-9799-87-5. DOI: 10.4108/ICST.SIMUTOOLS2010.8834.

[60] M. Sargent et al. *Computing TCP’s Retransmission Timer*. Internet Request for Comments 6298. RFC Editor, 2011. URL: http://www.rfc-editor.org/rfc/rfc6298.txt (visited on 02/19/2016).

[61] C. E. Spurgeon and J. Zimmerman. *Ethernet: The Definitive Guide*. 2nd. O’Reilly Media, 2014. 508 pp. ISBN: 978-1-4493-6184-6.

[62] K. Tan et al. “A Compound TCP Approach for High-Speed and Long Distance Networks”. In: *Proceedings of the 25th IEEE International Conference on Computer Communications*. INFOCOM. 2006, pp. 1–12. DOI: 10.1109/INFOCOM.2006.188.

[63] *Transmission Control Protocol*. Internet Request for Comments 793. RFC Editor, 1981. URL: https://www.rfc-editor.org/rfc/rfc793.

[64] A. Varga. "OMNeT++". In: *Modeling and Tools for Network Simulation*. Ed. by K. Wehrle, M. Güneş, and J. Gross. Berlin Heidelberg: Springer, 2010, pp. 35–59. ISBN: 978-3-642-12330-6 978-3-642-12331-3. DOI: 10.1007/978-3-642-12331-3_3.

[65] V. Vasudevan et al. "Safe and Effective Fine-grained TCP Retransmissions for Datacenter Communication". In: *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*. SIGCOMM. New York: ACM, 2009, pp. 303–314. ISBN: 978-1-60558-594-9. DOI: 10.1145/1592568.1592604.

[66] D. X. Wei et al. "FAST TCP: Motivation, Architecture, Algorithms, Performance". In: *IEEE/ACM Trans. Netw.* 14.6 (2006), pp. 1246–1259. ISSN: 1063-6692. DOI: 10.1109/TNET.2006.886335.

[67] H. Wu et al. "ICTCP: Incast Congestion Control for TCP in Data-center Networks". In: *IEEE/ACM Trans. Netw.* 21.2 (2013), pp. 345–358. ISSN: 1063-6692. DOI: 10.1109/TNET.2012.2197411.

[68] P. Yebenes et al. "Towards Modeling Interconnection Networks of Exascale Systems with OMNet++". In: *2013 21st Euromicro International Conference on Parallel, Distributed and Network-Based Processing*. PDP. 2013, pp. 203–207. DOI: 10.1109/PDP.2013.36.