Maynooth University
National University of Ireland Maynooth

Software Switching for High Throughput Data Acquisition Networks

A dissertation submitted for the degree of
Doctor of Philosophy

By:

Grzegorz Jereczek

Under the supervision of:
Dr. David Malone
Dr. Giovanna Lehmann Miotto (CERN)

Head of Department:
Prof. Ken Duffy

Hamilton Institute
Maynooth University

June 19, 2017

Grzegorz Jereczek: *Software Switching for High Throughput Data Acquisition Networks,* © June 19, 2017
email: grzegorz.jereczek@gmail.com
Hamilton Institute, Maynooth University, Maynooth

CONTENTS

LIST OF FIGURES

LIST OF ALGORITHMS

DECLARATION

I hereby declare that I have produced this manuscript without the prohibited assistance of any third parties and without making use of aids other than those specified.

The thesis work was conducted from October 1, 2013 to June 19, 2017 under the supervision of Dr. David Malone and Dr. Giovanna Lehmann Miotto (CERN) in Hamilton Institute, Maynooth University. This research project has been supported by a Marie Curie Early European Industrial Doctorates Fellowship of the European Community's Seventh Framework Programme under contract number *PITN-GA-2012-316596-ICE-DIP*.

Geneva, Switzerland, June 19, 2017

Grzegorz Jereczek

ACKNOWLEDGEMENTS

First of all, I would like to thank my two supervisors, Dr. Giovanna Lehmann Miotto at CERN and Dr. David Malone at Maynooth University, for their continuous support, guidance, encouragement, and motivation, but also friendliness. Giovanna led me into the field of data acquisition networks and offered her great technical expertise at every step. David has assisted me with the academic aspect of this work and shared his thorough knowledge of networks. My work would have been impossible without their great help. They also carried out the proofreading work and provided constant feedback to improve the research papers and this manuscript. I am very thankful that I had the chance to meet them.

I would also like to express deepest gratitude to Mr. Mirosław Walukiewicz from Intel Poland. His invaluable technical advice in the field of networks and software packet processing as well as numerous discussions with him were essential for me to focus on the right direction. I also appreciate all the support and feedback that I received during my secondment at Intel. Without the genuine commitment of Mr. Krzysztof Krzywdziński and continuous support of Mr. Mariusz Linda it would be impossible to conduct the crucial experiments for this thesis. I cannot forget about Mr. Andrzej Jaszcza, who first trusted me when starting my journey through computer networks.

I am grateful to the members of the ATLAS TDAQ group at CERN for their support in preparation of the experiments, technical discussions, and valuable reviews of the research papers, with special mentions for Mr. Mikel Eukeni Pozo Astigarraga, Dr. Wainer Vandelli, Dr. Tommaso Colombo, and Dr. Silvia-Maria Fressard-Batraneanu. I would like to thank the ICE-DIP team, especially Mr. Andrzej Nowak, Dr. Bob Jones, and Dr. Seamus Hegarty, for the continuous support.

Special thanks to the European Commission for this opportunity that changed my life in so many, often unpredictable, ways. This research project has been supported by a Marie Curie Early European Industrial Doctorates Fellowship of the European Community's Seventh Framework Programme under contract number *PITN-GA-2012-316596-ICE-DIP*.

At this point, I am obliged to give thanks to Prof. Thomas Zwick from Karlsruhe Institute of Technology, who helped me to build my self-confidence and who inspired me for the career in research.

Last, but not least, I would like to thank my family, for supporting me in everything I have done in my life. My thanks are addressed in particular to my parents for their unconditional support and sacrifices throughout my education that started in Poland and went on to Germany, Switzerland, and Ireland. Thank you, sister, for always making

sure I am safe and sound. And to my wife, Anna, and daughter, Maja, which came to my life in the last year of my PhD programme, for putting up with my terrible working hours and still continuing to support me. Anna, you have never missed the opportunity to tell me *well done* and you have made me feel pride in my achievements. A warm *thank you* goes also to all my friends, including the ICE-DIP fellows, who are there for me all the time.

ABSTRACT

The bursty many-to-one communication pattern, typical for data acquisition systems, is particularly demanding for commodity TCP/IP and Ethernet technologies. The problem arising from this pattern is widely known in the literature as *incast* and can be observed as TCP throughput collapse. It is a result of overloading the switch buffers, when a specific node in a network requests data from multiple sources. This will become even more demanding for future upgrades of the experiments at the Large Hadron Collider at CERN. It is questionable whether commodity TCP/IP and Ethernet technologies in their current form will be still able to effectively adapt to bursty traffic without losing packets due to the scarcity of buffers in the networking hardware. This thesis provides an analysis of TCP/IP performance in data acquisition networks and presents a novel approach to incast congestion in these networks based on software-based packet forwarding.

Our first contribution lies in confirming the strong analogies between the TCP behaviour in data acquisition and datacenter networks. We also provide experimental evaluation of different proposals from the datacenter environment for application in data acquisition to improve performance and reduce buffer requirements.

The second contribution lies in the design and experimental evaluation of a data acquisition network that is based on software switches. Performance has traditionally been the challenge of this approach, but this situation changes with modern server platforms. High performance load balancers, proxies, virtual switches and other network functions can be now implemented in software and not limited to specialised commercial hardware, thus reducing cost and increasing the flexibility.

We first design and optimise a software-based switch with a dedicated, throughput-oriented buffering mechanism for data acquisition. Our experimental results indicate that it performs significantly better than some typical Ethernet switches under heavy congestion. The optimised software switch with large packet buffer reaches maximum bandwidth and completely avoids throughput degradation typical for hardware switches that suffer from high packet drop counts.

Furthermore, we evaluate the scalability of the system when building a larger topology of interconnected software switches. We highlight aspects such as management, costs, port density, load balancing, and failover. In this context, we discuss the usability of software-defined networking technologies, Open vSwitch Database and OpenFlow, to centrally manage and optimise a data acquisition network. We have built an IP-only parallel leaf-spine network consisting of eight software switches running on separate physical servers as a demonstrator.

GLOSSARY

Term	Meaning
ALICE	A Large Ion Collider Experiment
API	Application Programming Interface
ARP	Address Resolution Protocol
ASIC	Application Specific Integrated Circuit
ATLAS	A Toroidal LHC ApparatuS
BDP	Bandwidth-Delay Product
bps	bits per second
CERN	The European Organisation for Nuclear Research
CMS	The Compact Muon Solenoid
COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit
DAQ	Data Acquisition
DCB	Datacenter Bridging
DCM	Data Collection Manager
DCN	Datacenter Network
DCTCP	Datacenter TCP
DDR	Double Data Rate
DIATCP	Deadline and Incast Aware TCP
DMA	Direct Memory Access
DMI	Direct Media Interface
DPDK	Data Plane Development Kit
DRAM	Dynamic Random Access Memory
ECMP	Equal Cost Multipath
ETS	Enhanced Transmission Selection
FIFO	First-In-First-Out
FSB	Front-Side Bus
GbE	Gigabit Ethernet
HLT	High Level Trigger
HLTSV	HLT Supervisor
HoL	Head-of-Line (blocking)
HPC	High Performance Computing

ICE-DIP	The Intel-CERN European Doctorate Industrial Program
ICT	Information and Communication Technologies
ICTCP	Incast Congestion Control for TCP
IETF	Internet Engineering Task Force
IIO	Integrated I/O
IMC	Integrated Memory Controller
IoT	Internet of Things
IP	Internet Protocol
L1r	Level-1 rate (events)
LAG	Link Aggregation Group
LAN	Local Area Network
LHC	The Large Hadron Collider
LHCb	The Large Hadron Collider beauty experiment
MAC	Media Access Control
MSS	Maximum Segment Size
MTU	Maximum Transmission Unit
NFV	Network Functions Virtualisation
NIC	Network Interface Controller
NUMA	Non-Uniform Memory Access
OF	OpenFlow
OVS	Open vSwitch
OVSDB	Open vSwitch Database
PC	Personal Computer
PCH	Platform Controller Hub
PCI	Peripheral Component Interconnect
PCIe	Peripheral Component Interconnect Express
PFC	Priority-based Flow Control
pps	packets per second
QCN	Quantized Congestion Notification
QoS	Quality-of-Service
QPI	QuickPath Interconnect
ROS	Readout System
RSS	Receive Side Scaling
RTT	Round-Trip Time
SDN	Software-Defined Networking
SRAM	Static Random Access Memory
SSH	Secure Shell

STP	Spanning Tree Protocol
TCP	Transmission Control Protocol
TCP RTO	TCP Retransmission Timeout
TCP SACK	TCP Selective Acknowledgement
TDAQ	Trigger and Data Acquisition
TLB	Translation Lookaside Buffer
ToR	Top of Rack (switch)
UDP	User Datagram Protocol
VLAN	Virtual LAN
VM	Virtual Machine
VOQ	Virtual Output Queueing

1 INTRODUCTION

This thesis is about the optimisation of data acquisition networks for large experiments based on commodity TCP/IP and Ethernet technologies. In this chapter, we present background information and some basic concepts for the topics to be discussed in the thesis with the review of the related literature. We also give the motivations behind our work highlighting the research objectives and contributions with an overview of the material presented in the following chapters.

1.1 BACKGROUND INFORMATION

1.1.1 *CERN and the LHC*

Over the centuries humanity has been exploring the laws of nature. A significant role in this exploration is played nowadays by scientists at CERN [32], the European Organisation for Nuclear Research, where the basic constituents of matter, the elementary particles [22], are studied. The particles approach speeds close to the speed of light in accelerators and then are made to collide. The effects of these collisions are observed by detectors, recorded by high throughput data acquisition systems, and finally analysed by physicists around the world seeking answers for questions about our universe. CERN provides the necessary instruments for this research like particle accelerators, detectors, data acquisition systems, or computing infrastructure.

On 10 September 2008 the world's largest and most powerful particle accelerator to date, the Large Hadron Collider (LHC) [53], started its operation at CERN. It consists of a 27-kilometre ring and four particle detectors, where particle beams travelling in opposite directions collide:

- ALICE - A Large Ion Collider Experiment [161],
- ATLAS - A Toroidal LHC ApparatuS [163],
- CMS - The Compact Muon Solenoid [164],
- LHCb - The Large Hadron Collider beauty experiment [166].

The overall view of the LHC and the experiment's detectors is illustrated in Figure 1.1.

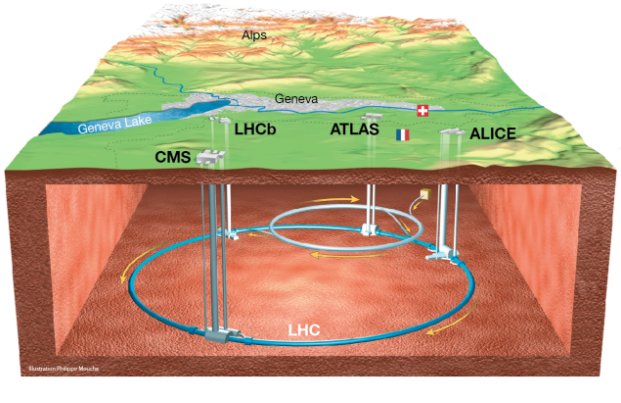


Figure 1.1: Overall view of the LHC and the 4 LHC detectors: ALICE, ATLAS, CMS and LHCb [53].

During the LHC's first run the existence of the so called Higgs boson was confirmed by the ATLAS and CMS experiments. As a result, Francois Englert and Peter W. Higgs were awarded the Nobel Prize in Physics in 2013 for their theoretical work and prediction of the existence of such a particle. The second run of the LHC started in 2015, after it had been upgraded to operate at higher energies and collision rates. This run will continue until 2018, when another increase in rates and energies is going to be pursued. All scheduled LHC upgrades ranging beyond 2020 require significant technological advances in the entire LHC infrastructure. Higher energies and rates are, however, essential in the search for new discoveries.

In 2013 the Intel-CERN European Doctorate Industrial Program (ICE-DIP) [165] was started by the CERN openlab [33] to research and develop new technologies in the domain of high throughput online data acquisition facing the future upgrades of the LHC facilities — the significant increases in the data rates produced by the experiments in particular. The public-private partnership of the ICE-DIP project makes it possible to evaluate the most innovative concepts available in the ICT industry in the demanding context of the LHC data acquisition systems. This Ph.D. is the result of my work within one of the ICE-DIP work packages, the objective of which was to *assemble a cost effective, high bandwidth data acquisition network capable of multi-terabit lossless throughput using commodity components.*

1.1.2 *Trigger and Data Acquisition Systems*

Data Acquisition (DAQ) is the process of sampling and recording data representing real-world physical signals [105]. The source of these data can be a wide variety of instruments like sensors, detectors, antennas, or telescopes. Large-scale experiments, like the four LHC detectors, produce increasingly high volumes of data. For example, in the first

run of the LHC (called Run 1, 2009-2013) its nominal collision rate was 40 MHz, which means that particles collided in the detectors every 25 ns. In case of the ATLAS experiment, the nominal size of a snapshot of a *collision event*[1] produced by the detector was 1.5 MB on average, which results in roughly 500 Tbit of data produced by the detector every second [117]. It is obvious that not all the data can be acquired and saved to permanent storage, from where they can be later retrieved for the off-line analysis. Moreover, these numbers continuously increase with the upgrades of the LHC. On the other hand, many collision events are not very interesting from the physics point of view. Therefore on-line selection procedures, *triggers*, are used to select only the interesting collisions for further analysis, reducing the total number of events recorded permanently. Trigger systems consist of a combination of dedicated high-speed electronics and software [22, 118]. Data acquisition and trigger systems are often jointly referred to as Trigger and Data Acquisition (TDAQ). An overview of the TDAQ systems of the four LHC experiments in the LHC's first run with the outlook for the future upgrades is available in [118].

Many functions of the LHC's TDAQ systems are realised by dedicated high-speed electronics, which is designed to sustain the amounts and rates of data produced by the experiments. There exists, however, a general desire to build the TDAQ systems with commercial off-the-shelf (COTS) equipment. It simplifies development, configuration, and maintenance, but also significantly reduces costs. Custom electronics are replaced with high-performance commodity processors, efficient software, and commodity networks whenever possible [118]. Furthermore, the ever-increasing usage of the Internet, and the advent of data-intensive services (*Big Data*) have started generating and moving quantities of data in the order of hundreds of PB each month. In [67] the LHC along with Internet search and content distribution or business data collection were given as an example of *data-intensive computing*, in which massive amounts of data must be processed to facilitate understanding of complex problems. The technologies developed and used by the IT industry to process these amounts of data are becoming commodity components, and therefore could be considered as viable competitors to purpose-built electronics in experiments like the LHC [48]. The same conclusions have been made in the context of the LHCb experiment by [148].

1.1.2.1 *Data acquisition networks*

One of the key components of TDAQ systems in distributed large-scale experiments is a network, called the data acquisition network. It collects the outputs from all the instruments to allow reconstruction and analysis of a physical process (Figure 1.2) [40]. At the LHC, depend-

1 Events in the LHC nomenclature are particle collisions inside of the LHC and their physical "fingerprints" are recorded by a detector.

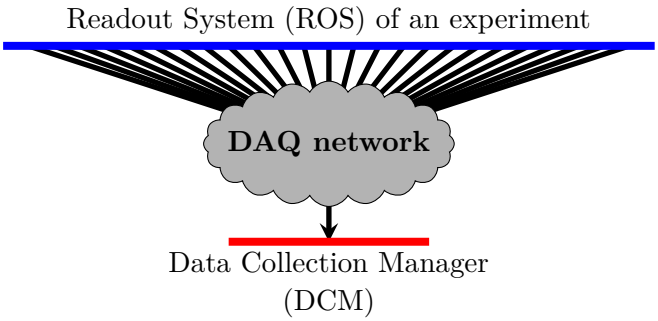


Figure 1.2: Many-to-one communication in a data acquisition network. Data originating from the experiment's instruments are sent over a network to data collectors in the event building/filtering farm for further processing. Only one collector is drawn for clarity. Depending on the size of an experiment, hundreds or thousands of independent collectors are used.

ing on the experiment, a DAQ network can also be referred to as an event building network. Depending on the size of an experiment a DAQ network must provide required levels of data bandwidth, latency, and reliability. Any data loss implies incomplete reconstruction of the physical process, and, in effect, oversight or disqualification of potential discoveries.

These requirements are often difficult to achieve because of the traffic pattern specific to data acquisition. As can be already inferred from Figure 1.2, many-to-one communication is associated with TDAQ systems, often accompanied by high burstiness of data coming from the readout system [118]. These conditions lead to so called *incast* [114] congestion — a problem that is present also in the IT industry, especially in datacenter networks (DCNs) [87]. It occurs when multiple nodes respond with data synchronously to a single requester. It has been observed that these responses, although not very large on average, suffer from a large packet loss rate [170]. The scale of the problem increases with the number of the responders, i.e. the size of the readout system in the case of DAQ networks. The details of the DAQ networks of the four LHC experiments will be reviewed in Section 2.1.

1.1.3 *The ATLAS detector*

ATLAS [163] is one of the two general-purpose detectors at the LHC, which investigate a wide range of physics. The scientific goals of ATLAS are the same as the ones of the second detector, CMS [164], but they use different technical solutions to achieve them. Following the discovery of the Higgs boson (see Section 1.1.1), in-depth investigation of its properties will be possible with further data. The experiment can also shed more light on theories like the extra dimensions of space, dark matter, or register completely new, unpredicted phenomena.

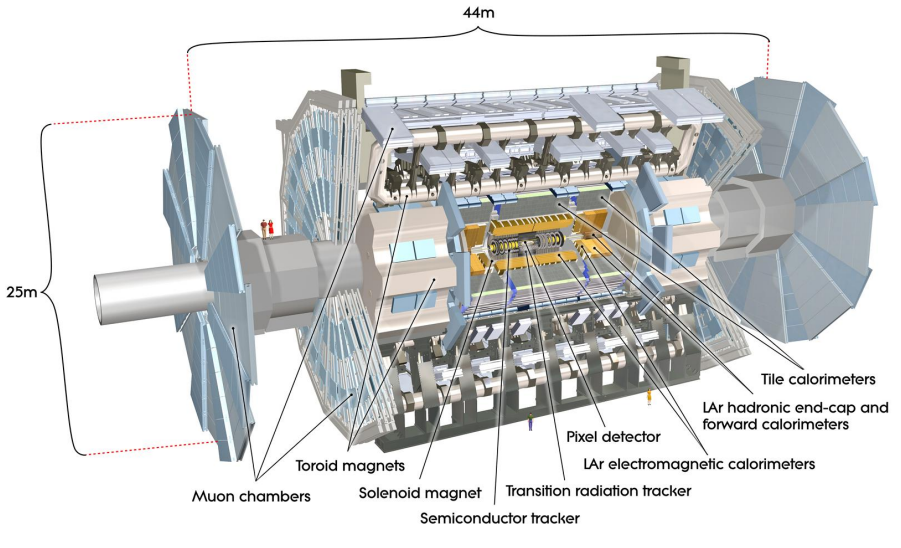


Figure 1.3: The ATLAS detector [163].

The ATLAS detector is presented in Figure 1.3. The particle beams accelerated in the LHC collide at the centre of the detector. New particles resulting from these collisions fly out in all directions. ATLAS records a set of their properties like paths through the detector, energies, or momenta. Different detecting subsystems measure specific sets of these properties.

As already described in Section 1.1.2, the ATLAS detector creates an enormous dataflow. An advanced TDAQ system is required to identify only interesting collision events and record them permanently. The outline of the system for the the 2015–2018 data-taking period is presented in Figure 1.4. The ATLAS multi-level triggers [162] are implemented both in dedicated hardware and on a commodity computer farm. The latter applies the most complex event selection algorithms in software, which are generically named as High Level Trigger (HLT). On the other hand, the the first-level hardware triggers use the simplest criteria to reject obviously uninteresting events as fast as possible, before they reach the experiment's readout system (ROS). The decision is often made independently in various parts of the detector.

HLT is a PC-based, large distributed system, interconnected through a data acquisition network based on Ethernet and TCP/IP technologies. Most of the events are already discarded based only on partial event data retrieved from ROS (*event filtering*), but in order to finally accept an event as interesting, all data belonging to a particular event must be delivered to an HLT node (*event building*). In general, event filtering and building functions can take place on different nodes, but in the LHC Run 2 (2015-2018) the same nodes are used in the case of ATLAS. The DAQ network was designed to sustain a throughput of several 10 GB/s

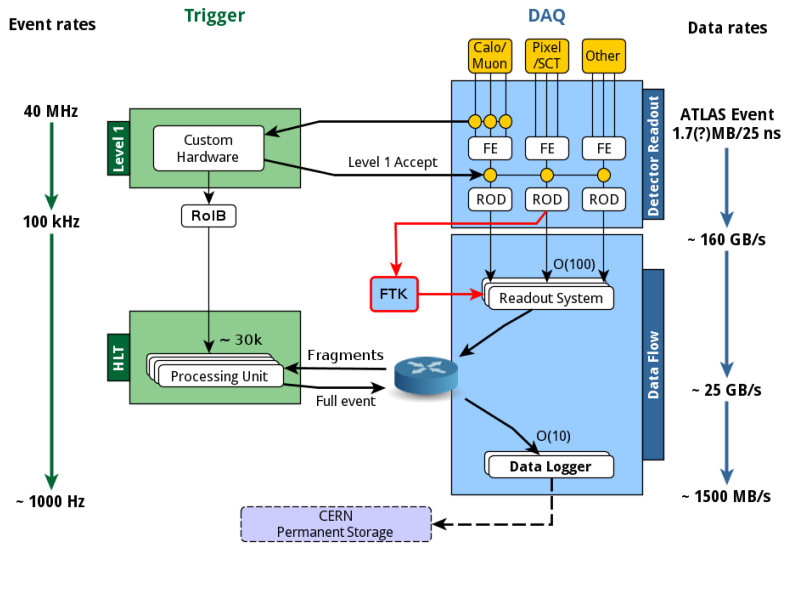


Figure 1.4: The DAQ/HLT system of the ATLAS detector for Run 2 (2015-2018) [39].

for the LHC's second run, and transport data from 100 readout nodes to approximately 30000 CPU cores executing HLT algorithms [130]. The bandwidth requirement will further increase in Run 3 (2020-2023) of the LHC [17], and beyond.

Because of the scale of the experiment and the resulting requirements on the network system, the ATLAS DAQ network (see Figure 1.5) is used throughout this work as a case study to research new solutions optimising commodity networks for use in data acquisition systems. The conclusions are, however, not limited to ATLAS but are applicable to many other networks susceptible to the problems arising from the many-to-one communication pattern as well as other TDAQ systems. The ATLAS DAQ network, having very demanding traffic characteristics, is a good environment to evaluate candidate technologies.

1.1.4 *Networking on general-purpose computers*

One avenue that could be explored to enhance the DAQ capabilities in ATLAS and other large experiments is software packet processing on general purpose servers. This approach has recently become a real alternative for specialised commercial networking products thanks to new capabilities of commodity hardware and user space networking frameworks like the Intel Data Plane Development Kit (DPDK) [80]. High performance load balancers, proxies, virtual switches and other network functions can be now implemented in software and not limited to specialised commercial hardware, thus reducing cost and increasing

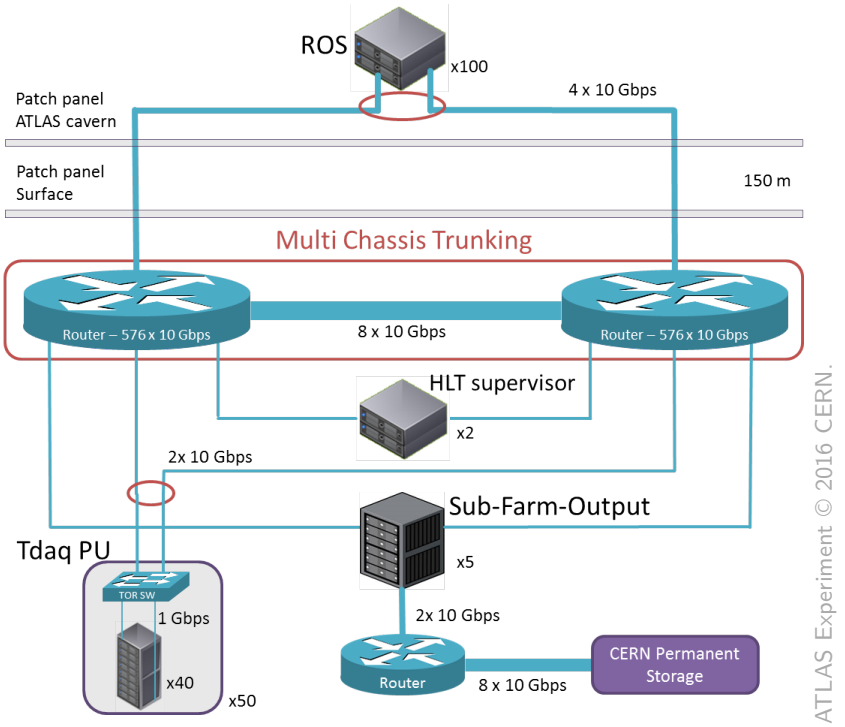
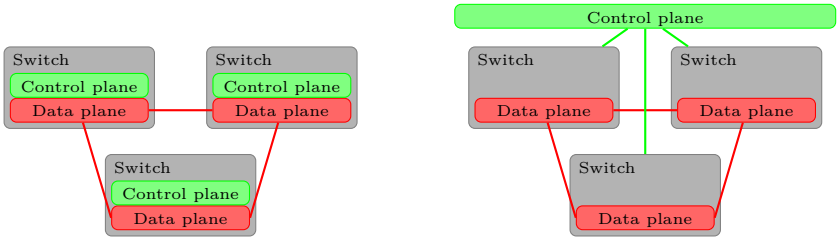


Figure 1.5: Network architecture of the ATLAS DAQ/HLT system for Run 2 (2015-2018) [39].

flexibility. The presence of software-based commercial networking products like Brocade Vyatta Virtual Router [173] or offerings of 6WIND [2] prove the viability of this approach. With a combination of higher flexibility and features for a target application, a solution tuned for a particular network could be pursued.

Together with the Software-Defined Networking (SDN) paradigm [124], a completely new perspective to build, manage, and optimise networks opens up. In SDN, the networking control and forwarding planes (also called data plane) are physically decoupled and the network intelligence is logically centralised in an SDN controller as depicted in Figure 1.6. This controller has a global view of the entire network and is responsible for maintaining all of the network paths and programming each of the devices in the network [113]. The OpenFlow protocol [126], originally proposed in [108], is used for communication between the controller and the devices. This concept also contrasts with traditional networks with a distributed control plane, in which network devices are comprised of both: a data plane, being switch fabric connecting ports, and a control plane that is the brain of a device, implementing various protocols [113]. SDN significantly improves flexibility because dedicated algorithms can be easily integrated in software within the controller framework.



Traditional Network Software-Defined Network

Figure 1.6: In an SDN network control logic is decoupled from the forwarding hardware in a centralised controller in contrast to traditional networks, in which devices implement both and the control is distributed across them.

1.1.5 *Summary*

Data acquisition systems of large scale experiments, like those of the LHC, require high bandwidth and reliable networks. Many-to-one communication is the typical pattern for these systems and it is at the source of a network congestion problem. The latter will become more severe for future upgrades of the detectors at the LHC. There are reasons to evaluate software switching on COTS servers and SDN technologies to provide reliable transport in congested conditions. The flexibility of design in software and performance of modern computer platforms are a strong basis on which to build a DAQ-dedicated network.

1.2 MOTIVATION

This work presents methods for optimisation of DAQ networks for high throughput based on commodity TCP/IP and Ethernet technologies. They were chosen as underlying networking technologies due to the general desire to use commodity technologies in TDAQ systems, as we explained already in the previous section. The cheap, flexible and easy to maintain switched Ethernet has become the world's most ubiquitous wired computer network. It also interplays easily with the TCP/IP protocol stack, which is dominant in the Internet [157]. Ethernet constantly evolves and the speeds continue to climb. 4x10GbE (10 Gigabit Ethernet) ports are now becoming standard in the servers available on the market. 40GbE/100GbE is already standardised in IEEE 802.3ba [76] with products already available on the market and 400GbE is under development [74]. These trends show that Ethernet will remain capable of sustaining the data rates required by the upgrades of the LHC experiments, which were highlighted on the example of ATLAS in Section 1.1.3.

The incast pattern is, however, particularly demanding for Ethernet switches and TCP flows. The ports connected to receiving data collectors are overcommitted by the data sent from many readout units.

Switches with insufficient buffers drop some or most of the packets. Since the problem is systematic, built-in TCP retransmission mechanisms are not a solution [118]. In an ideal situation, the network should adapt to the traffic and avoid congestion before it happens. One way of achieving this is to include enough buffering space within the network, to accommodate traffic spikes and bursts without the need for discarding data [139]. For large experiments, like the ones of the LHC, it means particularly large packet buffers. Only expensive, high-end switches or routers can be taken into consideration [118], but they cannot be regarded as commodity equipment due to their cost. Furthermore, the widely used rule-of-thumb for large gigabyte buffers in routers has been challenged [63, 172], so a new trend to decrease the amount of buffering in the networks is now more likely. On the other hand, the challenge of congestion control in high speed Ethernet, especially in case of data acquisition, will become even more critical with the ever increasing link speeds.

It becomes compelling to consider the possibility of designing a data acquisition network mixing switches with servers that act themselves as networking devices — *software switches*. This concept reaches back to the first generation of network bridges in 1980s, when some of them were built using general-purpose computers with standard network interface controllers (NICs) [150]. The switch designs then started to progress through increasing levels of integration — a process that never stops. More and more functions can be integrated into a single silicon chip. Performance, among other things, is improved with higher levels of integration. Still, the internal design of modern switches and routers resembles typical computer architectures. Both network devices and general purpose computers are built with components such as CPUs, memories, buses, and network interfaces [107, 122, 150]. Furthermore, commonly used memory types are also the same [122, 128]:

- fast and expensive Static Random Access Memory (SRAM),
- slower and cheaper Dynamic Random Access Memory (DRAM).

The reasons to reach back to the original generations of switches are twofold. First, flexibility when using commercial devices is limited. As we described in the previous paragraph, there exist some rules-of-thumb for the sizes of memories in commercial networking devices. Even the rare large-buffer switches have a fixed size of this memory and the queueing mechanisms can be adjusted only within the limits allowed by the vendors. Furthermore, many features available on the commercial networking devices are not required in DAQ networks, unnecessarily increasing their cost. Second, the specifications of modern server boards promise nowadays adequate levels of performance and buffering. A high performance switching application on a modern server with multiple NICs connected over the PCI Express (PCIe) bus gives the opportunity to substantially extend the buffering capabilities (constrained solely by

the amount of DRAM memory) to accommodate the many-to-one data bursts and to perform flexible optimisations tailored to the DAQ traffic patterns. The evolution of the performance of the switches built with commodity servers is presented in Section 2.3.

In this work we study whether such a specialised switch could operate without packet losses in DAQ systems while maintaining high throughput, avoiding incast congestion without any controls on the injected traffic, beyond that arising naturally in a DAQ application.

Depending on the size of a DAQ system, a number of interconnected software switches is necessary to provide full connectivity. Today, the classical architecture with two deep-buffered large routers at the core of the network interconnects approximately 2000 nodes in the ATLAS experiment [39]. It is of interest to study a topology based on specialised software switches and possibly traditional top of rack (ToR) switches with small buffers as an alternative for the classical architecture, if the required port density is guaranteed. No less important are aspects as administration, configuration, fault tolerance and load balancing. SDN gives the opportunity to integrate a network controller of a DAQ network with an already existing run controller of an experiment thanks to the protocols like OpenFlow. Motivated by the recent developments in the SDN domain we consider what advantages it provides in the context of data acquisition networks. It is also of interest to explore how the entire solution based on SDN, TCP/IP protocol stack, and packet processing in software can be optimised as a whole to provide the maximum performance of a DAQ network.

1.3 THESIS OVERVIEW AND RESEARCH OBJECTIVES

This thesis is organised as follows. In Chapter 2 we present the literature review with a focus on current state of networking in large TDAQ systems. We discuss typical aspects and problems, but special attention is given to the generic approaches to incast avoidance in many-to-one communication applications. We also present current topics in software packet processing and SDN, which form the basis building blocks for our study.

In Chapter 3 we discuss the most relevant properties of typical data acquisition networks for large experiments and extend the description of the ATLAS DAQ system from this chapter. We define basic requirements, metrics to quantify the performance and evaluation methodology that are used throughout this thesis. At this point, it is worth noting that our research is mostly experimentally driven.

Chapter 4 aims to provide understanding of the TCP/IP performance in data acquisition networks and analysis of the general approaches to solve the incast pathology. We consider potential analogies between DAQ and datacenter networks and discuss the applicability of the ex-

isting solutions to incast by evaluating their performance on real hardware.

One avenue to solve the incast pathology receives special consideration in Chapter 5. Here, we try to answer our main research question. We discuss whether incast congestion in DAQ networks can be avoided by using software switches with large packet buffers in the main memory of a server-class computer. This discussion is supported by evaluation of a prototype software switch using various designs. Our main objective in this chapter is to prove performance advantage of a single software switch under strong incast congestion.

In Chapter 6 we continue this study and discuss how large-scale networks based on software switches could be built and managed. We explain the advantages in taking the software-defined approach with centralised control plane for building a larger topology of interconnected software switches. Our main objective though is to show how the popular leaf-spine topology could be implemented and optimised specifically for data acquisition. We focus on such aspects as bandwidth scalability, packet routing, load balancing, resilience, costs, and space constraints. Here, the discussion is supported with a prototype leaf-spine data acquisition network consisting of eight software switches running on distinct physical servers.

Chapter 7 contains an additional evaluation of a new class of Ethernet devices — multi-host Ethernet controllers. These devices combine a traditional network interface controller used on servers with an advanced Ethernet switch. This can have some advantage when considering physical space usage of the proposed network design. However, since their performance characteristics are slightly different, we recheck in this chapter that performance is still adequate. Also, we discuss whether these new type of devices could overcome the known shortcomings of software switching in datacenter networks.

Concluding remarks and possible future lines of work are given in Chapter 8.

1.4 PUBLICATIONS

The following **conference papers** have been submitted/published during the course of my PhD:

Grzegorz Jereczek, Giovanna Lehmann Miotto and David Malone.
Analogues between tuning TCP for Data Acquisition and datacenter networks.
2015 IEEE International Conference on Communications (ICC), London, UK, June 2015.

Grzegorz Jereczek, Giovanna Lehmann Miotto, David Malone and Miroslaw Walukiewicz.
A lossless switch for data acquisition networks.

2015 IEEE 40th Conference on Local Computer Networks (LCN), Clearwater Beach, FL, USA, October 2015.

Grzegorz Jereczek, Giovanna Lehmann Miotto, David Malone and Miroslaw Walukiewicz.
A lossless network for data acquisition.
20th IEEE-NPSS Real Time Conference, Padova, Italy, June 2016.
IEEE Transactions on Nuclear Science, 2017.

Grzegorz Jereczek, Giovanna Lehmann Miotto, David Malone and Miroslaw Walukiewicz.
Approaching incast congestion with multi-host Ethernet controllers.
(submitted)
2017 IEEE Conference on Network Function Virtualization and Software Defined Networks, Berlin, Germany, November 2017.

Due to the industrial nature of the funding a number of **technical talks** were given during the course of this Ph D:

Data Acquisition Networks for Large Experiments.
Intel Labs, Gdansk, Poland, February 2014.

Analogues between tuning TCP for Data Acquisition and Datacenter Networks.
Intel Labs, Gdansk, Poland, October 2015.

Data Acquisition Networks for Large Experiments.
RIPE 69 Meeting, London, UK, November 2014.

Lossless Software Switching for Data Acquisition Networks.
Intel Labs, Gdansk, Poland, and Leixlip, Ireland, July 2015.

Open vSwitch Training.
Intel Labs, Gdansk, Poland, October 2015.

A Lossless Network for Data Acquisition.
Intel Labs, Gdansk, Poland, December 2015.

Software Switching for the LHC Experiments at CERN.
Keynote at the Software Professionals Conference.
Intel Labs, Gdansk, Poland, October 2016.

1.5 ADDITIONAL MATERIAL

The following weblink contains software repository used in this thesis: https://github.com/gjerecze/daq-software-switching.

2 LITERATURE REVIEW

In this chapter we present the current state of networking in large TDAQ systems, focusing on typical aspects and problems. Specifically, we gather important conclusions on the generic approaches to incast avoidance in many-to-one communication applications, not limited to data acquisition. We discuss how the work presented in this thesis advances the state of the art. Finally, we present current topics in software packet processing and SDN, which form the basic building blocks for our study.

2.1 DATA ACQUISITION NETWORKS

The challenge of data acquisition networks is the bursty many-to-one traffic pattern of DAQ data flows, which cause incast congestion as we highlighted in Section 1.1.2.1. Synchronous transfer of event data from multiple sources to a single compute node places a burden on the congestion control mechanism of a protocol and the network hardware itself. The default congestion control mechanism of TCP/IP has been found to be not capable of dealing with multiple relatively small flows in parallel [114]. Provided that the network does not have enough bandwidth to handle the burst rates or enough buffers to absorb these bursts, costly packet losses and retransmissions will occur. The DAQ networks of the four LHC experiments are particularly prone to incast because of their size and data rates. A number of solutions have already been implemented for the first and the second run of the LHC. Further research, including this work, is ongoing to provide alternative approaches for the future. Neufeld gave an introduction to the many-to-one challenge in DAQ in [118]. He also provides an overview of the TDAQ systems of the four LHC experiments in its first run and the outlook for the future upgrades.

2.1.1 *The ATLAS DAQ network*

The ATLAS DAQ/HLT system (see Figure 1.3) continuously evolves with subsequent runs of the LHC. The data rates in the ATLAS DAQ network (Figure 1.5) continue to increase, from approximately 80 Gbps in Run 1 to 400 Gbps in Run 2 [130], with plans to increase further in Runs 3 and 4. Today, as we already noted in Section 1.1.3, the ATLAS

DAQ/HLT system is based on 10 Gbps Ethernet (10GbE) network with TCP as the transport layer protocol. The core of the network is built around two large routers. Single event data of ∼1.7 MB is spread across around 100 readout nodes that are connected directly to the routers, while ∼2000 filtering nodes are organised in racks of at most 40 servers that connect to the core via ToR switches [39]. Ethernet was identified as the most suitable technology for the ATLAS DAQ network when it was first designed [146] and has been used ever since.

Stancu described in [152] the network congestion problem typical for the ATLAS DAQ and proposed an application-layer solution — traffic shaping. Its performance in the present system was further analysed in [39]. Traffic shaping is a simple credit-based system allowing the reduction of the impact of the DAQ traffic burstiness. It is employed on the aggregation side, Data Collection Managers (DCMs). Each of them is assigned a fixed number of credits. One credit permits a request for one event fragment. A DCM does not request more fragments than its currently available quota. The quota is reduced when requests to the ROS are sent and increased upon receiving the response with event data. This algorithm limits the burstiness of the data flow by spreading the DCM requests over time, thus taking down the instantaneous pressure at the switch queues.

Despite the successful implementation of the traffic shaping algorithm in the large DAQ network of the ATLAS experiment, it was shown in [39] that the buffering space available in a network plays a crucial role. Also, large buffers are still provided by telecom-class routers to avoid congestion in the network core. Optimising the system performance is, however, a trade-off between the cost of a better performing network, the cost of developing better traffic shaping techniques, and the cost of the inefficiencies introduced by network congestion. In this work we aim at removing the network congestion and reducing the network cost by using switches with expandable buffers in the cheaper DRAM memory of a commodity server.

2.1.2 *The LHCb DAQ network*

The LHCb's data acquisition network in Run 1 was based on Ethernet as described in [101, 119]. The filtering farm is of the same size as in the ATLAS experiment, whereas the number of event data sources (300) is significantly larger. Nevertheless, the single event size is only 35 kB with maximum rate of 1 MHz resulting in 280 Gbps of average bandwidth. Neufeld et al. indicated that only one very large core router was capable of sustaining the 300 to 1 overcommitment providing 256 MB of shared buffer space for a set of 48 ports [119]. The authors also mentioned the sources of packet loss that they had observed. Among them were the internal algorithms in the routers, which in practice cannot be substantially adjusted, like the buffer distribution or the distribution

of packets across link aggregation groups (LAG, IEEE 802.3ad [76]). A special workaround for the LAG hash-based algorithm was proposed to ensure that the same link of a LAG is used for packets belonging to a particular event, so that the many-to-one overcommitment is avoided in the subsequent network stage. In our research we propose a DAQ network based on software packet processing and SDN, in which packet routing can be easily optimised for a specific application as we will show in Chapter 5 and Chapter 6. Antichi et al. demonstrated in [8] the requirement for large packet buffers in DAQ networks by performing live traffic capture and time analysis in the LHCb DAQ.

The outlook for the LHCb Runs 2 and 3 can be found in [101, 102]. The event size will increase to 100 kB across 500 sources and the event rate will increase to 40 MHz. This will require a 32 Tbps bandwidth DAQ network, for which InfiniBand [11] and Ethernet technologies are considered. In [101] the author analysed a topology based on pizza-box switches to reduce the cost of the network based on high-end switches. It requires, however, up to 2.6 MB of buffering per port. He proposed a schedule of fixed time slots to send the event data to reduce the per port buffer requirement. This is mandatory because most of the ToR switches use so called *cut-through*[1] switching method with small buffers. The per port buffer requirement drops to 65 kB but only at the input load of 70 %. In our work we propose a solution that gives the means to provide the required buffering levels and operate at full available network load for a wide range of the system parameters, like the event size or the number of readout units.

2.1.3 *The CMS DAQ network*

Bawej et al. gave the summary of the evolution of the CMS DAQ from Run 1 to Run 2 in [19]. The average event size is comparable to that of the ATLAS experiment. It increased from 1 MB in Run 1 to 2 MB in Run 2 with the rate of 100 kHz at the output of the dedicated hardware trigger. The overview of the CMS DAQ system for Run 2 is depicted in Figure 2.1. The CMS DAQ network has evolved from a combination of Myrinet [112] and Ethernet to a combination of Ethernet and InfiniBand technologies to take the advantage of high data transmission speeds of the latter [18]. Instead of increasing the number of nodes in the network, the link speeds have been increased and the event building nodes (64) have been optimised to provide the required bandwidth under the increased load. The InfiniBand architecture fits better in this concept than Ethernet, because of the available data rates and the so called *zero-copy* mechanism, which reduces the overhead of the software protocol stacks in the operating system of the end hosts, as in case

1 *Cut-through* switches can begin transmitting a frame before it has even been fully received at the input. In contrast, *store-and-forward* switches receive (store) a frame completely before any decisions regarding forwarding are made[150].

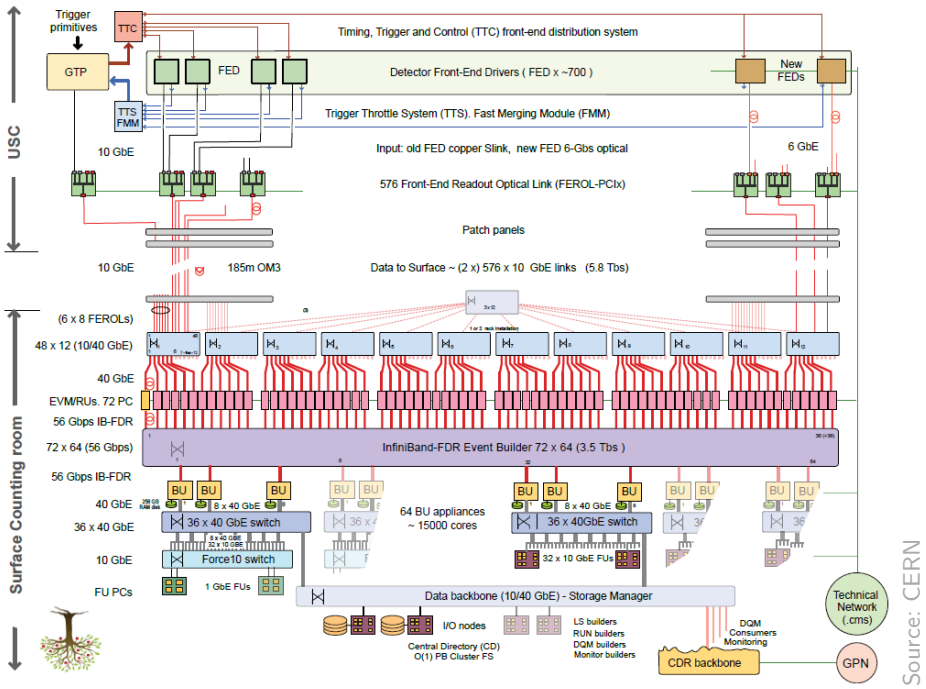


Figure 2.1: Architecture of the CMS DAQ system for Run 2 of the LHC [19].

of the TCP/IP stack. InfiniBand connects event building nodes with the readout system (84 nodes). The many-to-one overcommitment is handled by the built-in flow control mechanism of InfiniBand. Event filtering is performed by separate nodes that communicate with the event builders over an Ethernet network. We will return to the discussion on InfiniBand and Ethernet in Section 2.2.1.

2.1.4 *The ALICE DAQ network*

The characteristics of the DAQ system belonging to the ALICE experiment is different from the other experiments. The rate of the events is low and their size is large. Two separate networks were used for event building and event filtering (HLT) like at CMS. A 160 Gbps Ethernet network was used during Run 1 with 175 *local collectors* pushing data to 83 *global collectors* as outlined in [31], whereas an InfiniBand network served the HLT farm of 225 nodes [95]. According to [31], only software adaptation and replacement of the obsolete computing hardware was conducted in preparation for Run 2. The requirements will significantly increase first for Run 3 [30]. There will be a single shared computer farm for both event building and filtering, which will require a network interconnecting 2000 nodes and providing 5 Tbps aggregate bandwidth during the peak data taking periods. Ethernet and InfiniBand in various topologies are considered. Rybalchenko et al. gave further details on the challenges facing the ALICE DAQ network [145]. 11 GB of event

data distributed across approximately 250 nodes will need to be delivered to one of 1500 processing nodes. The authors identified that incast congestion will become an issue in this case and analysed an application layer solution: staggering of the data transfers, originally proposed by [175]. The staggering approach tries to mitigate incast by delaying the transmission of data on each source by a unique offset, still maintaining high network throughput. The solution requires manual adjustment of the delay value for the given scenario and is normally recommended to be close to the transmission time of one data unit. Precise timing is required to reach full performance, for smaller event fragment sizes in particular. The presented results confirm the effectiveness of the method in the context of the requirements of the ALICE experiments with large event fragments.

2.1.5 *Summary*

Data acquisition networks of the four LHC experiments share a common challenge. This challenge is providing performance and reliability at the same time under heavy many-to-one congestion in terabit networks. Each of the experiments incorporates its own mechanisms for congestion control on top of the Ethernet and/or InfiniBand technologies to accommodate for the insufficient packet buffers in the network devices. In the following section we will extend the review of the available solutions to the same problem, but originating from different types of networks.

2.2 SOLUTIONS FOR MANY-TO-ONE COMMUNICATION

Many-to-one communication has been widely studied in the literature. As it turns out, this traffic pattern is found to be challenging not only in data acquisition networks, but also in datacenters as we highlighted in [87], which increases the number of potential solutions to the problem. In the following we will extend the experiment-specific solutions presented in the previous section with a general review of the work on the many-to-one congestion and promising networking technologies in the context of data acquisition. A number of proposals for incast mitigation ranging from the link through the transport up to the application layer can be found in the literature. A broad review of available solutions was presented in [139, 142, 144, 177]. We will review the most relevant of these in Section 2.2.2.

Although our research concentrates on commodity Ethernet as we explained in Chapter 1, we will start with a short discussion on Ethernet and InfiniBand technologies, which are the dominant interconnect technologies in the high performance computing (HPC) world and also the two technologies used to built the DAQ networks at the LHC nowadays. According to the November 2015 Top 500 Supercomputers list

[168] 47.4 % of the supercomputers deploy InfiniBand and 36.2 % Ethernet. Myrinet is no longer used by any of the LHC experiments and also its share in the Top 500 Supercomputers list dropped from 20.2 % in June 2005 to only 0.2 % in June 2014.

2.2.1 *Ethernet versus InfiniBand*

Ethernet [76] and InfiniBand [11] are the dominant interconnect technologies as explained in the previous section. One of InfiniBand's primary focuses is low latency, which is an important factor in HPC applications [100], but in data acquisition the high, long-term throughput under heavy congestion is more relevant. Nevertheless, InfiniBand has built-in capabilities for hardware-level flow control and congestion control [68, 137]. Therefore, InfiniBand switches are relatively cheap, because they can operate with limited buffer memories and provide means to operate in many-to-one communication scenarios with very low latency. It requires, however, overprovisioning as explained in the previous section. The analysis in [148] showed that a DAQ network based on InfiniBand technology can be significantly cheaper because of the high cost of the telecom level routers with enough buffering space in the case of Ethernet. This is the obstacle which we address in this work. Liu showed in [101] an extensive comparison of InfiniBand and Ethernet in the context of DAQ networks and gave a slight preference to the latter in the conclusion. Ethernet was originally evaluated for application in the ATLAS experiment by [146] and the reasons for this choice are still valid today.

Despite InfiniBand's strong position in the HPC and deployments in DAQ (CMS [18] and ALICE [95] experiments), it is not a technology that is as widely spread in various domains as Ethernet. The latter is the most ubiquitous kind of computer network to date and not limited to HPC [100, 141, 157]. It will now also have to compete with the InfiniBand-related Intel Omni-Path Architecture, which has been announced recently [24] as a next generation fabric for the HPC. Furthermore, the InfiniBand programming model [11] is complicated and differs significantly from the well-established Berkely socket API [154].

Ethernet is, however, a best-effort interconnect technology with no guarantee of delivery. Packets can be dropped in response to a variety of conditions, including congestion. Reliability is normally provided by high-level protocols like TCP, but it becomes more and more challenging with the ever increasing data rates of the LHC experiments. In the recent years, a number of enhancements to the Ethernet standard have been proposed to provide so called *lossless Ethernet*. Ethernet flow control (IEEE 802.3x [76]), Priority-based Flow Control (PFC, IEEE 802.1Qbb [75]), Enhanced Transmission Selection (ETS, IEEE 802.1Qaz [75]), and Congestion Notification (IEEE 802.1Qau [75]) are enhancements to the Ethernet standard, which define mechanisms for

congestion control at the link level. The last three are defined under the Datacenter Bridging (DCB) [44] umbrella that adapts Ethernet to high-performance networking and closes the gap with InfiniBand [141]. An overview of these technologies is available in [100, 180]. We will discuss them briefly and also other means, including upper-layer solutions, to improve the performance of Ethernet-based networks in the context of data acquisition in the following sections.

2.2.2 *Ethernet and TCP/IP technologies*

2.2.2.1 *Enhancements to the Ethernet standard*

The basic Ethernet flow control mechanism is the IEEE 802.3x pause frame. If the packet buffer of a network node starts to fill up, it can send a pause frame to the neighbouring node to temporarily stop all traffic. Although this mechanism can guarantee lossless operation, it cannot achieve full throughput in incast scenarios [139] and also introduces Head-of-Line (HoL) blocking, which further reduces the bandwidth utilisation [59, 69, 100, 180]. In our evaluation in Section 4.5.3 we analysed a simple setup with a single switch. Although the performance is significantly improved, theoretical bandwidth cannot be reached. Stancu et al. evaluated this technology in [153] for the application in the ATLAS experiment in Run 1 of the LHC and concluded that HoL blocking can be avoided with switches that employ separate input buffers for each outgoing port, but the system can be affected by the congestion spreading effect. The pause frame mechanism evolved to the IEEE 802.1Qbb Priority-based Flow Control, in which not all traffic, but frames belonging to a particular traffic class, can be paused, so that the non-congested traffic remains unaffected. The HoL blocking is now limited to the particular traffic class, but only up to eight traffic classes are supported. Since there is only one type of traffic in DAQ networks requiring careful flow control, the one transporting event data, PFC brings no advantages over the regular pause frame mechanism. Flow-based, instead of class-based, congestion control is required, if more flows of the same class suffer from the limitations of pause frames [180].

Quantized Congestion Notification, defined in IEEE 802.1Qau, can be used to further optimise the traffic flow. It provides flow-based congestion control. A frame with feedback based on the congestion level is sent from a QCN congestion point (e.g. a switch) back to a QCN reaction point (e.g. a server), which limits the sending rate. As explained in [100], QCN works well only for long lived data flows and requires a lot of fine tuning based on the network topology. Furthermore, since frames are sampled randomly at the congestion point, it may happen that the source that is throttled may not be the main source of congestion [100]. These factors can make QCN application in DAQ even more complicated, because the traffic is composed of hundreds of relatively

small data flows, which do not cause congestion on their own, but first when summed up, as we will show in Chapter 3. It was confirmed in [47, 178] that QCN does not perform well under incast congestion. Although the proposed modifications improve its performance, full link utilisation is still not reached. Tanisawa and Yamamoto came to similar conclusions in [158] — optimisations are required when large number of flows traverse a bottleneck link.

2.2.2.2 *Solutions at higher layers*

TRANSPORT LAYER Transmission Control Protocol (TCP) [55] is the most widely spread transport layer protocol, and as such has been thoroughly analysed in terms of congestion control, which is built into the protocol. Many alternative TCP flavours have been proposed and implemented to diminish or eliminate the issues arising from congested links. A broad set of alternatives deal specifically with many-to-one communication patterns. First indications of poor TCP performance with many flows were already presented by Morris in [111], who stated that "one way to work around the problem is to make sure routers have not just one round-trip time [RTT] of buffering, but buffering proportional to the total number of active flows". Incast was first documented by Nagle, Serenyi, and Matthews in their paper on cluster storage [114]. More recent analytic models of TCP incast behaviour can be found in [36, 98].

Datacenter TCP (DCTCP) is the well-known TCP variant for datacenters proposed by Alizadeh et al. in [6]. DCTCP is now an active Internet draft at the IETF [20]. It leverages the Explicit Congestion Notification (ECN) [55] mechanism at the IP layer to keep the switch queues small while maintaining high throughput. An ECN-capable switch under congestion marks packets with a special ECN-flag: *Congestion Experienced*. The TCP receiver upon observing this flag sends the feedback about congestion back to the TCP sender, which adjusts its sending rate according to an algorithm defined by DCTCP. The authors show that DCTCP achieves similar performance to the one of deep buffered switches. It is argued, however, that the short-message traffic is penalised due to the queue buildup phenomenon. This drawback is avoided with our design thanks to the dedicated queues for incast-sensitive flows. Furthermore, DCTCP fails to avoid incast, if there are so many senders that the packets sent in the first RTT overflow the buffers, as already indicated by the authors in [6]. This situation is not uncommon for high-bandwidth low-latency DAQ networks under severe all-to-all incast congestion.

Bai et al. in [13] further explained the limitation of the proposals that control the TCP windows to reach the optimal aggregate throughput (*window-based* solutions), like DCTCP [6] or Incast Congestion Control for TCP (ICTCP) [174]. In very large systems with synchronised flows even a minimal TCP window that is of the size of a single packet is

sufficient to overwhelm the switch buffer. This situation is not uncommon for high-bandwidth low-latency DAQ networks like the one of the LHC DAQ systems as we will show in Section 4.4. The authors propose therefore a "delay-based" mechanism of proactive regulation of TCP acknowledgements (TCP ACKs), PAC, to regulate the traffic so that links are fully utilised without incast congestion. The difficulty of this approach lies in controlling the TCP ACK rate such that no bandwidth is wasted and incast is avoided. The results presented indeed demonstrate better scalability of the PAC algorithm in the number of concurrent flows than that of DCTCP and ICTCP. In their tests the authors used only a single receiver though. It can be suspected that under more stringent conditions (all-to-all incast) the ACK regulation can become even more challenging.

Conversely, Hwang, Yoo, and Choi proposed a window-based algorithm, Deadline and Incast Aware TCP (DIATCP) [73], which overcomes the limitation of single packet TCP window. Some senders can be temporarily paused by sending an advertised TCP receive window (*awnd*) equal to zero in the TCP ACK from the receiver. We will evaluate DIATCP in Section 4.5.2.2.

A third group of solutions for TCP incast mitigation, after window- and delay-based solutions, is *recovery-based* [13], which reduces the impact of incast after it has occurred. The most obvious way to reduce the throughput degradation is to minimise the time between a packet drop and its retransmission. In case of TCP incast this time is determined by the TCP Retransmission Timeout (TCP RTO) parameter that is of an order of hundreds of milliseconds compared to microsecond RTTs in typical datacenter or data acquisition networks (see Section 4.2). The fine-grained TCP RTO was proposed by Vasudevan et al. in [171]. Although the goodput is significantly improved, the recovery-based solutions cannot be regarded as valid solutions for DAQ networks in general. Incast congestion is permanent, not occasional, so systematic retransmissions would lead to overall underutilisation of the available bandwidth.

User Datagram Protocol (UDP) [55] is a simple alternative to TCP. It does not provide reliability, sequencing, flow control, or congestion control, so these have to be implemented in the application layer, if required. On the other hand, if a DAQ network is lossless and delivers the packet in-order, a simple push-architecture for data acquisition based on UDP can be used. We will give some examples of application layer solutions to incast congestion in the paragraph devoted to the application layer.

Perry et al. proposed in [132] an alternative datacenter network architecture, which minimises switch queues and packet retransmissions in datacenter networks — *Fastpass*. It introduces a standalone arbiter that schedules precisely each packet in the network and determines its path. Fastpass is implemented as transport layer protocol in Linux, beneath TCP or UDP and can be used transparently by applications.

No modifications to the networking hardware are required. The authors demonstrated that a single arbiter has the ability to schedule 2.21 Tbps of traffic in software on eight CPU cores, which make this architecture a promising candidate for data acquisition, although no results in an incast scenario have been presented yet.

APPLICATION LAYER The literature is also rich in proposals that deal with the many-to-one communication pattern at the application layer. The advantage is that the *application* has better view of the parallel communication, like the total number of flows. This knowledge can be used to reduce the impact of incast. The authors in [97] provided a review of some of the proposals. Among them are client- and server-side staggering, which we mentioned already in the work related to ATLAS and ALICE experiments in Section 2.1, or global scheduling of data transfers.

An analogous approach to the ATLAS traffic shaping algorithm is used also in Facebook's [54] datacenters in their memory caching solution, which is also prone to incast congestion [121]. A sliding window mechanism is used at the client's side to limit the number of outstanding requests to various servers. The size of this sliding window corresponds to the number of traffic shaping credits in the ATLAS DAQ network.

The advantage of pursuing an application-level solution to incast is clear — the number of data packets traversing a network in parallel can be tuned more easily than at the lower network levels. In our work we extend this approach even more. One of the key characteristics of data acquisition networks is their configuration and traffic characteristics remain constant over long period of times, as we will describe in more detail in Chapter 3. Furthermore, this is known a priori. It does not usually change after an experiment has started and continues its operation in that state for months or even years. We will show how to take advantage of this global knowledge about the network in Chapter 6.

2.2.3 *Summary*

The majority of the approaches to incast congestion focus on the control of the packet injection rate into the network so as not to overwhelm its buffers. All of them have means to improve, to some extent at least, the performance of DAQ and other incast-sensitive networks. It was claimed in [118], however, that these mechanisms, flow control in particular, do not work well in DAQ with constant instantaneous overload, unless the network is significantly overprovisioned, because they are designed to absorb fluctuations only. Our approach differs in that there is no rate control required on the sender side and enough buffering is provided in a cost-effective and flexible way, which simplifies the entire system. As it is pointed out in [39, 118, 148], very expensive buffers in

the read-out network are the main obstacle in the use of a simple push architecture[2] in data acquisition. Also, in datacenters, large buffers have been proposed as a solution for the incast congestion problem, but again the high cost of such network devices prevent it from being considered as a valid option [142]. Furthermore, every switch configuration will have some maximum capacity, so the scalability is seriously limited as noted by [139].

In this work we argue that this alternative approach for many-to-one communication networks, large network buffers, can be still considered as a cost-effective, flexible, and scalable solution. We also evaluate this approach in the demanding all-to-all incast scenario as opposed to all-to-one scenario used in most of the available literature. Our approach differs also in the use of the global view of a DAQ network to optimise flow distribution across available paths, perform rate adaptation, and allocate buffers. The main building blocks of our proposal are software packet processing and Software-Defined Networking. In the following sections we will review the key aspects of these technologies.

2.3 SOFTWARE PACKET PROCESSING

Software switches and routers[3] are not a new concept as already shown in Section 1.2. The first network devices built with general-purpose computers and NICs were, however, consequently replaced by dedicated hardware for performance reasons. In the 2000s attempts to use conventional PCs as software routers were made again in order to facilitate flexibility in opposition to closed and inflexible designs of the commercial devices [23, 25, 26, 35, 94]. Both the forwarding code in the Linux kernel and specialised projects, like the Click software router [94], were used. Click has remained a popular framework for network experiments until today.

Data plane performance was, however, the key limitation of software-based packet forwarding. It was rather difficult to reach high packet processing rates[4]. In 2007 a rate of 700 kpps (*pps* — packets per second) was reported by [25] when forwarding Ethernet frames between two 1GbE ports. It translated into the throughput value of 0.47 Gbps, when forwarding minimum-sized frames (64 B). Saturation could be reached with frames of size 160 B and longer. The maximum achieved

2 Push architecture means that sources send the data whenever they are ready. In contrast, in the pull architecture the destinations request data from the readout units [118].

3 In the literature both terms are used to describe applications that can actually provide the same functionality — packet forwarding and manipulation. Traditionally, switches forward packets based on layer 2 addresses (MAC), whereas routers use layer 3 (IP).

4 Packet processing rate, given in *packets per second* (pps), is one of the main performance indicators for devices that forward packets. For software switches, the available CPU cycles set the upper limit on the number of packets that can be processed in a unit of time, independent of their size.

throughput in a scenario with multiple NICs was 2.5 Gbps and could not reach too far beyond the half of the theoretical value. The main limiting factors were the computational capacity and the bandwidth/latency of I/O buses [26].

But the performance kept improving over the years with technological advances in commodity computers. I/O improved with the introduction of the PCIe buses. The available computational cycles increased with the architectures supporting multiple multi-core CPUs, across which packet processing could be distributed with multiqueue NICs[5] [26]. In 2008 a rate of 5.9 Mpps was reached with eight CPU cores by Bolla and Bruschi [26], 7.1 Mpps with five cores by Egi et al. [50], and 8.2 Mpps with eight cores by Argyraki et al. [10]. Additionally, it was stated in [50] that a single core in their evaluation setup could process 2.5 Mpps.

The performance did not scale ideally when increasing the number of cores, reaching saturation at some point. It was shown in [10, 50] that the fundamental limit on the performance of a single software router was the memory latency (not bandwidth) and the NUMA (Non-Uniform Memory Access) architectures were pointed to as a potential improvement. With the new Intel Nehalem microarchitecture [120] the performance indeed improved, reaching 3.87 Mpps with one and 23.2 Mpps with eight CPU cores (2.8 GHz) and two 10GbE ports [103]. This family of processors introduced for the first time the Intel QuickPath Architecture that is still used by the latest Intel processors. It replaced the front-side bus (FSB), which was used before for communication between CPUs and a chipset that contained the memory controller and connections to other buses. In this new architecture each CPU has an integrated memory controller (IMC). The CPUs are connected with each other and with the I/O hub[6] by a new high-speed, point-to-point link — QPI (QuickPath Interconnect). Han et al. optimised the packet processing software and reached 40 Gbps throughput for small packets forwarding on multiple cores (59.5 Mpps). Memory bandwidth and available CPU cycles were named as the limiting factors for reaching 100 Gbps.

Dobrescu et al. took into account the full IP routing functionality when performing their performance evaluation in [49]. It turned out that the performance degrades from 19 Mpps to 12 Mpps on their evaluation setup. Extra CPU cycles were required to perform tasks like packet header manipulations or IP table lookups. They proposed the

5 Multiqueue NICs have multiple hardware transmit and receive queues, also called *tx-* and *rx-rings* [26, 49]. These rings are used by the DMA (Direct Memory Access) engine to transmit and receive packets to and from the NIC. The packets arriving at a NIC are classified internally into the queues and each core can then process a subset of packets from its assigned queues.

6 The PCIe controller was integrated into CPU first with the Intel SandyBridge microarchitecture [78].

RouteBricks architecture — the idea of scaling the software router by parallelising its functionality across multiple PC's.

Early 2010s gave raise to software packet processing frameworks, which were to be used instead of the standard network stack of the Linux kernel in order to improve performance. These frameworks can be used to build custom applications to process packets. The performance of the software switch/router projects was also improved thanks to them. The most popular frameworks include: netmap [143], DPDK [80], PF_RING [138], and Snabb Switch [151]. The technical aspects behind them, their comparison as well as key advances in the commodity computers were discussed in [16, 51, 61, 110]. The latter provides also a model describing packet processing software in general. Although the architectures of those projects differ, they all provide comparable, very high performance in packet processing on x86 servers.

DPDK was even seen by Zhou et al. in [179] as a successor of RouteBricks and showed that saturating 80 Gbps for packets of 192 bytes or larger is practical. Rizzo demonstrated that simple packet forwarding in the netmap framework saturates a 10GbE link with 14.88 Mpps for 64 B packets on a single CPU core running at 1.7333 GHz only [143]. It means that a single core could forward 183 Gbps of maximum sized Ethernet frames. This is the type of frames that is mostly carried by the DAQ network of the ATLAS experiment, as we will show in Chapter 3. The bandwidth of 183 Gbps lies in the same order of magnitude as the requirements of the experiment in Run 1. These facts show that the performance of software switching is already enough for it to be considered as viable option for data acquisition networks.

In 2009 as a response to the challenges in networking in the virtualised environments, which gained increasing popularity, the Open vSwitch (OVS) project was presented [135]. This open source *virtual switch*[7] was purpose-built for these environments and was meant to replace the Linux Ethernet bridge that was used by many hypervisors and has been present in the Linux kernel since 2002. Nevertheless, OVS is a software switch that can be used also as a switch (router) interconnecting physical ports on a general-purpose server. It supports typical network management interfaces and protocols and can be considered as a fully-fledged switch. Performance in this scenario was evaluated in [52]. Although 1.88 Mpps is far from the values reported in the previous paragraph, the DPDK-accelerated version of OVS reached good performance of 11.31 Mpps. The official support for DPDK was added in the OVS release 2.2 [125]. Open vSwitch provided now both: performance and support for protocols that are typically required from network devices. OVS has already become the default virtual switch in all important cloud frameworks [51]. Its design and implementation details are available in [136]. In this thesis we will evaluate OVS in the

7 Virtual switches provide network access for virtual machines (VMs) by linking virtual and physical network interfaces (hence called *virtual*).

context of data acquisition and extend it with optimisations tailored for this type of systems.

To summarise, performance has traditionally been the challenge of packet processing in software. This situation changes, however, with modern server platforms. High performance load balancers, proxies, virtual switches and other network functions can be now implemented in software and not limited to specialised commercial hardware, thus reducing cost and increasing the flexibility. Software switches, like OVS, have already become an important part of cloud networking architectures. In our work we build on the lessons learnt over the last years and try to exploit the flexibility of the design in software to provide a dedicated network for data acquisition. Our study will differ from the previous ones in that we will take into account the key factors from the viewpoint of data acquisition, like high throughput for large data transfers and reliability under many-to-one congestion.

2.4 NETWORK TOPOLOGIES

Scalability is an important aspect of data acquisition networks as they need to provide the required port density and aggregate bandwidth for a particular system. For the experiments like the detectors at LHC it means designing and building networks that interconnect thousands of readout and filtering nodes.

Traditionally, DAQ networks have used large core-router style devices [39, 48, 148], which could offer not only the adequate number of ports and bandwidth, but also large buffers to diminish packet drops resulting from incast congestion. An example is the ATLAS DAQ network, presented in Figure 1.5. We discussed the drawbacks to this approach in Section 1.2. Similarly in datacenters, traditional network architectures consist of two- or three-level trees of switches or routers with large devices of higher bandwidth in the core level to increase the overall bandwidth of the network, but still remain oversubscribed to reduce the costs as described by Al-Fares, Loukissas, and Vahdat in [56]. Just like Charles Clos for telephone switches [38], the authors noticed, however, that large-scale networks can be built from many small commodity switches organised in a topology rather than fewer larger and more expensive ones.

A variety of topologies, like torus, mesh, butterfly or Clos networks, and routing algorithms can be found in the literature on interconnection networks (sometimes called *fabrics*) [42]. Traditionally, the research focused on fabrics for on-chip networks to deliver data between components in a CPU, building supercomputers, or connecting input to output ports in network routers internally. These topologies have already become popular in building datacenter fabrics [177]. It was pointed out in [5] that the datacenter networks evolve to a simple Clos-based ar-

chitecture [38], often called leaf-spine or fat-tree topology[8], that aim to approximate a large non-blocking switch. This approach can be also interesting for data acquisition. Carena et al. evaluated the bandwidth scalability of the Clos network [38] for the future runs of the ALICE experiment [30]. It was shown that this architecture can easily scale to multi-terabit networks, but the aspect of congestion control was not discussed. The CMS DAQ in Run 2 is built around an InfiniBand Clos network of 18 individual 36-port switches and relies on the InfiniBand's internal flow control mechanism for congestion avoidance [19].

The obstacle for adoption of topologies based on small commodity switches in data acquisition is their performance under incast congestion, which we will explain in more detail in Chapter 3. Alizadeh and Edsall indicated in [5] that larger buffers both in leaf and spine switches of the leaf-spine topology are needed for incast patterns. A similar conclusion was made in the evaluation of a topology based on commodity switches for the LHCb experiment [101], highlighted already in Section 2.1. We address this issue with the use of software switches in this work. Software switches with extended buffering capabilities open up the possibility of using these topologies and optimising their queueing algorithms for data acquisition. This is possible without the need for the expensive core routers while holding on to the commodity TCP/IP and Ethernet networks. In Chapter 6 we will propose and analyse the performance and management of a network, which design is based on optimised software switches and a modified leaf-spine topology for multi-terabit DAQ networks.

An important aspect to consider is the forwarding algorithm used to move packets across these topologies. There are multiple paths between all pairs of hosts, so an efficient algorithm to balance the load across these paths is needed to optimise the performance. This task is difficult for datacenters, supercomputers or lower-level interconnection networks, where the traffic patterns change dynamically. Many proposals from the literature address proper load balancing in these dynamic scenarios [42, 56, 144, 177]. In datacenter environments the current state of the art is equal-cost multi-path (ECMP, IEEE 802.1Qbp [75]), which statically assigns flows to paths using hashing. Static load-balancing cannot guarantee the best performance for networks without uniform or predefined workloads [57, 144]. It should be possible though to optimise the performance in networks with workloads that are known a priori. This is the case in many DAQ networks. It should be feasible to assign flows to paths in such a way that maximises throughput. In Section 6.2.2 we will propose a method to perform this assignment in order to maximise the offered bandwidth for DAQ-specific data flows. Instead of using ECMP hashing, we will *program* the network explicitly

8 Originally, in the fat-tree topology the link capacities increase towards the top of the tree. In the present designs multiple switches are used to emulate this architecture. In the end fat-trees resemble the Clos topology.

with SDN technologies, which allows for straightforward implementation of both: optimum load balancing strategy and incast avoidance for the topology based on software switches.

2.5 SOFTWARE-DEFINED NETWORKING

SDN has gained popularity both in research and industry. Open vSwitch as well as many commercial hardware switches support now the OpenFlow protocol [108], which is the most prominent API between the control and data plane in SDN networks. These planes are meant to be decoupled as we described in Section 1.1.4. Switches, being datapath element, are *programmed* by an external controller (often called *network operating system*) using OpenFlow (see Figure 1.6). Programming means in this context that a set of rules is installed on a switch. Each rule consists of a pattern to match specific packets and actions to be performed on those packets (e. g. dropping, packer header manipulation, forwarding to a port, etc.). Example applications, like load balancers, dynamic access control, routers, and many others, can be easily created in software using one of the different network controller platforms. The literature offers a variety of network designs based on SDN. An overview of these projects as well as a broad introduction to the topic of programmable networks and SDN is available in [58, 66, 96, 113, 123].

Network programmability can also offer advantages in DAQ networks. It was also noticed by Cui, Yu, and Yan, who pointed out in [41] that the SDN features can greatly facilitate big data applications. Programmability and flexibility can become the means to improve network performance for complex traffic patterns. In our work we concentrate on data acquisition and evaluate dedicated algorithms to forward packets across the network in such a way that minimises congestion and optimises data collection latency in DAQ systems. It resembles to some extent the proposal for Quality-of-Service (QoS) control in a converged network presented in [93]. The proposed network controller assigns flows to rate limiters and priority queues to enforce aggregate bandwidth usage. However, in order to enforce the rate limits, packets are dropped. Mohan, Divakaran, and Gurusamy showed in [109] that providing QoS guarantees with OpenFlow QoS API leads actually to batches of packets being dropped periodically, which deteriorates the performance. Dropping packets is not an option for DAQ as we already explained in Section 1.1.2.1. Instead, we provide large enough queues in the software switch, so that the packets are not lost and can still be rate-limited. The network controller assigns the flows to these queues in such a way that the overall performance of the DAQ system is optimised. An overview of network resource management using OpenFlow was given in [181]. The authors indicated that rate control for individual flows is challenging because maintaining a queue for each individual

flow in hardware is needed. In our approach we overcome this with software switches, in which large numbers of software queues can be used.

Network programmability can also include the end-hosts, not only switches. With OVS being already part of Linux it is straight-forward to include them in the common network control plane, particularly if multiple network ports are available. A single protocol, OpenFlow for example, can be then used to implement optimal load balancing for the entire network. This idea was first evaluated in the context of datacenter and wide-area networks in [116] with a positive result. In Chapter 6 we will show how it could be implemented for DAQ.

The same authors analysed also the question of ARP (Address Resolution Protocol) handling. This protocol is used for address translation between IP and hardware MAC (Media Access Control) addresses used by various networking technologies [55]. Traditionally, so-called ARP requests packets are broadcast on the local network to retrieve the MAC address for the specified destination IP address. In order to avoid loops, the Spanning Tree Protocol (STP, IEEE 802.1d [75]) is normally used to create loop-free topology, but redundant ports are blocked, so available network bandwidth is significantly reduced. Alternatively the network can be segmented into virtual networks (VLANs) and the ECMP protocol [71] can be used to distribute traffic across multiple paths or links can be grouped into LAGs [76]. A recent protocol enabling multipath without the need to segment the network is TRILL (Transparent Interconnection of Lots of Links) [169]. An overview of these technologies is available in [46]. In case of SDN networks, however, ARP handling must be implemented differently. The network controller needs to define and install appropriate rules on the switches for ARP packets. In [37, 116] the authors used the so-called proxy-ARP mechanism, in which ARP packets are intercepted by the switches and redirected to the network controller, instead of being broadcast. The controller in turn instructs the switches to send an ARP reply message with the appropriate MAC address. In Chapter 6 we will present a way to remove ARP packets altogether for DAQ-like, isolated networks. In an SDN network the entire packet forwarding can be based solely on IP addresses without the need to use hardware MAC addresses, required in traditional networks. In this way network configuration is simplified and ARP traffic can be eliminated, which in large networks can reduce the bandwidth utilisation for normal traffic [37].

One of the main points against SDN is the flow installation latency, which is the time it takes a controller to add or modify a rule on a switch. SDN can operate in two control models [123]:

- reactive, and
- proactive.

In the former, for every first packet of a new flow that arrives at a switch this packet is sent to the controller to make a decision about

an action for this flow. A corresponding rule is then installed on the switch and subsequent packets can be forwarded by the switch at once. This model incurs additional delay that for large number of short-lived flows can significantly reduce the performance. Scalability can be also of concern, if the controller must handle a large volume of new flows. Alternatively, in the proactive mode the controller makes an attempt to install rules on the switches without waiting for the first packet of new flows. In this case, the switches rarely need to consult the controller about new flows. As we already mentioned, all flows in a DAQ network are known in advance, so the proactive mode can be used and the flow installation latency is not an issue.

In this configuration, the flow installation latency can have more influence on the efficiency of the failover mechanism. In case of a link or interface failure, the controller has to detect it, recompute alternative paths and install them on the switches. A network operates with degraded performance for the time it takes to perform these tasks. It is therefore important to optimise the process. Different approaches to provide resiliency exist. An overview of the techniques can be found in [12]. We will discuss failover in more detail in the context of DAQ networks in Section 6.2.4.

In SDN reliability of the control plane is critically important. A single controller introduces single point of failure that can bring down the entire network. The main mechanism to recover from these failures is to use backup controllers that can take over the network control when the primary controller fails. This problem is also well-known and solutions exist. More details are available in [4, 12]. Multiple controllers can also be used to improve the scalability of flow management for larger networks. Review of the work in this area is available in [12].

Another widely criticised aspect of SDN is the flow table capacity [96]. The forwarding rules are stored in the so-called flow tables, defined in the OpenFlow specification [126], inside network devices, which have limited capacity. This is even more challenging, if multiple OpenFlow tables are to be used. Considering, however, the type and number of rules that would be required in a DAQ network (see Chapter 6), this is not a real issue. Especially that the OpenFlow tables' capacities can be easily extended with software switches [179] that we intend to evaluate jointly with SDN.

Summarising, programmable networks, like SDN, offer promising features for data acquisition networks. Among others, the software mechanism for implementing custom algorithms for routing and load balancing across a network can bring compelling advantages. Optimising the data flow in a DAQ network can become much simpler and more efficient than in case of traditional networks, in which flexibility is limited by the network hardware vendors. Possible limitations of SDN either do not appear in DAQ or working solutions are already provided.

2.6 SUMMARY

In this chapter we reviewed the relevant literature on the topics to be discussed in the thesis.

We started by looking at the data acquisitions networks of the four large experiments of the LHC. In particular we reviewed what solutions were used in the past and where are the challenges for handling the bursty many-to-one traffic pattern in the future. There are some obstacles in the adoption of the commodity Ethernet and TCP/IP networks. The main one is the lack of buffering in the network. Although solutions improving the performance of these networks for many-to-one communication exist both in data acquisition and in datacenters, all of them require careful tuning and sender-side buffering. Furthermore, it is difficult to reach full performance.

Packet processing in software with flexible and extensible buffering can be an alternative avenue. We presented the evolution of software switches and routers. Nowadays, the performance is sufficient to build a switch on a commodity server that is capable of forwarding hundreds of gigabits of traffic. A high performance, production-quality software switch, OVS, is already available and has become an important part of modern datacenters. Nevertheless, assessment and possibly optimisation for data acquisition are required.

When combined with the Clos-based network architectures, a topology based on software switches that could scale to terabits without incast congestion seems feasible. Particularly if proper load balancing across multipath is employed, very high network utilisation can be reached. The work on software-defined networks is promising for DAQ, particularly in respect of resource management. QoS guarantees are often based on packet drops to enforce proper rate control though. However, drops would not be required if an adequate number of large-enough queues is guaranteed on the switches. We will evaluate this in the following chapters by exploring software switching in the context of data acquisition.

3

PERFORMANCE IN DATA ACQUISITION NETWORKS

In this chapter we discuss the most relevant properties of typical data acquisition networks for large experiments. First, we give the necessary definitions to quantify the performance. Then we define basic requirements, which can be regarded as common for many DAQ networks, and consider what are the important aspects when optimising them. We also present in more detail the ATLAS DAQ system, which is used as the use case throughout this thesis. In the end, we describe our performance evaluation methodology.

3.1 INTRODUCTION

As we have already described in Section 1.1.2.1, the data acquisition network is one of the key components in distributed large-scale experiments. It is required for collecting the experiment's data from all of its instruments. This process is illustrated in a lightweight fashion in Figure 1.2, and with more details in Figure 3.1. Data that are read by some readout electronics from an experiment are usually buffered on multiple readout units. They constitute the readout system. Data that belongs to the same physical process (e. g. the same collision event in LHC), must be then merged together and saved to disk. If there is not enough bandwidth to this storage, filtering takes place. In order to accomplish these tasks a network is required, which interconnects all the nodes that are used for readout, merging (called event building in the LHC experiments), filtering, and permanent storage. Sometimes some of them can be even performed on the same nodes [102]. In other configurations distinct networks for event building and event filtering can be used. Nevertheless, in all cases a demanding bursty, many-to-one communication pattern develops, because data enter the network synchronously from many readout nodes and have to be routed to a single data collector for further processing as depicted in Figure 1.2.

3.2 DEFINITIONS

Before specifying the requirements on DAQ networks, we explain the necessary terms used throughout this thesis to quantify their performance. We follow the terminology given in [128] and put it in the con-

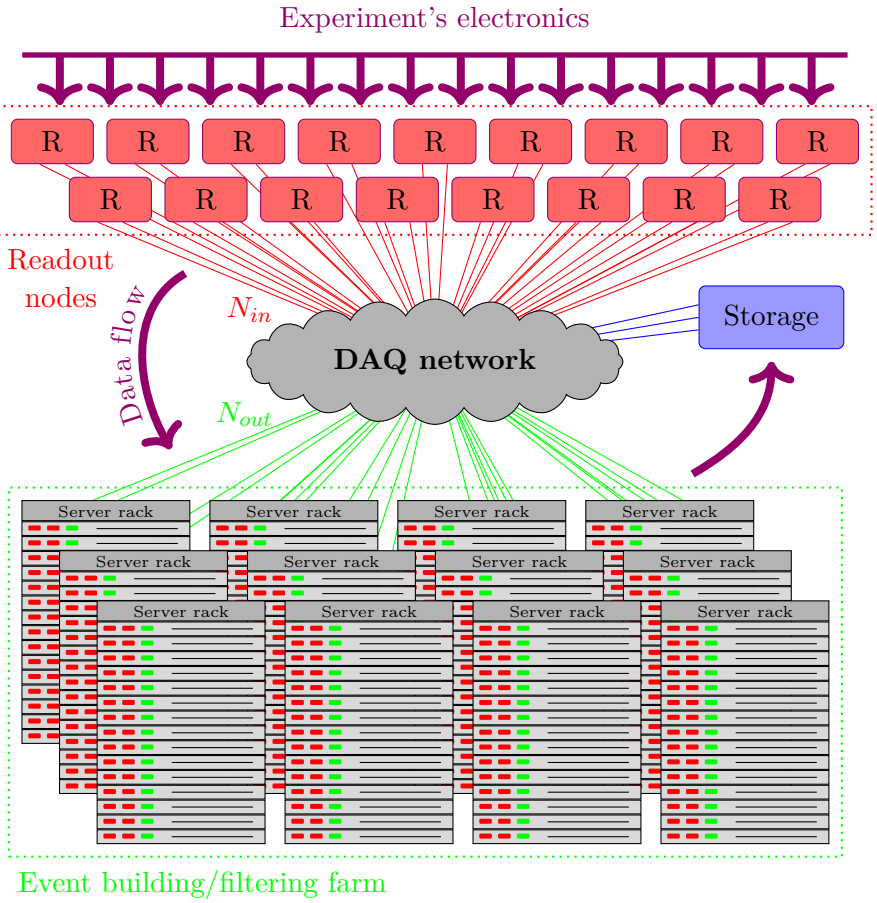


Figure 3.1: Overview of nodes connected to a data acquisition network. Functions of particular nodes can vary. In some configurations readout nodes (R) can also perform event building and/or filtering. Also, distinct networks for both tasks can be used.

text of data acquisition. The definitions are provided in the following list:

- Bandwidth (capacity) B: The data-carrying capability of a network, measured in bits per second (bps). Here, for a particular DAQ network we define it as the maximum aggregate raw bit rate (measured below the network layer and above the physical layer) from the readout nodes to the data collectors. It does not include protocol overheads of the upper layers. In the LHC nomenclature in turn, *data bandwidth* often refers to pure event data rate of an entire DAQ system.

- Throughput T (utilisation U): The quantity of raw bits successfully transferred per unit of time (fraction of bandwidth). Maximum achievable throughput is equal to the bandwidth.

- Goodput G: The quantity of pure event bytes successfully transferred to the data collectors per unit of time. It can be also called

application layer throughput, as it takes into account protocol overheads between the physical and application layers. The goodput of a DAQ network corresponds to the actual data bandwidth of a DAQ system.

- Theoretical goodput G_{theory}: The theoretically calculated maximum goodput of a network.
- Offered load G_{off}: The sum of all the event data that the readout nodes have ready to send at a particular time (absolute value or percent of the theoretical goodput).
- Sustained load G_{sust}: The actual goodput (in the function of the offered load) expressed as the fraction of theoretical value.
- Saturation goodput G_{sat}: Usually it is not possible to reach the theoretical value (even without limits on the offered load) and the goodput saturates at a lower value.
- Maximum loss-free goodput G_{mlf}: The maximum goodput for which no losses occur in the network (where packet retransmissions or other recovery mechanisms are not needed).
- Event size: The total size in bytes of a single snapshot in an experiment. These data are usually distributed across multiple readout nodes (N_{in}).
- Data collection latency l: The time it takes to transfer all data belonging to a particular event from the readout nodes to a collector. In the pull architecture it spans from the moment of sending the request to the readout system for the first fragment to the moment of receiving the last fragment. In the push architecture it is the time between sending the first fragment and receiving the last one.
- Mean l_m and jitter σ_l: The mean of the data collection latency and its variations during some data-taking period.

The formulas used in this thesis to calculate the above parameters can be found in Appendix A.

3.3 REQUIREMENTS ON DAQ NETWORKS

In this section we will discuss high-level requirements for DAQ networks, which we will use later to assess the proposals defined in the thesis. These requirements were already discussed in the context of the LHC experiments in [101, 146]. In general, a data acquisition network as part of a larger data acquisition system must reliably transport in a timely way an experiment's data sustaining some predefined load, which cannot be limited by the specific traffic pattern. Therefore, the key requirements are focused around the following aspects:

- reliability,
- data bandwidth,
- data collection latency,
- scalability,
- fault tolerance, and
- costs.

3.3.1 *Reliability*

Reliability is, next to performance, one of the two most important requirements on DAQ networks. In most experiments full event/process data are required in order to provide scientifically valid results. If a packet is lost in the DAQ network, all data belonging to the same physical process have to be discarded. Therefore, a lossless network is desired for data acquisition. This requirement can be fulfilled by ensuring that the network does not drop any packets and/or by providing a recovery mechanism in case of losses, like the TCP retransmission mechanism. These recovery mechanisms can usually introduce performance degradation as they are used to recover from occasional drops only (e. g. temporal traffic spike in the network or congestion control). They should not be used to provide lossless operation, if systematic packet drops are present. In this case alternative solutions are preferred.

3.3.2 *Data bandwidth*

The required event data bandwidth (as opposed to the network bandwidth) at the input to a DAQ network (i. e. readout system) is usually specified in the requirements of a particular TDAQ system. In the case of the LHC experiments, it is usually given as the event rate, $L1r$ (*level-1 rate*). With this rate events arrive at the readout system and are sent over the network to the data collectors. Together with the average event size[1], e_{avg}, the required goodput, G_{req}, can be calculated as

$$G_{req} = L1r \cdot e_{avg} \,.$$

This is the aggregate goodput that a DAQ network must provide to the application layer in order to move event data from the total of N_{in} ports of the readout nodes to the N_{out} output ports in the filtering farm (see Figure 3.1). Apart from that, each of N_{in} nodes in the readout system requires sufficient input bandwidth to the network. For simplicity, in

1 In some DAQ systems an event can be dropped by the filtering algorithms even before all the data was fetched, like in the ATLAS experiment (see Section 3.5.1). In this case the required data bandwidth is less and can estimated by using an equivalent event size instead.

this work we assume a homogeneous readout system with the same input bandwidth and event fragment size at each node. Also, we set zero processing time at the event building/filtering nodes in order to treat the DAQ network in isolation (see Section 3.5 for more details).

In general, the same network can be also used for the traffic from the event building/filtering farm to storage as well as control traffic of the experiment. In this case, the required data bandwidth is increased. Nevertheless, in this work we neglect these types of traffic as their bandwidth requirements, in many cases, are orders of magnitude lower. The performance of a DAQ network is then primarily determined by the data flow from the readout to the farm.

3.3.3 *Data collection latency*

The mean value and variance (or, in other words, jitter) of the data collection latency are factors important to the performance of an entire DAQ system. Both are correlated with data bandwidth and should be analysed together. Blocking the nodes in the farm from processing because of an increased collection time translates into lost CPU time. It can lead to underutilisation of the entire farm and reduced overall system bandwidth, if the latency increases too much and there is not enough CPUs available in the farm.

For example, in the pull architecture, a node requesting data from the readout system is idled, if the response time exceeds an expected value. Hence, a large jitter prevents us from designing an efficient system, in which data are fetched in advance while still processing previous chunks[2]. Although the mean value should be minimised as well, it is not as critical as the jitter as long as the overall data bandwidth is not impacted [146]. Similarly, in the push architecture, if saturation goodput exceeds the requirements, the mean latency is not critical for the performance. All events are eventually delivered to the farm.

The data collection time of a DAQ network depends on the following components [128]:

1. Serialisation delay: The time required to put the bits to be transmitted onto a transmission line with given speed.

2. Propagation delay: This delay results from the finite speed of light/signal in the transmission medium (the time required to move the bits along the cables).

3. Packet-switching delay: The amount of time that a router or switch needs to move a packet between the input and output ports (without queueing).

2 This can be seen as pipelining and can improve the performance, if data collection latency is comparable with event processing time [146].

4. Network stack delay: The delay introduced in the network stacks of the sending and receiving hosts.

5. Queueing delay: The amount of time a packet waits in the queue of a network device before being switched to the output port. It increases with utilisation.

6. Recovery delay: The time it takes to recover a lost packet.

The first four components sum up to the zero-load network latency, which gives the time it takes for a packet to be moved from a source to some destination in a network without any other traffic. The serialisation delay of the event data traversing the network is the dominant factor, if there is neither contention nor packet losses in the network. As an example, it takes 800 µs to transmit 1 MB (this is the order of the event sizes in the LHC experiments, see Section 2.1) over a 10GbE interface. For typical cable lengths in DAQ networks the propagation delay is counted in nanoseconds, whereas packet-switching delay in modern network devices is just a few microseconds [128]. The delay introduced by the network stacks is dependent on the performance of the servers and by the type of the stack in use. In this work we assume, that this time is also significantly lower than the serialisation delay. The latter gives therefore an estimate of the ideal data collection latency.

This estimate ignores, however, additional delay due to contention with any other collection process running in parallel over shared links in a network. This is the usual scenario in DAQ networks, where event data is moved to hundreds or thousands of independent collectors. In this case the data collection time becomes a function of the offered load [42] and depends also on the queueing algorithms used in switches and routers as well as recovery mechanisms, if there are packet losses. Normally, data collection time increases with offered load for the these collectors that use at least one common link in the network as the flows compete for the access to the same link, as explained in [42]. Eventually, a vertical asymptote is reached for some value of the offered load when saturation goodput of a network is reached. The exact shape of this curve for an entire system depends on various factors, like traffic pattern, queueing algorithms, or topology. This increase in collection time is expected and not problematic, as long as jitter is low and saturation goodput is close to theoretical value.

In general, data collection time is therefore determined by serialisation and queueing delays. The former cannot be influenced without changing the interface speeds. The latter can be minimised by optimising routing across the network, traffic shaping, or adjusting the configuration of the network devices to the extent allowed by their vendors. These optimisations translate directly into increased overall goodput.

Nevertheless, mean latency and jitter can increase substantially, if systematic packet losses occur. In order to guarantee lossless operation, retransmission mechanisms are required. They introduce though

additional delays and waste some of the bandwidth to transmit the same packet again. For this reason, this source of the delay should be eliminated in the first place.

3.3.4 *Scalability*

The evolving requirements of an experiment affect its data acquisition network. It may be required to increase the number of nodes in the read-out system, extend the event building/filtering farm and/or increase the goodput. Therefore, when designing a network, its scalability must be also taken into account.

3.3.5 *Fault tolerance*

In case link or switch failures occur somewhere in the network, they should be detected and alternative paths through the network should be established. In this way the performance degradation can be minimised for the time of repair. In the ideal case, path monitoring and rerouting should be done automatically by the network.

3.3.6 *Costs*

It has been already explained in the introduction to trigger and data acquisition systems in Section 1.1.2 that there is a general desire to build TDAQ systems with COTS equipment. It is also another factor to be taken into account when designing a DAQ network. If a commodity network fulfils all the requirements, it is preferred over specialist technologies. In this case savings can be made on the equipment itself and/or on the costs of hiring experts or providing training on these technologies. The risk that they will disappear from the market in the future can be minimised.

Beyond that, it is also important is to choose solutions based on widely used and supported standards whenever possible, such that not only the dependence on home made solutions is avoided, but also the dependence on single vendors (e. g. InfiniBand). The costs of hiring experts or providing training can be minimised in this case as well.

Not less important are the costs related to power consumption. A commodity solution requiring inadequately large amounts of energy can prove to be more expensive than specialist technologies in the end.

3.4 THROUGHPUT VERSUS LATENCY OPTIMISATION

In the previous section we defined network latency under zero-load, which depends on link speeds, network size and packet processing delays in switches and at end-hosts. It is clear that this latency is dominated by

serialisation delay for the type of traffic in data acquisition, where large amounts of traffic are moved by the network. Therefore, optimisation of network latency does not significantly improve the overall performance as long as serialisation delay dominates other components. Furthermore, if the pipelining or push architecture is used, network latency does not affect the goodput of the system at all.

In many other systems, however, it is usually important to minimise network latency. For example in HPC applications, the overall performance of a supercomputer depends on how fast its processors can communicate with each other [100]. This is the reason for the popularity of InfiniBand networks in the HPC. Low-latency InfiniBand networks provide in general better performance than commodity Ethernet in this respect.

However, network latency under zero-load ignores the delay due to contention with other packets that leads to queueing and/or packet losses. Once this effect is included, latency becomes a function of offered traffic. Therefore, optimising the network under load leads to improving both the latency under-load (in our case, the data collection time) and the throughput (and goodput in consequence) of this network. This is the major area for improvement in case of DAQ because of the demanding communication pattern. In order to avoid ambiguity with zero-load network latency, we use the term throughput optimisation throughout this work[3]. We concentrate on designing DAQ networks on software-based components, which can be jointly optimised for high throughput in many-to-one communication scenarios.

3.5 PERFORMANCE EVALUATION METHODOLOGY

In the previous sections we showed that reliability and performance are equally important in DAQ. Therefore, in order to evaluate solutions for data acquisition networks, we focus on both these aspects. We measure the saturation goodput and compare it with the theoretical value. Then, we analyse latency distributions under different loads. In every case we verify whether systematic packet losses are present in the network.

As most of the work in this thesis is experimental, test equipment was required to apply the desired traffic pattern to devices, and measure their response (i. e. goodput, data collection latency, losses). For the generation of traffic we used the ATLAS TDAQ software in emulation mode. It allowed us to create arbitrary data-taking configurations on a set of computers, generate traffic with many-to-one communication pattern, and record metrics relevant for data acquisition. In the next section we will describe the ATLAS TDAQ system in more detail and give the ATLAS-specific nomenclature for its components. Nevertheless,

3 We use the term throughput optimisation instead of goodput optimisation as we concentrate on the optimisations on the network, not the application, layer.

the conclusions from these evaluations are not limited to ATLAS as long as similar network traffic characteristics apply.

3.5.1 *The ATLAS TDAQ system*

In this section we will extend the description of the ATLAS TDAQ system given in the introduction (see Section 1.1.3) in order to explain the basic nomenclature and configurations used to evaluate the performance of the solutions proposed in this thesis.

The outline of the ATLAS TDAQ system, called ATLAS DAQ/HLT, for the 2015-2018 data-taking period is presented in Figure 1.4. It conforms to the generalised diagram of a DAQ system presented in Figure 3.1. The process of filtering the data, which come from various ATLAS subdetectors, has multiple stages. The first stage, Level-1 trigger, is a dedicated, hardware-based system with very low latency. The events are either accepted or rejected using coarse-grained data from a subset of ATLAS detectors with a frequency of 40 MHz [39]. Then, if an event is accepted by Level-1, its data fragments are distributed (using custom links) to hardware buffers in the readout nodes that are commodity servers. In ATLAS they are called Readout System (ROS) nodes. The total event size is ∼1.7 MB and these data are spread across approximately 100 ROS servers. The ROS can be seen as an interface between the ATLAS-specific data acquisition components and commodity equipment.

The next stage is performed by the event building/filtering farm, which is called the High Level Trigger (HLT). Events are assigned to Processing Units (PUs) by an HLT supervisor (HLTSV). Each PU is a worker process on a commodity server performing event reconstruction and analysis. There are multiple PUs on each server of the HLT farm (working on independent events), but only one Data Collection Manager (DCM) that requests event fragments from ROS on the behalf of the PUs. The DAQ network transports the requests for fragments from HLT to ROS and the requested data from ROS to HLT. The PUs incrementally retrieve and analyse event fragments from ROS until an accept or reject decision can be made. An event can be rejected without analysing all of its fragments. Thus, the required network bandwidth can be reduced. All accepted events are transferred to permanent storage via Data Logger nodes. As of 2014, the HLT was built with ~30000 PUs on ~2000 filtering nodes, but the farm is continuously evolving. For further details please refer to [39, 130].

The architecture of the ATLAS DAQ network is presented in Figure 1.5. It is a 10GbE Ethernet network and a custom, ATLAS-specific application layer protocol implemented on top of TCP/IP is used for event data transport. ROS nodes are connected directly with four 10GbE links to the core of the network, which is built around two large routers with big buffers. Therefore, the maximum theoretical in-

put bandwidth to the DAQ network cannot exceed 4 Tbps. The required average for Run 2 is 400 Gbps [130] though. The four links are used in order to accommodate the redundancy requirement. In the HLT farm, the nodes are organised in racks of at most 40 servers that connect to the core via ToR switches (two 10GbE uplinks to the core, and single 1GbE link to every HLT node). Synchronous requests for event fragments from multiple ROS nodes turn into many-to-one communication pattern. Traffic shaping at the application layer and large buffers in the network core are currently used to avoid incast congestion, as we described in Section 2.1.1. On average, there is $1.7\,\text{MB}/100 = 17\,\text{kB}$ of data belonging to a particular event on a single ROS node, but the spread is large. Each ROS node accommodates up to 24 input links from the detector electronics running at 160 Mbps to 200 Mbps. There are ROSes with as little as a single input link and ROSes hosting 24 links running at almost full speed. It can be also noted that the majority of frames traversing the ATLAS DAQ network carry the maximum allowed number of bytes, which for Ethernet is specified by its MTU (Maximum Transmission Unit) of 1500 B.

Overall, the entire system is operated by a distributed ATLAS TDAQ software. It can be also configured to work in an emulation mode, in which ATLAS specific electronics is not required and events are replaced either by preloaded or dummy data.

3.5.2 *Evaluation procedure*

In the following we will describe the evaluation procedures used for investigating how DAQ networks conform to the requirements defined in Section 3.3. Results of these evaluations for different network configurations, which are the subject of this thesis, will be presented in the following chapters.

The generalised evaluation setup used throughout this thesis is presented in Figure 3.2. It consists of two main parts:

1. Traffic generation,
2. Network under test.

ATLAS TDAQ software in emulation mode is used for traffic generation. Readout applications (also called ROS applications) deliver dummy event data that are requested by DCMs in an emulated HLT farm. Each node is a commodity server and runs readout applications, data collectors (emulated HLT rack of DCMs), or both in parallel to imitate a larger data collection configuration. Dummy events are assigned to DCMs by the HLTSV with controllable rate, which defines the offered load. DCMs request event data fragments from ROS, but a single DCM does not request another event before it receives all fragments of the previous one from all available ROSes (in all our tests a single DCM requests data on behalf of one dummy PU only, unless

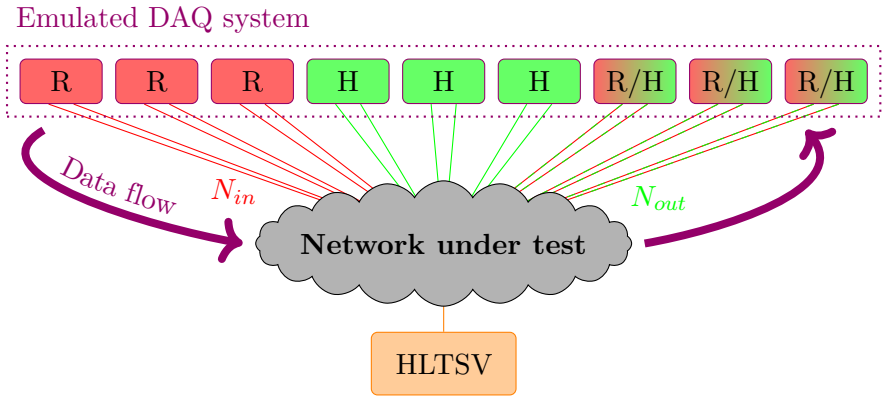


Figure 3.2: Generalised evaluation setup. Both readout system (R) and filtering farm (H) are emulated with the ATLAS TDAQ software. Each box represents a distinct node running either emulated readout applications, emulated rack of data collectors of the HLT farm, or both.

otherwise stated). Only full event building is performed, and all events are rejected without any processing. In this way the DAQ network can be analysed in isolation from other factors. As a result, there are many interleaved all-to-one communication patterns or a single, fully meshed all-to-all traffic.

All tests are performed with event sizes that are large enough to assume that most frames will be of maximum size allowed for Ethernet (MTU of 1500 B). But the exact size of event fragments can vary in different tests. As explained in the previous section, large frames can be considered typical in data acquisition. The general performance of software switches for small packets is broadly studied in the literature (see Section 2.3).

If we need to run the network so that congestion control is not the limiting factor, we disable dynamic TCP congestion control in all ROS nodes and instead use a static sender congestion window (see Section 4.5.2.1). This window can be set to a very large value so that each ROS response is not rate-limited by TCP, further increasing the incast effect. It also allows us to evaluate the performance of our prototype networks without the influence of the congestion control and test in the best scenario from the viewpoint of data acquisition simply pushing all the available data on to the wire. We further simplify the system by disabling TCP selective acknowledgements (TCP SACKs) for most of the tests, so that it resembles a simple, UDP-like, push scenario. Normally, this mechanism improves the performance by limiting the number of retransmitted segments in case only some of them are lost in the network [55].

For our evaluations the following metrics are calculated or measured (following the definitions in Section 3.2):

1. The theoretical bandwidth and goodput for a given network configuration (see Appendix A for exact calculations). As the underlying protocols remain the same in every test, the considered protocol overheads come from Ethernet, IP, TCP and ATLAS data flow protocols.

2. Saturation goodput and, for some tests, maximum loss-free goodput (when HLTSV does not limit event rate).

3. Sustained load and distribution of data collection latency as a function of offered load as set in HLTSV.

4. For some tests, TCP retransmissions as an indication of packet losses in the network. In most cases though, systematic packet losses result in degradation of the sustained load and increased data collection latency and jitter.

Goodput is calculated from the average sustained event rate as reported by the HLTSV, whereas histograms for data collection latency are recorded by the ATLAS TDAQ software. Histogram granularity is 1 ms. For a single event, data collections latency is the timespan between sending the first request to the ROS and receiving the last data fragment from the ROS. TCP retransmissions are summed over all ROS nodes from the numbers reported in Linux network statistics during the measurement. All these metrics are measured in the so called *steady-state* [42, 131], when a network has reached equilibrium. In each test iteration stabilisation time is 360 s and the measurement time is 120 s.

3.6 CONCLUSION

In this chapter we have defined and discussed basic definitions, metrics and performance evaluation methodology for data acquisition networks, which will be used in the following chapters of this work. In data acquisition, reliability and performance are equally important. On one hand, losses in event data imply incomplete reconstruction of the physical process, and, in effect, oversight or disqualification of potential discoveries. On the other hand, the systems need to perform well at very high rates. These two requirements are often difficult to achieve in parallel because of the many-to-one communication traffic pattern, often accompanied by high burstiness, as we will show in the next chapter.

4

MANY-TO-ONE PATTERNS IN DATA ACQUISITION

With this chapter, the original contributions of this thesis begin. We first focus on the TCP/IP performance in data acquisition networks and discuss some analogies and differences between them and datacenter networks. Then, we give a high-level analysis of the problem and, based on a simple estimate, we introduce what are the general approaches to solve incast pathology. We present evaluation and comparison for some of the related techniques, which are used in datacenter or in data acquisition networks. We also include a short discussion on the state of the art TCP variant for datacenters, DCTCP, and its potential use in data acquisition. This chapter contains results published in [87].

4.1 INTRODUCTION

In the previous chapter we presented in more detail typical configurations of DAQ systems. In all those cases, a demanding bursty, many-to-one communication pattern develops, because data enter the network synchronously from many readout nodes and have to be routed to a single data collector for further processing, as was depicted in Figure 1.2. This pattern can have catastrophic consequence on performance, particularly for TCP-based communication. It leads to so-called incast congestion, if traffic bursts are too large to fit into the buffers of switches and routers.

4.2 TCP PERFORMANCE IN DAQ NETWORKS

Designing a DAQ network with hundreds of overlapping many-to-one communication patterns is challenging. We indicated this already in the introduction to this thesis, particularly in Section 1.1.2.1. But the consequences of incast congestion in this scenario are already visible when only one communication process of this type is present.

In order to illustrate them we use an emulated data-taking session with a single data collector that requests data from a controllable number of readout nodes (maximum of 200 nodes). The experiment uses the real ATLAS DAQ network, see Figure 1.5. The readout system is emulated though on the HLT farm (*Tdaq PU* in Figure 1.5) in order

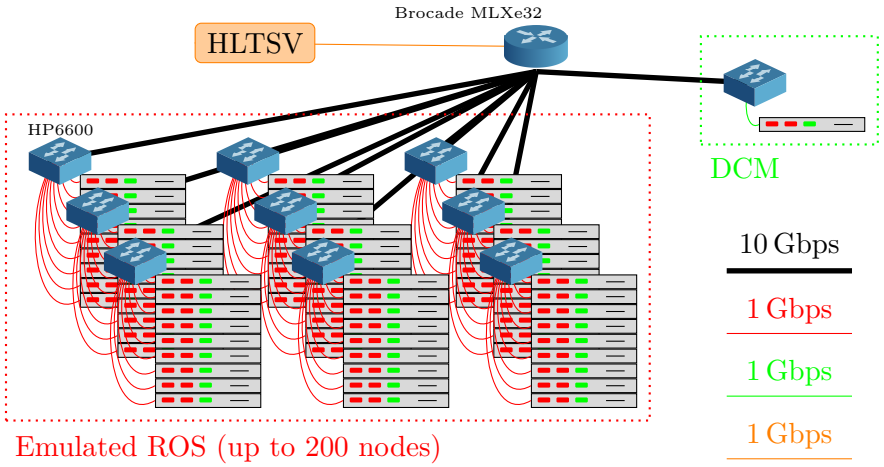


Figure 4.1: Test setup with one data collector and up to 200 ROS nodes (9-12 per rack) with twelve event data fragments on each.

to have better control on the total number of nodes. The actual configuration for this test is presented in Figure 4.1. HP6600 ToR switches [72] are used to aggregate machines within the same rack. The core of the network is made with the Brocade MLXe32 router [28] using the Virtual Output Queueing (VOQ) architecture [99]. ToR switches and the core router are connected via 10 Gbps uplinks. All connections within the rack are 1 Gbps. ROS PCs are physically located in different racks (9-12 nodes in each) so as not to limit the system's bandwidth by the speed of the links from the ToR switches connecting ROSes to the core router. There is 12 kB of data (twelve distinct fragments of 1 kB each) for a single event on each emulated ROS and the total event size is increasing with the number of nodes used in a given iteration up to 2.4 MB. The offered load is set in such a way that the event data rate does not exceed the 1 Gbps link to the only data collector, even for the largest set of ROS nodes ($L1r = 50$ Hz). This test is performed with a traditional, unreliable Ethernet network and with TCP as the transport layer protocol with the default, TCP cubic, congestion control algorithm.

What can be seen in Figure 4.2 is the link utilisation of this single data collector over a longer period of time. A single event is built out of data fragments distributed across 200 ROS nodes in this case. There are multiple long periods of time when the link remains idle for at least 200 ms. This is typical observation for TCP-based applications that generate many-to-one traffic bursts in a network. When a data collector requests full event data from all the ROS nodes simultaneously, it causes a large traffic burst. Switches without sufficient buffer capacity drop some portion of the packets.

In this particular case it happens at the ToR switch connecting the data collector to the DAQ network, see Figure 4.1. Although the speed of this link is 1 Gbps, which is enough to sustain the requested load,

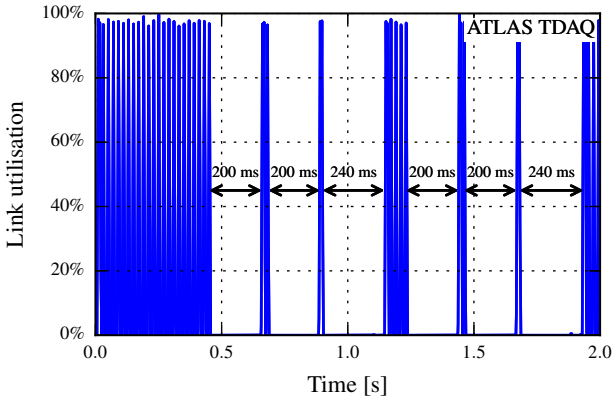
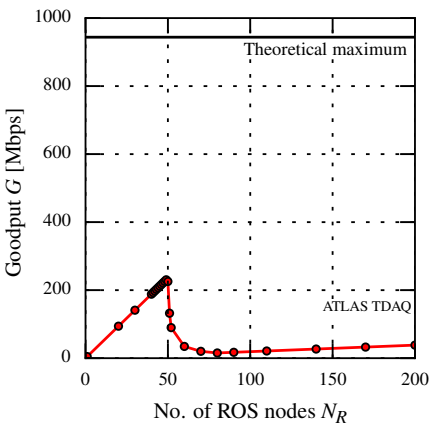


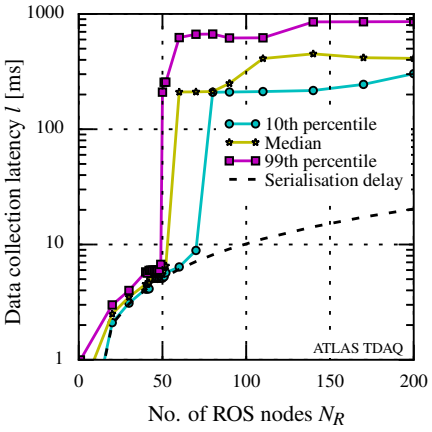
Figure 4.2: Link utilisation of a data collector over longer period of time. It remains idle for at least 200 ms after a TCP timeout, which suspends data collection.

there is a 10 Gbps uplink to the DAQ network, over which requested event data are fetched. Some of the packets are dropped because of this oversubscription and the fact that frames carrying requested event fragments from various ROSes cannot fit into the memory of the ToR switch. In this case Ethernet, as a best-effort protocol, is allowed to drop packets. These losses trigger standard TCP recovery mechanisms. Because particular ROS-to-DCM TCP flows are relatively small (approximately nine frames are required to move event fragments from a single ROS), most of the lost packets are recovered using TCP timeout mechanism, and not the so called *fast recovery* [55]. If a packet is lost and there are no more packets of the same flow arriving at a TCP receiver, a TCP sender must wait for a TCP timeout before retransmitting a lost segment. These timeouts introduce delays of hundreds of milliseconds, compared to three orders of magnitude lower 200 µs RTT of the DAQ network or to one order of magnitude lower serialisation delay ($\frac{200 \cdot 12\,\text{kB}}{1\,\text{Gbps}} = 19.2\,\text{ms}$ for sending event data, without protocol overheads, over 1 Gbps link). The typical value for the minimum TCP retransmission timeout (RTO) in TCP stack implementations of common Linux distributions is 200 ms [171] (including Scientific Linux 6, SLC6 [149], used at CERN). While the actual value changes during the lifetime of a TCP connections based on the measurement of the experienced RTT, a minimum value is usually set by the operating system [55]. This is the reason for the large jitter of data collection latency.

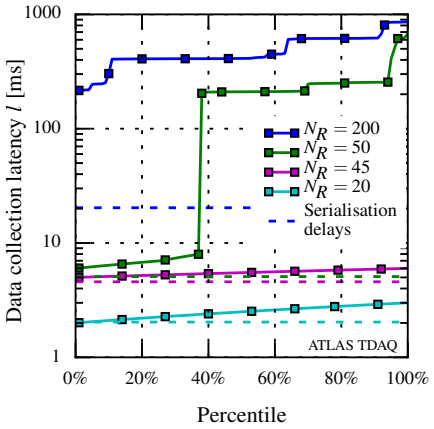
Figure 4.3 shows the goodput and latency characteristics when increasing event size by increasing the number of ROS nodes that are used to build an entire event. Goodput increases first linearly with the size of the event. This is expected as the event rate is kept constant and only the event size changes. Data collection latency also matches expectations as it is close to the serialisation delay of the entire event and protocol headers over a 1 Gbps link. But for more than 50 ROSes goodput suddenly collapses. At this point the burst of packets coming



(a)



(b)



(c)

N_R	l_m [ms]	σ_l [ms]
240	2.5	0.0
540	5.5	0.0
720	164.4	146.0
2400	487.4	155.3

Figure 4.3: Goodput (a) and event data collection latency (b) when increasing event size by increasing the number of readout nodes, and the exact distributions of latency for four different event sizes (c). Event rate at the input is kept constant ($L1r = 50\,\text{Hz}$).

from ROS exceeds the size of the buffer at the ToR switch connecting the DCM to the network. From this it can be estimated that the HP6600 ToR switch has about $50 \cdot 12\,\text{kB} = 600\,\text{kB}$ buffer per 1 Gbps output port.

The latency distribution shows that an increasingly large amount of events suffer from data collection latency of more than 200 ms, which is caused by TCP timeouts as explained in the previous paragraph on the example of $N_R = 200$. As a consequence, the mean value and jitter are significantly elevated. Because of the latter the network becomes highly unpredictable, so efficient data acquisition process cannot be established.

As can be seen from Figure 4.1, a large oversubscription happens already at the core layer of the network, where multiple ROS racks

are connected to a single router. Packets from many 10 Gbps links are switched to a single 10 Gbps link towards the DCM. This router has, however, large enough buffer to accommodate the entire burst from ROS, so packet losses occur first at the ToR switch connected to the DCM. If a commodity router or switch was used instead, congestion would have happened already at the core layer. This behaviour can be emulated by changing the size of the queues at the router, which is presented in Figure 4.4. Event rate at the input is kept constant and the number of ROS is set to 40 in order not to overflow the ToR switch at the data collector. When the buffer size in the core router is not sufficient, packets are lost again, even though the system should operate in steady state as shown in the previous paragraph with $N_R = 40$. Similarly as with drops in the ToR switch, large data-collection mean latency and jitter are observed. When the queue size is large enough, requested load is sustained and latency is close to the serialisation delay. It is akin to an observation about the amount of buffering in the networks that was made in 1997 by Morris in [111], who claimed that it should be proportional to the total number of active TCP flows[1].

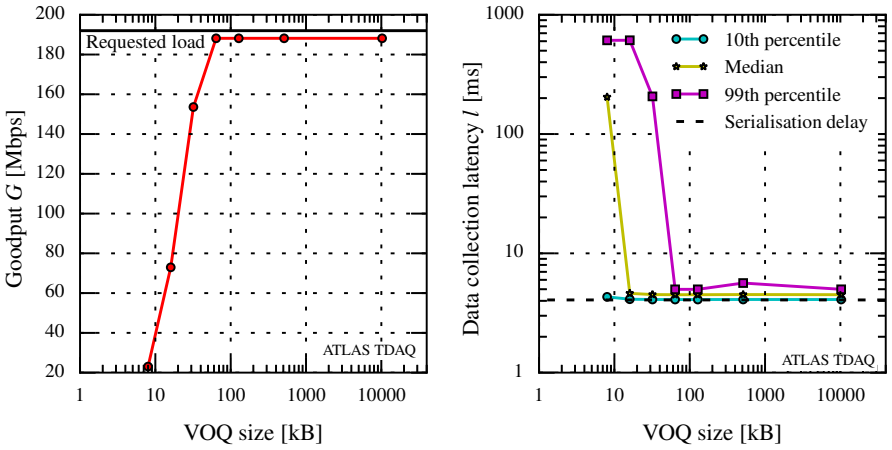
But the problem is even more complex, when the entire filtering farm becomes active. In this case, hundreds of overlapping many-to-one patterns emerge. The exact system behaviour and the severity of the problem depend on many factors like the DAQ system configuration and its size, event data distribution, network topology and protocols, amount of buffering available in the network, or queueing mechanism in switches and routers. In any case, many-to-one communication patterns are prone to systematic packet losses. A deterministic retransmission process could be achieved with fine-grained TCP RTO as proposed by Vasudevan et al. in [171]. But network bandwidth would be systematically wasted for these retransmissions, reducing its overall efficiency. If possible, packet drops should be therefore eliminated in this first place. General approaches to incast mitigation will be discussed in Section 4.4.

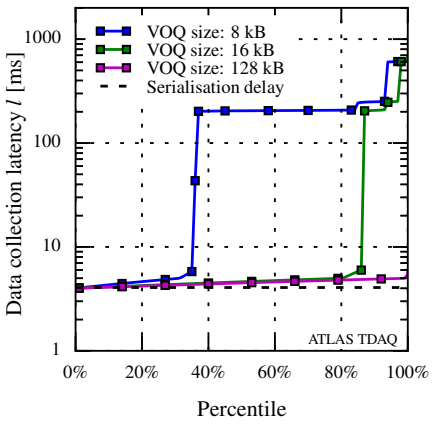
Summarising, many-to-one communication is a significant challenge in data acquisition networks. It is particularly demanding for Ethernet networks, where packets can be dropped as a response to congestion. The problem can also occur when other technologies are used, but the symptoms may be different. Furthermore, many-to-one patterns are a challenge not only in data acquisition, which we will show in the next section on the example of datacenter networks.

4.3 THE ANALOGIES AND DIFFERENCES TO DCN

As it turns out, there are strong analogies between DAQ and datacenter networks in the context of many-to-one communication. Synchronous

1 This observation stands in contrast to the current trend to reduce the amount of buffering in the backbone routers when increasing the number of competing flows [9].





VOQ size	l_m [ms]	σ_l [ms]
8 kB	163.8	154.6
16 kB	44.8	115.3
128 kB	4.5	0.0

Figure 4.4: Goodput (a) and event data collection latency (b) when changing the queue size in the core router, and the exact distributions of latency for three different cases (c). Event rate at the input is kept constant ($L1r = 50\,\text{Hz}$) and the number of ROS is $N_R = 40$ in order not to overflow the ToR switch at the data collector.

requests for event data fragments from a single data collector to a large number of readout nodes closely resemble the behaviour of some applications in datacenters, which are also affected by the incast pathology [36]. The typical examples of datacenter applications that are susceptible to incast are [177]:

- Cluster storage, in which servers concurrently respond to a request from some client for a block of data that is striped across multiple nodes [64]. It was in a paper on cluster storage [114] where incast was first reported by Nagle, Serenyi, and Matthews.
- Web search, where search queries are returned near simultaneously by many workers.

- Batch processing (e.g. MapReduce [45]), where intermediate results from many mappers are transferred to reducers.

Similarly as in data acquisition networks, high-speed links and low RTTs are typical for DCNs [142], so similar observations regarding TCP performance are made. Solutions from datacenters could be therefore tested in DAQ systems and, if successful, would help in increasing their level of commoditisation. We reviewed the literature on the techniques proposed for datacenters to solve or reduce the impact of TCP incast in Section 2.2. We will also analyse some of them in Section 4.5.

Nevertheless, there are also many differences between datacenter and DAQ, which can make the adoption of some of the solutions easier in one domain than the other. This is the case with the attempt to provide large network buffers with software switches, which is the core of this thesis. These differences justify our choices with regard to the specific design of the software switch (see Chapter 5) and, in the first place, the use of extremely large buffers. We also show that the potential bottlenecks of software switching [52] are insignificant in the case of traffic patterns typical for DAQ.

FARM SIZE: The DAQ farm of the ATLAS experiment consists of ~2000 servers filtering the LHC data and 100 servers in the ROS, compared to typical DCNs containing hundreds of thousands of servers [177]. The forwarding tables (containing entries to find the proper output interface for a packet) in the network are therefore relatively small. In software switches it costs some CPU cycles to perform lookups in those tables. Thus, evaluation of software switches should normally contain the analysis of performance dependency on the table size. In case of DAQ networks this effect is negligible.

WORKLOAD: DAQ networks are active during data collection period, which for the LHC experiments is counted in months and years. They are exclusively dedicated to deliver the event data from the readout system to the event building/filtering farm and they are isolated from any other flows, including the control traffic. The consequences are twofold. First, there are no flows in the DAQ whose latency could suffer from the large network buffers. Thus, the bufferbloat problem [63] is not an issue. Second, the network can operate in a nearly static configuration for a very long time. This configuration can be set at the beginning of the run and tailored for the current data taking period. In contrast, network usage in DCNs changes dynamically. Typical traffic in DCNs is traditionally classified into short-lived mice and large, long-lived elephant flows [157]. Any solution to the incast-vulnerable traffic flows must not disturb the fairness of all other traffic in the network, which may make the design more complicated.

PACKET SIZES: In Section 3.5.1, it was shown on the example of the ATLAS experiment that event data traversing the network are generally large in relation to the Ethernet's minimum frame length of 64 B. The performance analysis of a network can be therefore limited to large packets, which makes the adoption of software switches significantly easier than in datacenters, where maximum forwarding performance for small packets is also required because of the variety of traffic patterns. In case of software switching, the forwarding performance is mostly limited by the maximum packet rate and not the packet size. It is therefore easier to saturate a link with large packets. It is also worth noting that, in general, DAQ networks do not send or receive packets from untrusted sources, and so attacks on the network are less likely. For example, no one will ever generate requests resulting in many small packets to perform a denial-of-service attack.

SWITCHING LATENCY: One of the important performance indicators of a network switch or router is its latency. It is particularly critical in high frequency trading or high performance computing. In data acquisition, due to the large event sizes, the total data collection latency is dominated by the serialisation delay (see Section 3.4). Minimising the switching latency is therefore not critical in DAQ as long as it is kept within a reasonable range. This is another factor simplifying the adoption of software switches in DAQ.

4.4 GENERAL APPROACHES FOR MANY-TO-ONE COMMUNICATION

The many-to-one communication pattern leads to incast congestion in high-bandwidth and low-latency networks, when the senders collectively send enough data to surpass the buffering abilities of a network. Incast is studied mostly in the context of TCP protocol, when its consequences are catastrophic, as we showed in Section 4.2. Detailed analytic models to predict incast behaviour and the impact on the application performance are available in the literature (see Section 2.2.2). In this work, we concentrate on eliminating the consequences of incast congestion in the first place, not reducing or analysing its exact impact. As it was explained in Section 2.2.2, incast congestion is persistent in data acquisition. A solution to provide lossless operation is therefore needed to provide the highest utilisation of the resources. In order to derive an estimate showing when lossless operation in a network is possible we will use the typical term describing network capacity — the bandwidth-delay product.

4.4.1 *The bandwidth-delay product*

The bandwidth-delay product (BDP) of a network path determines the amount of data that can be stored in the network in transit to the receiver [55]. It is often considered as an important quantity when estimating the buffering required for TCP. BDP is calculated as a product of the network delay and the speed of the bottleneck link on the path from sender to receiver:

$$BDP = B_{bottleneck} \cdot delay\,. \tag{4.1}$$

Depending on the context, *delay* means usually the RTT or one-way latency of a network. The former is used when the receiver signals to the sender that the data is arriving (e. g. TCP) [133]. In general, the network utilisation is maximised when a sender transmits data, once per delay period, at least as large as the BDP. If the value is exceeded, these additional bytes need to be buffered in the network and queueing delay occurs. Otherwise, the network remains underutilised. This corresponds to the general recommendation for capacity planning for each link's buffer in a router [14], being $2 \cdot T \cdot C$. T is the single direction latency and C is the transmission capacity of the link.

As an example, the BDP of the ATLAS DAQ network (see Section 4.2) is

$$BDP = 1\,\text{Gbps} \cdot 200\,\mu\text{s} = 25\,\text{kB} \approx 16\,\text{Packets}$$

with a single Ethernet frame requiring 1538 B on the wire for maximum sized TCP segments (see Appendix A.2) and considering the 1 Gbps link from a ToR switch to a DCM as the bottleneck. This means that 100 ROS nodes sharing this link, when sending a single TCP segment to one data collector, would exceed the network's BDP by a factor of six. This excess must be buffered in the switches, otherwise packet drops are observed.

If we now consider the 10 Gbps uplink from a ToR to the core router to be the bottleneck, the BDP raises to

$$BDP = 10\,\text{Gbps} \cdot 200\,\mu\text{s} = 250\,\text{kB} \approx 162\,\text{Packets}\,,$$

but there are now 4000 independent flows using this link (40 data collectors connecting to 100 ROSes). With one packet on each flow the BDP is exceeded by a factor of 25, so even more buffering is required in the core of the network.

4.4.2 *The onset of incast congestion*

The calculations in the previous section can be generalised and used as an estimate on when the onset of incast can occur in some network. Let's analyse a network with overlapping many-to-one communication

patterns, as presented in Figure 3.2. A single sender does not inject into the network more than the sender window W_{ij} of bytes on a single flow from the i-th readout node to the j-th data collector. Subsequent bytes can be sent after the first set has been successfully received by the data collector or the window was not fully used. The sender window can be subject to some congestion control algorithm and can vary over time. In the worst case, when all data transmission occurs synchronously on all flows sharing some bottleneck link in parallel, the total number of bytes in transit over this path is

$$W_{global} = \sum_{i=1}^{N_R} \sum_{j=1}^{N_D} W_{ij} \,, \tag{4.2}$$

which we refer to as *global window*. This path is optimally utilised, if this global window is at least of its BDP[2]. On the other hand, if BDP is exceeded beyond the available buffering capacities of the network, packets are dropped and incast congestion can occur. In order to guarantee lossless operation and optimal utilisation the condition

$$BDP \leq W_{global} < BDP + Q_{bottleneck} \tag{4.3}$$

has to be fulfilled, where Q denotes the total buffer space available in the queues of a network for those flows at the bottleneck link. Depending on the network topology this condition should be normally analysed taking into account different bottleneck links, like we did in the previous section.

We assumed here that there are no other flows in the network, which is valid, in general, in data acquisition (see Section 3.3), but not in datacenters.

4.4.3 *Incast avoidance*

Combining the inequity (4.3) with (4.1) we obtain

$$B_{bottleneck} \cdot delay \leq W_{global} < B_{bottleneck} \cdot delay \; + Q_{bottleneck} \,, \tag{4.4}$$

which points to three general approaches to prevent incast congestion and provide lossless operation:

1. Increasing bottleneck link speeds $B_{bottleneck}$.
2. Increasing buffering capabilities of a network $Q_{bottleneck}$.
3. Controlling the amount of packets entering a network W_{global}.

2 The meaning of *delay* in equation (4.1) depends on the protocol in use. For TCP it should be the RTT of the network, but for a simple, UDP-like push design it should be the latency from senders to the bottleneck link.

The first solution is neither efficient nor scalable, since high speed links are expensive and network would not be fully utilised on average. With more bandwidth on the bottleneck link less pressure can be indeed put on the switch buffers, as data bursts could be drained to the output faster. But on the other hand, this additional bandwidth is used to accommodate temporal overcommitments only, so the average network bandwidth is not improved (left-hand side of equation (4.4) would be violated on average). Furthermore, any excess of the available bandwidth could be used for increasing the overall system throughput (i. e. increasing the event rate and/or size) instead, if another solution is available. This point is particularly important when facing the requirements of the LHC upgrades as explained in Section 2.1.

Although effective, extending buffer sizes in switches is also the least scalable approach as networking hardware with larger memory costs significantly more and cannot be regarded as commodity. No less important, the queueing mechanisms employed in routers are specific to their manufacturers and have limited reconfigurability. We explained those aspects already in the motivation of this thesis in Section 1.2. In the following chapter we will present an alternative, which can significantly reduce costs and tailor the queueing algorithms to the needs of data acquisition.

The third path includes all technologies that provide congestion and flow control mechanisms, which control the data injection rate into a network to avoid congestion while keeping it fully utilised. Promising alternatives range from the link layer trough the transport layer up to a dedicated application layer solution. A detailed review of the alternatives is provided in Section 2.2. It should be noted, however, that even with a perfect solution for incast congestion, the buffer pressure would move back to the source hosts, where more data would await transmission over the network. As a consequence, buffering in the readout system has to include additional memory to handle network congestion. Computational complexity could also increase, if this congestion control is performed at the senders. This prevents the use of push architectures with simplest readout nodes. Despite those limitations, many mechanism exist and can be used improve the performance and reduce buffer requirements of a DAQ network. In the following section we will evaluate example solutions for TCP-based applications.

4.5 EXAMPLE SOLUTIONS FOR TCP INCAST

In this section we focus our attention on some solutions that prevent or mitigate TCP incast by limiting the number of packets that are transmitted over a network and maintain high network utilisation at the same time. This can be achieved by means offered by different network layers.

4.5.1 *Application layer solutions*

One of the ways to control the global send window in order to keep optimum network utilisation, as defined by the condition (4.4), is to provide appropriate means directly in the application. A successful implementation is the traffic shaping algorithm used at the ATLAS experiment [39, 152]. References to other examples were provided in Chapter 2.

In ATLAS's traffic shaping, each of the data collectors in the HLT farm is assigned a fixed amount of credits. One credit permits a request for one event fragment. A DCM does not request more fragments than its currently available quota. The quota is reduced when requests to the ROS are sent and increased upon receiving the response with event data. Here, we provide sample results for the traffic shaping algorithm. A detailed analysis is provided in [39].

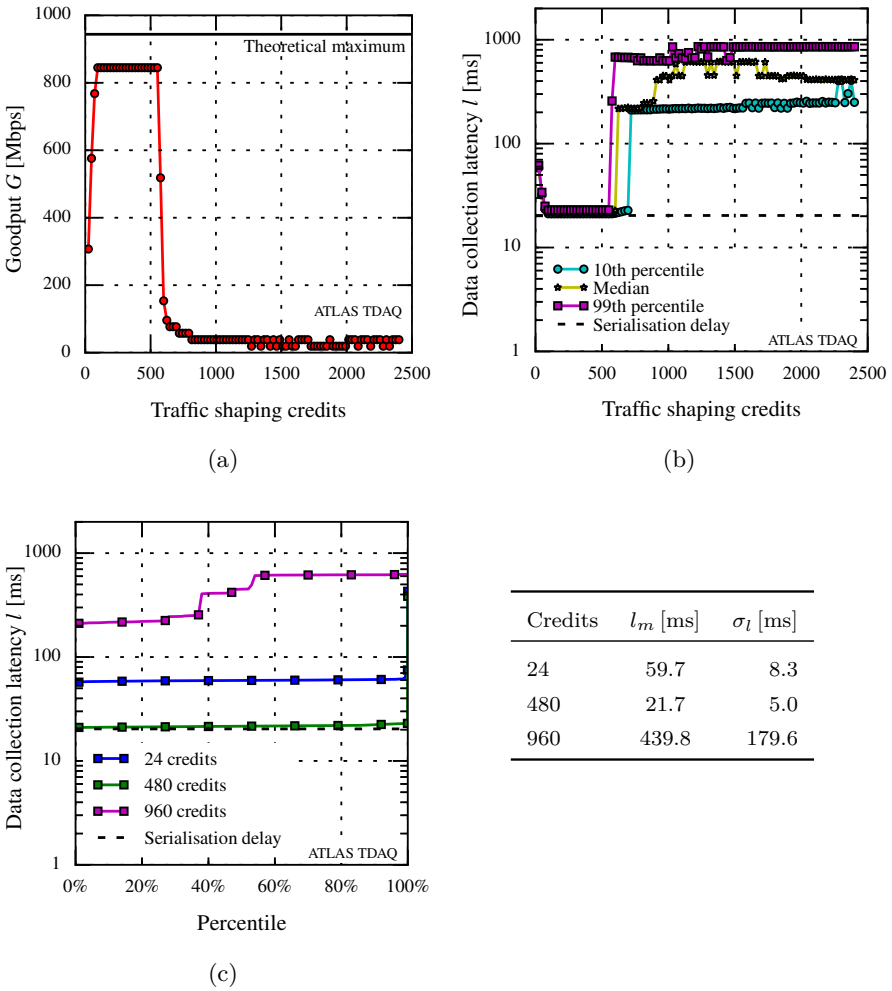
EVALUATION SETUP We use the same configuration as in Section 4.2, which was depicted in Figure 4.1. A fixed number of 200 ROS nodes is used in this case. Each of them provides twelve event fragments of 1 kB. The theoretical bandwidth (see equation (A.7)) is

$$\begin{aligned} B_{inout} &= \min\left(N_R n_R b, N_H n_H b\right) \\ &= \min\left(200 \cdot 1 \cdot 1\,\text{Gbps}, 1 \cdot 1 \cdot 1\,\text{Gbps}\right) \\ &= 1\,\text{Gbps}\,. \end{aligned}$$

On the other hand, the theoretical goodput, calculated with equation (A.12), is $G_{theory} = 943.8\,\text{Mbps}$. The offered event rate is set to 50 Hz, slightly above the theoretical value (102 %).

EVALUATION RESULTS The capability of traffic shaping to mitigate TCP incast is demonstrated in Figure 4.5. In the optimum operating range, between 100 and 550 credits, the mean latency of event data collection stays near its minimum of 21.7 ms. It is close to the serialisation delay of the entire event size, including protocol overheads, on a 1 Gbps link of 20.3 ms, thus proving the efficiency of the algorithm. Below 100 credits the network is not fully utilised as we stay on the left-hand side of the inequity (4.4). Mean latency is increased, but there are no signs of TCP retransmissions. On the other hand, the switch buffers are overstressed and start to drop packets above 550 credits quota (right-hand side of the inequity (4.4)). For example, for 960 credits, every event data collection process suffers from at least one TCP timeout as there are no events with latency lower than 200 ms (see Figure 4.5c).

Analogous plots can be established for different configurations. In this way, the traffic shaping algorithm can be tuned to provide best results. The algorithm is particularly efficient in avoiding incast in the last hop before data collectors because traffic is controlled on per DCM basis. Large buffers are still required in the core of the network (as in the



Credits	l_m [ms]	σ_l [ms]
24	59.7	8.3
480	21.7	5.0
960	439.8	179.6

Figure 4.5: Goodput (a) and event data collection latency (b) when changing the quota of traffic shaping credits per data collector, and the exact distributions of latency for three different cases (c). Event rate at the input is kept constant ($L1r = 50\,\text{Hz}$) and the number of ROS is $N_R = 200$ in order not to overflow the ToR switch at the data collector. Each ROS provides twelve event fragments of 1 kB

ATLAS DAQ network, see Section 3.5.1), which has to sustain traffic flows to multiple HLT racks. The credit quota cannot be decreased beyond one, so if the number of DCMs requesting a single fragment in parallel is already enough to overflow the buffers in the network core (it is actually also possible in the last hop for ToR switches with tiny memories), the algorithm will fail.

4.5.2 *Alternative TCP congestion control algorithms*

TCP congestion control mechanisms are used to control the amount of packets that are injected into a network. This is realised by controlling

the evolution of the so called *send window*, which limits the packets that can be simultaneously injected into the network on a single TCP flow. In Linux, different congestion control algorithms can be used for each connection. The user can choose from a number of alternatives. In the distribution popular at CERN (SLC6 [149]) the default is TCP cubic [55]. It is also possible to implement custom modules that can be easily loaded into the kernel without its recompilation. Many solutions for incast-avoidance were therefore proposed in the form of customised TCP congestion control modules, which we reviewed briefly in Section 2.2.2. Here, we discuss the possibilities offered by TCP congestion control in the context of data acquisition.

A single TCP sender is allowed to send no more than the send window W of unacknowledged bytes, which is the minimum of sender's congestion window ($cwnd$) and the receiver's advertised window ($awnd$) [55]:

$$W = \min(cwnd, awnd)\,.$$

The optimal window is about the size of a network's BDP. It ensures that enough bytes are in-flight to keep its links busy as described in Section 4.4.

For TCP-based DAQ networks, the condition (4.3) together with equation (4.2) can be therefore rewritten as

$$BDP \leq \sum_{i=1}^{N_R}\sum_{j=1}^{N_D} \min(cwnd_{ij}, awnd_{ij}) < BDP + Q_{bottleneck}\,, \quad (4.5)$$

where $\min(cwnd_{ij}, awnd_{ij})$ is the send window of the flow from i-th ROS to j-th data collector.

4.5.2.1 *Static congestion window*

A simple static configuration of the send window (*static TCP*) in ROS, $cwnd_{ij} = cwnd$, can be a potential solution to TCP incast. The $cwnd$ can be easily configured with a trivial Linux kernel module (available at [43]). Assuming a large advertised receiver's window, $awnd_{ij} > cwnd_{ij}$, on all HLT hosts, the global send window simplifies to

$$\begin{aligned} W_{global} &= \sum_{i=1}^{N_R}\sum_{j=1}^{N_D} \min(cwnd_{ij}, awnd_{ij}) \\ &= \sum_{i=1}^{N_R}\sum_{j=1}^{N_D} cwnd_{ij} \\ &= \sum_{i=1}^{N_R}\sum_{j=1}^{N_D} cwnd \\ &= N_R N_D\, cwnd\,. \end{aligned}$$

and the condition (4.5) for $cwnd$ of a single flow becomes:

$$\frac{BDP}{N_R N_D} \leq cwnd < \frac{BDP + Q_{bottleneck}}{N_R N_D}$$

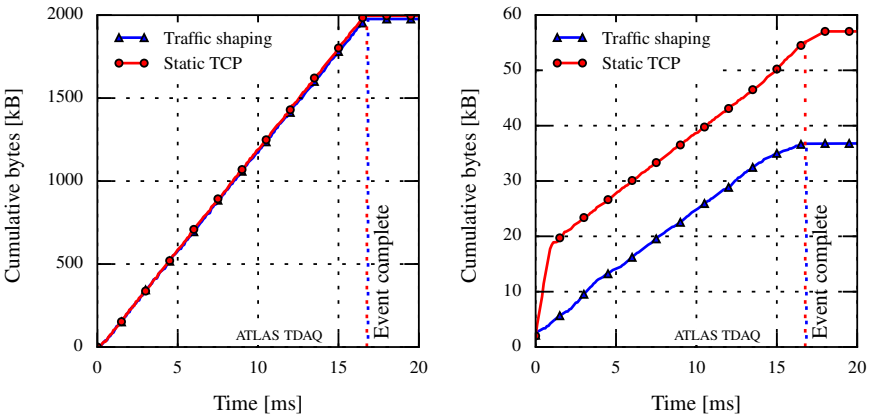
This inequity can be used to approximate the required amount of buffering. With fixed *cwnd* we guarantee that there will be no more than $N_R N_D$ *cwnd* packets[3] in-flight in a DAQ network.

EVALUATION SETUP The setup that is used to evaluate this approach is a similar configuration as in Section 4.2 (see Figure 4.1). Here, we extend the study and provide results and comparison with traffic shaping. We analyse a single event collection process and the performance of a single data collector. A fixed number of 147 ROS nodes is used in this case. Each of them provides twelve event fragments of 1.1 kB. The total event size is therefore 1.9 MB. The theoretical goodput is now $G_{theory} = 943.4$ Mbps, but we keep the offered event rate at 50 Hz, which for this configuration is 82 % of the theoretical maximum. In this evaluation we focus on latency characteristics at high load, just below saturation.

EVALUATION RESULTS The congestion window *cwnd* is fixed at each ROS to two packets to provide optimum performance. With 147 ROSes there are no more than $147 \cdot 2 = 294$ packets (or 441 kB for packets with an MTU of 1500 B) traversing the network in parallel. It is enough to provide full network utilisation (BDP is approximately 16 packets, see Section 4.4). On the other side, it is not enough to overrun the ToR switch buffers, which we estimated to about 600 kB in Section 4.2. In Section 4.5.4, using a different setup, we will provide a comparison between different technologies. There, we will also show how the performance changes for different congestion windows.

The collection process of a single event Figure 4.6 shows the IO behaviour of traffic shaping and static TCP configuration when collecting a single event. These data were generated from packet dumps on a DCM. In case of both schemes the last response with data from ROS arrives at the DCM at almost the same time (17 ms). There is a major difference when comparing traffic in the opposite direction, from DCM to ROS. With static TCP, the DCM sends requests for all event fragments at once (see higher slope in the beginning in Figure 4.6b), whereas with traffic shaping the requests are distributed in time depending on the instantaneously available credits. This can have significant implications on the switch buffers depending on their architecture and configuration, as the traffic is not restricted on a single ROS to DCM flow. In our setup the ROS racks are connected to different blades of the core router (Figure 1.5) with independent virtual queues (VOQs). The instantaneous size of a single VOQ is stable over the time with lower maximum in case of static configuration (Figure 4.7). We can explain that by the fact that a particular ROS node responds with a burst of event data for a single request, if traffic shaping is used. Static TCP

3 The send window is usually given as the number of maximum-sized packets (MSS).



(a) ROS to DCM (b) DCM to ROS

Figure 4.6: IO graphs for the data collection process of a single event. The last ROS response (a) arrives after 16.766 ms for static *cwnd* of two packets and after 16.84 ms for traffic shaping (396 credits quota).

limits that on each ROS to DCM flow, but they are all transporting data simultaneously, so the load is spread evenly across VOQs.

The total number of bytes sent from DCM to ROS (Figure 4.6b) is 20 kB higher for the static configuration due to the increased number of TCP ACK packets. With *cwnd* of two packets an ACK is sent for every second packet coming from ROS. With the traffic shaping approach their rate is lower, especially if TCP segmentation offloads are enabled on the DCM side. This effect has minor impact on the overall performance, since the main data flow is in the other direction. Another observation is the fact that DCM keeps sending TCP ACKs for another 1.3 ms after having received the last DCM response. This is true only for static TCP. In the next paragraph we will see that this will be the cause for slightly increased mean latency of data collection seen by the application layer.

Performance of a single DCM The distribution of event collection latency is presented in Figure 4.8. Event rate is kept constant at 50 Hz, which keeps the 1 Gbps link from ToR switch to DCM close to saturation. It appears that static TCP mitigates incast as well as traffic shaping. There are no TCP timeouts in both cases. The mean latency is slightly higher for static configuration, which is contrary to the time it takes to collect a single event (Figure 4.6) as seen from a TCP dump. The explanation lies in the fact that the mean latency in Figure 4.8 is observed by the application layer DAQ software. The increase is therefore caused by an additional overhead of handling all ROS connections in parallel. It is also visible in the increased TCP ACK traffic in Figure 4.6b for the static TCP after having received the last data

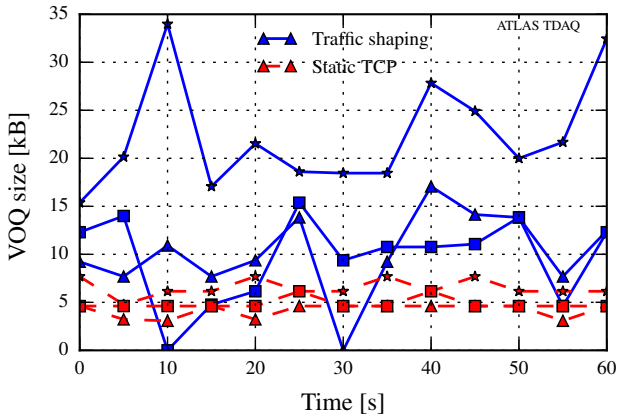
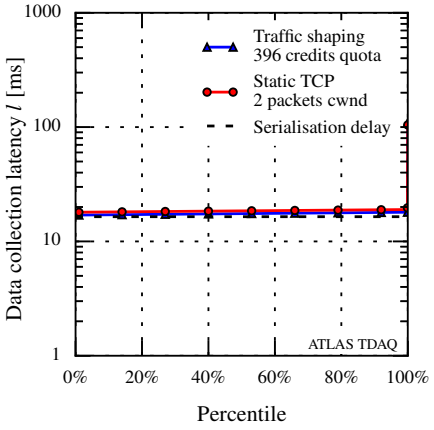


Figure 4.7: Maximum VOQ size of three different blades during event data collection. One blade, which connects more ROS racks, experiences higher load.



Algorithm	l_m [ms]	σ_l [ms]	G_{sust}
Traffic shaping	17.5	1.19	81 %
Static TCP	18.5	1.11	81 %

Figure 4.8: Event data collection latency with one data collector. Offered load is 83 %. Static TCP and traffic shaping eliminate TCP timeouts with the former having slightly higher latency.

response from ROS. The difference in mean latency is minor and does not significantly affect the entire system's performance.

4.5.2.2 *DIATCP*

Deadline and Incast Aware TCP (DIATCP) is a TCP variant proposed for datacenters [73]. From variants proposed in the literature we chose DIATCP for evaluation in DAQ networks. The decision was dictated by the availability of the code, the fact no changes are required to the networking hardware, and straightforward integration with the ATLAS DAQ software.

ALGORITHM DIATCP is deployed only at the aggregator (the DCM in our case). In DIATCP, the peers' (ROS) sending rates are controlled to avoid incast and application deadlines. For a detailed description re-

fer to [73]. In the DAQ world the events must be delivered reliably with minimum delay for the duration of the experiment. There is no concept of strict deadlines, so we only analyse DIATCP's incast-avoidance feature.

In contrast to the static *cwnd*, where the congestion window of the sender is used to control the packets injected into a network, DIATCP utilises the advertisement field in the TCP ACKs to allocate a specific window size to each ROS peer. It is the *awnd* (see Section 4.5.2) that is controlled by DIATCP and the condition defined by equation (4.5) takes the following form:

$$BDP \leq \sum_{i=1}^{N_R} \sum_{j=1}^{N_D} awnd_{ij} < BDP + Q_{bottleneck}\,, \tag{4.6}$$

where $awnd_{ij}$ is the advertised window to the ROS on the i-th ROS to j-th data collector flow. We have the knowledge of all incoming flows at the aggregator, so we can jointly regulate their advertised windows, as opposed to controlling *cwnd* (single ROS peer maintains a single connection to a particular DCM). The sum of all the advertised window sizes at the j-th DCM is DIATCP's global window as defined in [73]:

$$gwnd_j = \sum_{i=1}^{N_{ROS}} awnd_{ij}\,.$$

With the same global window at each DCM, $gwnd_j = gwnd$, the condition defined by equation (4.6) becomes

$$\begin{aligned}
BDP &\leq \sum_{i=1}^{N_R} \sum_{j=1}^{N_D} awnd_{ij} < BDP + Q_{bottleneck} \\
BDP &\leq \sum_{j=1}^{N_D} \sum_{i=1}^{N_R} awnd_{ij} < BDP + Q_{bottleneck} \\
BDP &\leq \sum_{j=1}^{N_D} gwnd_j < BDP + Q_{bottleneck} \\
BDP &\leq \sum_{j=1}^{N_D} gwnd < BDP + Q_{bottleneck} \\
BDP &\leq N_D\, gwnd < BDP + Q_{bottleneck}\,.
\end{aligned}$$

Hwang, Yoo, and Choi in [73] gave guidance on tuning *gwnd* depending on the network's RTT. We limit ourselves for *gwnd* fulfilling the requirement given by equation (4.6) and being comparable with *cwnd* and traffic shaping quota in terms of total in-flight bytes in the network.

One can see analogy between traffic shaping and DIATCP. The global window resembles the credits quota assigned to a DCM. The first one is calculated in packets, whereas the second is calculated in event fragment units. We can therefore treat DIATCP as traffic shaping implemented at the transport layer.

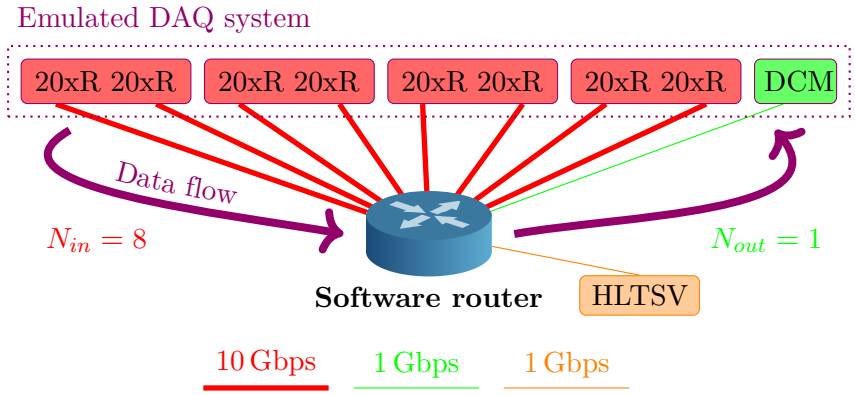


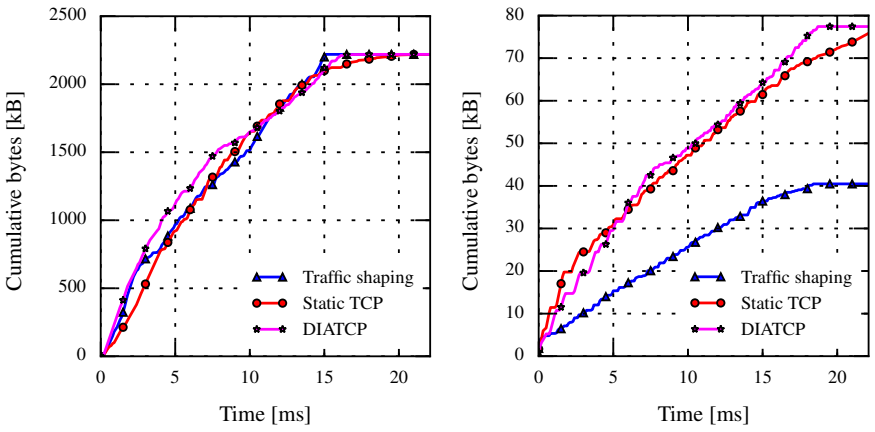
Figure 4.9: Setup for evaluation of DIATCP. 160 readout nodes are emulated on four nodes (eight virtual machines) with twelve 1.1 kB event data fragments on each.

EVALUATION SETUP DIATCP cannot be implemented in the form of a TCP congestion control module, as the static TCP variant. It requires changes to the TCP stack in the receiving path and, as a result, kernel recompilation. For this reason, it could not be deployed in the ATLAS DAQ system, which we used in the previous evaluations. Instead, we use an experimental testbed, see Figure 4.9. It consists of 160 ROS applications emulated on four server class PCs equipped with dual-port Intel 82599 10 Gbps Ethernet controllers. Each of them hosts two virtual machines. Thus, a single 10 Gbps link is shared by 20 readout applications. There is also a separate node running one data collector with 1 Gbps controller. Each ROS and DCM is connected via a server that is equipped with the same Intel adapters. It is configured as a software router using the Linux IP forwarding capabilities. With the help of different Linux queueing parameters, it is possible to emulate the behaviour of a real DAQ network. Each ROS application is configured with twelve event fragments of 1.1 kB. Using equation (A.7), the theoretical bandwidth is

$$\begin{aligned} B_{inout} &= \min\left(N_R n_R b, N_H n_H b\right) \\ &= \min\left(8 \cdot 1 \cdot 10\,\text{Gbps}, 1 \cdot 1 \cdot 1\,\text{Gbps}\right) \\ &= 1\,\text{Gbps}\,. \end{aligned}$$

which can be used in equation (A.12) to calculate a theoretical goodput of $G_{theory} = 943.4\,\text{Mbps}$. The offered event rate is kept at 40 Hz, which for this configuration is 72 % of the theoretical maximum. This value was chosen in order to focus on the performance of the collection process of a single event, with the offered load just below saturation to avoid potential limitations of the testbed.

The collection process of a single event The IO graphs in Figure 4.10 indicate comparable latencies for DIATCP and traffic shaping. Static



(a) ROS to DCM (b) DCM to ROS

Figure 4.10: IO graphs for the data collection process of a single event. The last ROS response (a) arrives after 18.94 ms in case of DIATCP with *gwnd* of 160 packets, 22.19 ms for static *cwnd* of two packets and after 18.48 ms for traffic shaping (421 credits quota).

TCP performs measurably worse with the last ROS response arriving 3 ms later.

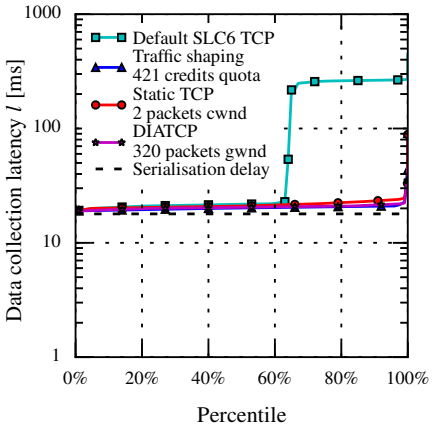
We believe this is because of the use of a server-based router instead of a real switch: the traffic is spread among different Ethernet cards and CPU sockets on the server. The BDP of this network (0.5 ms RTT) is already exceeded with a *cwnd* of two packets, but the complexity of this configuration has more implications on the results when serving all TCP flows in parallel compared to a standard router.

On the other hand, higher TCP traffic in the direction to ROS with DIATCP (Figure 4.10b) is caused by frequent TCP advertised window (*awnd*) updates.

Performance of a single DCM Figure 4.11 confirms that DIATCP can be also a valid candidate for DAQ. TCP timeouts are gone from the system in case of all three solutions. The mean latencies of DIATCP, static TCP and traffic shaping are comparable.

4.5.2.3 *DCTCP*

The well-known Data Center TCP (DCTCP) protocol is considered the state of the art TCP flavour for low-latency datacenter networks. It leverages the Explicit Congestion Notification (ECN) mechanism to keep the queues small while maintaining high throughput. The ECN support is required in the network switches though. Because of the lack of such devices, we rely on the conclusions provided by Alizadeh et al. in [6]. Although DCTCP provides comparable latencies as deep buffered switches under incast congestion and ensures good performance for typical traffic patterns in datacenters, it fails to avoid incast if there are so



Algorithm	l_m [ms]	σ_l [ms]	G_{sust}
Default SLC6	106.5	114.20	16.3 %
Traffic shaping	20.2	1.67	70.5 %
Static TCP	21.4	1.61	70.5 %
DIATCP	20.5	1.03	70.5 %

Figure 4.11: Event data collection latency with one data collector. Offered load is 83 %. Static TCP and traffic shaping eliminate TCP timeouts with the former having slightly higher latency.

many senders that the packets sent in the first RTT overflow the buffers. This situation is not uncommon for high-bandwidth low-latency DAQ networks like the one of the ATLAS DAQ/HLT system. For this reason, DCTCP can be regarded as a potential solution for DAQ networks, but within this limitation. Similar observation can be made for static TCP configuration, where the congestion window cannot be decreased below one packet per TCP flow. In case of DIATCP or traffic shaping, which we evaluated in the previous sections, this limitation is overcome by the receiver-side congestion control. The receivers can temporarily suspend the traffic on some or most of TCP flows depending on the quotas assigned to them. These quotas are assigned a priori, so the network is not overflown in the first RTT. This approach can fail first when just a single packet towards each single data collector in the entire farm is enough to overload the network. Furthermore, DCTCP remains dependent on the active queue management schemes offered by switch vendors.

4.5.3 *Link layer solutions*

As we described in Section 2.2.2, Ethernet standards provide now some mechanisms that can provide lossless operation, thus eliminating incast congestion, even across multiple hops. It has been shown in the literature that although they improve the performance and eliminate packet losses, highest throughput is difficult to achieve. We came to similar conclusions while evaluating the basic Ethernet flow control mechanism — IEEE 802.3x pause frame. If the packet buffer of a network node starts to fill up, it can send a pause frame to the neighbouring node to temporarily stop all traffic.

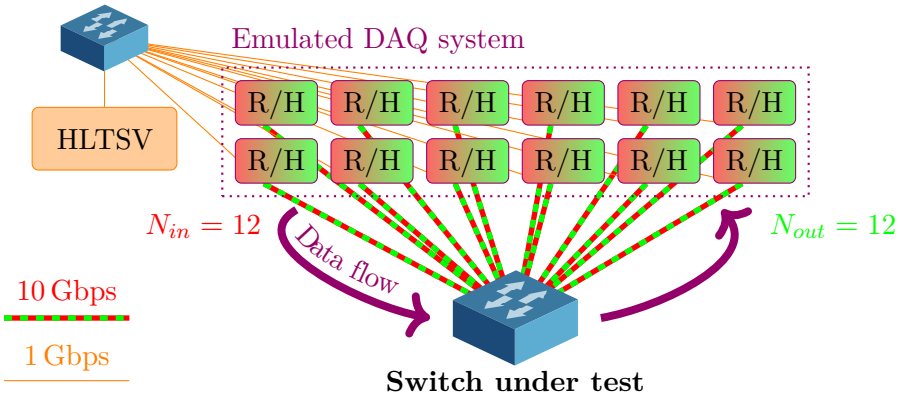


Figure 4.12: Setup for evaluation of the Ethernet pause frame mechanism (IEEE 802.3x). Twelve ROS nodes and twelve filtering racks are emulated on the same hosts. Each emulated rack contains twelve independent data collectors. A separate network is used for communication with the HLT supervisor (1GbE).

EVALUATION SETUP We use a simple setup with twelve nodes hosting both readout and data collector applications. This is a testbed setup, because pause frames were not supported on the real ATLAS hardware. Each of the nodes is connected with a single 10 Gbps link to the DAQ network that is built with a single ToR switch. Two different models, which we call ToR_A and ToR_B, are used and both of them support Ethernet pause frames. The setup is depicted in Figure 4.12. On a single node there is one readout application and twelve data collectors, which emulate an HLT rack. Each readout node provides 128 KiB of dummy data for a single event, so the total event size is 1.5 MiB, but only eleven fragments traverse the physical network links, because of sharing the nodes between ROSes and DCMs. Using equation (A.7), the theoretical bandwidth is

$$\begin{aligned} B_{inout} &= \min\left(N_R n_R b, N_H n_H b\right) \\ &= \min\left(12 \cdot 1 \cdot 10\,\text{Gbps}, 12 \cdot 1 \cdot 10\,\text{Gbps}\right) \\ &= 120\,\text{Gbps}\,, \end{aligned}$$

which can be used in equation (A.12) to calculate a theoretical goodput of $G_{theory} = 113.9\,\text{Gbps}$.

EVALUATION RESULTS The results of this evaluation are presented in Figure 4.13. The following configurations are tested:

1. ToR_A+TCP_cubic: ToR switch A with TCP cubic congestion control.
2. ToR_B+TCP_cubic: ToR switch B with TCP cubic congestion control.
3. ToR_A+Pause-CC: ToR switch A with pause frames. TCP congestion control disabled.

4. ToR_B+Pause-CC: ToR switch B with pause frames. TCP cubic congestion control disabled.

The ToR switches with the default TCP Cubic congestion control algorithm reach less than 20 % of the theoretical goodput. We do not provide results without congestion control in this case because the system became unstable[4].

TCP timeouts are clearly visible in the latency distribution in Figure 4.13b and 4.13c with substantial numbers of event collection latencies exceeding 200 ms. Enabling Ethernet pause frames (and disabling TCP congestion control again) significantly improves performance, but they still reach only 64 % and 85 %. We see timeouts are avoided and jitter is small, but the latencies are high. We will later look at the performance of an OVS-based switch using the same evaluation setup in Section 5.6.1.

This evaluation with just a single switch already proves some limitations of the IEEE 802.3x pause frame mechanism. Although packet losses are avoided and jitter is minimised, the full theoretical bandwidth remains unreachable. Furthermore, the results on different switches vary, which suggests that performance is dependent on the vendor implementations. These limitations can have even greater implications for larger DAQ configurations due to head-of-line blocking, which has been already demonstrated in other works as we indicated in Section 2.2.2.

4.5.4 *Comparison*

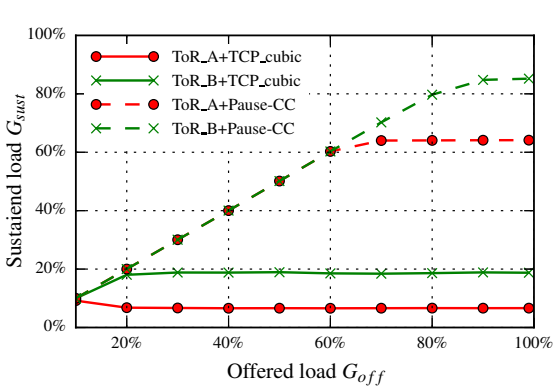
In this section we provide results for a different configuration and compare the evaluated solutions from three different layers:

1. The application layer — traffic shaping,
2. The transport layer — static TCP configuration,
3. The link layer — pause frames.

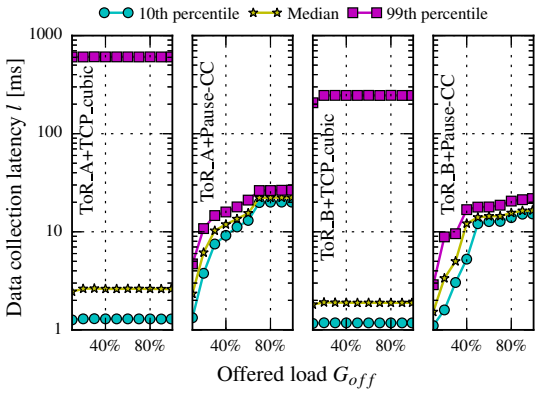
Because of the more complex implementation of DIATCP and the fact that the algorithm is analogous to traffic shaping, we did not include it in this comparison.

EVALUATION SETUP A larger, experimental testbed with twelve readout nodes and a rack of 30 data collectors is used to emulate a subset of a real DAQ system. Each ROS is connected with a single 10 Gbps link to the DAQ network that is built with two ToR switches. The setup is depicted in Figure 4.14. Each readout node provides 24 kB of dummy data for a single event, so the total event size is 288 kB. Here, in order to better emulate a real DAQ system, we configure each

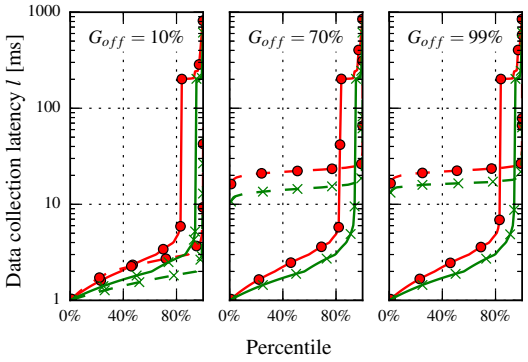
4 The data collection latency exceeded the default timeouts configured in the DCMs, so collection processes were being aborted.



(a)



(b)



(c)

Configuration	G_{off}	l_m [ms]	σ_l [ms]
ToR_A+TCP_cubic	10 %	46.3	114.5
	70 %	49.1	119.6
	99 %	48.5	119.5
ToR_A+Pause-CC	10 %	2.4	0.99
	70 %	22.1	2.2
	99 %	22.3	2.3
ToR_B+TCP_cubic	10 %	14.8	57.8
	70 %	16.0	61.6
	99 %	16.0	62.1
ToR_B+Pause-CC	10 %	1.6	0.7
	70 %	14.5	1.8
	99 %	16.6	1.7

Figure 4.13: Evaluation results of the Ethernet IEEE 802.3x pause frame mechanism with the simple test setup from Figure 4.12. Sustained load (a) and event data collection latency (b) as a function of the offered load, and the exact distributions of latency for three different cases (c).

data collector node with twelve processing units (see Section 3.5.1), which request independent events in parallel. Using equation (A.7), the theoretical bandwidth is

$$\begin{aligned} B_{inout} &= \min\left(N_R n_R b, N_H n_H b\right) \\ &= \min\left(12 \cdot 1 \cdot 10\,\text{Gbps}, 1 \cdot 1 \cdot 10\,\text{Gbps}\right) \\ &= 10\,\text{Gbps}\,. \end{aligned}$$

which can be used in equation (A.12) to calculate a theoretical goodput of $G_{theory} = 9.47\,\text{Gbps}$. Event rate remains unlimited, except for the load tests. Before we move on to the comparison, we will first tune both the traffic shaping algorithm and static TCP congestion window to find their optimum configurations.

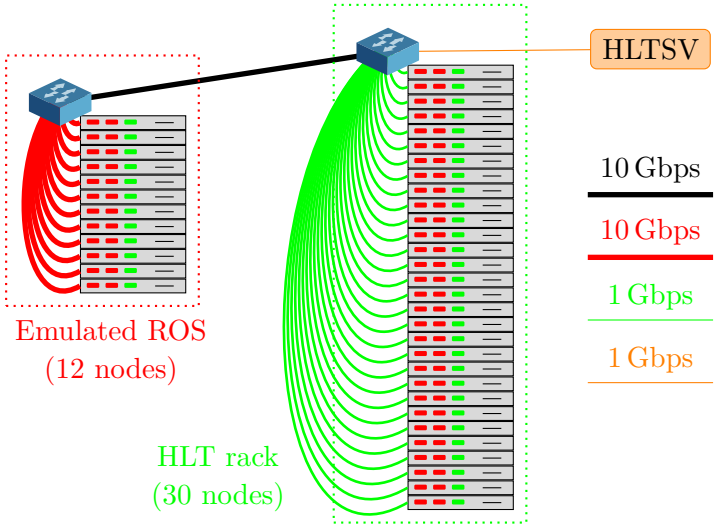
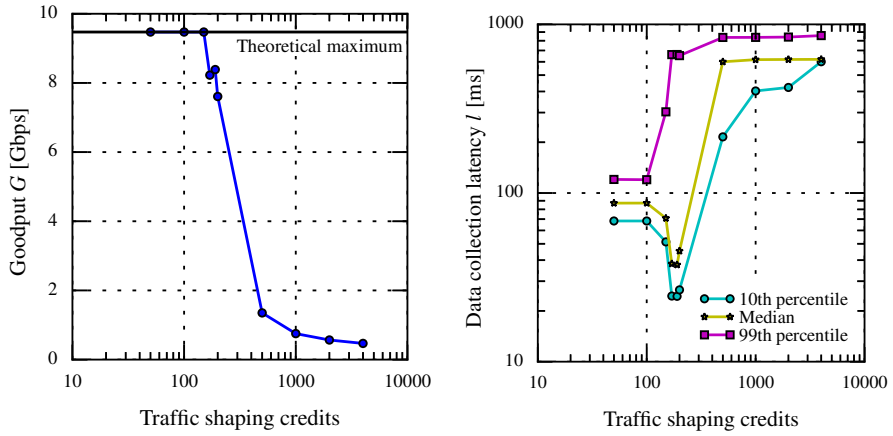


Figure 4.14: Test setup with 30 data collectors and up to 12 ROS nodes with 24 event data fragments on each.



(a) (b)

Figure 4.15: Goodput (a) and data collection time (b) when tuning traffic shaping credits for the setup depicted in Figure 4.14.

TRAFFIC SHAPING TUNING Figure 4.15 shows that the traffic shaping algorithm performs best below a 100 credits quota. Both goodput and data collection latency are optimum with the former achieving the theoretical maximum. Although the median latency decreases at first above 100 credits, the 99-th percentile approaches 1000 ms, indicating TCP timeouts and packet losses. The sustained load is thus lower, so those events that are not penalised by losses can be collected faster. For even larger quotas, every single data collection process suffers from at least one TCP timeout. We will use quota of 100 for the comparison with other technologies.

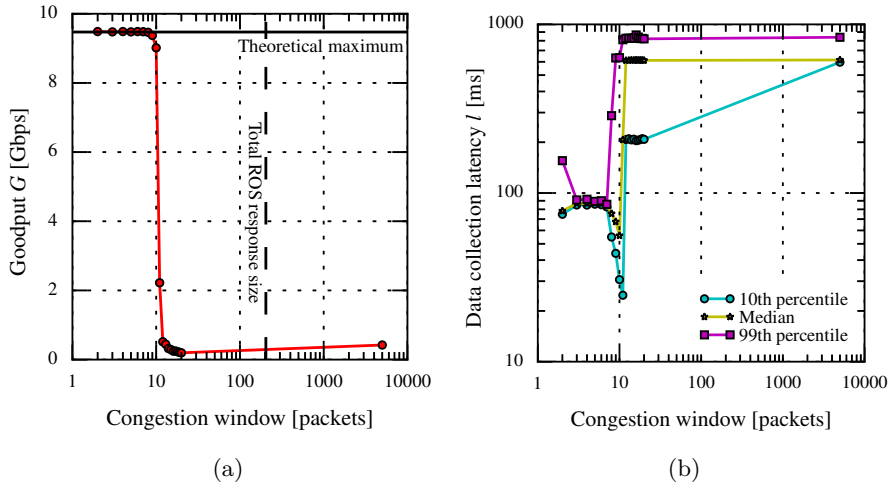


Figure 4.16: Goodput (a) and data collection time (b) when tuning TCP congestion window *cwnd* for the setup depicted in Figure 4.14.

STATIC TCP TUNING Analogous tuning is performed for the static congestion window *cwnd*. The results are presented in Figure 4.16. The maximum theoretical goodput is achieved with congestion windows less than ten, but the data collection latency is optimum for the values between three and seven. If *cwnd* is two, the network remains underutilised, operating below its BDP. For the values between eight and ten, similarly as with traffic shaping, packet drops start to occur, so the overall load is lower and some events can be actually collected with lower latency. For even larger congestion windows, all events are collected with significantly larger latencies because of numerous TCP timeouts. We will continue to use $cwnd = 7$ for the following comparison.

RESULTS Detailed comparison of traffic shaping, static TCP configuration, and pause frames is presented in Figure 4.17. As can be seen from Figure 4.17a, all three approaches sustain the requested load in the entire range, up to 100 %. Different from the performance with the previous testbed, see Section 4.5.3, Ethernet pause frame mechanism is also capable of providing full performance. In our previous pause frame tests, it was not. The difference now is that the current evaluation setup is less demanding than it was in Figure 4.12, where all-to-all incast congestion with twelve 10 Gbps ports was invoked. In the current setup, twelve-to-one overcommitment at 10 Gbps occurs at the switch connecting ROS nodes and additional congestion can appear at the switch connecting data collectors with 1 Gbps links to the ROS switch over the 10 Gbps uplink.

But signs of worse capabilities of the pause frame mechanism are visible in the latency characteristics (Figure 4.17b and 4.17c.) Data collection latency and jitter are substantially higher than in case of traffic shaping and static TCP. Thanks to the large number of independent

processing units in data collectors, full goodput can be achieved despite the higher latency.

Traffic shaping and static TCP achieve the same performance in terms of goodput and latency. Interestingly, data collection time does not increase with the load as expected (see Section 3.3). On the contrary, it decreases slightly. The reason is that both algorithms are tuned for the 1 Gbps links, so the bandwidth in the 10 Gbps uplink to the ROS nodes can be shared fairly across multiple data collectors. Furthermore, there are twelve independent processing units on each collector. For lower loads it can happen that the HLTSV assigns most of the events to the processing units on the same collector. This results in higher data collection latency because those units share the same 1 Gbps link to the network. For higher loads, there is a higher probability that the load is better spread across available collectors.

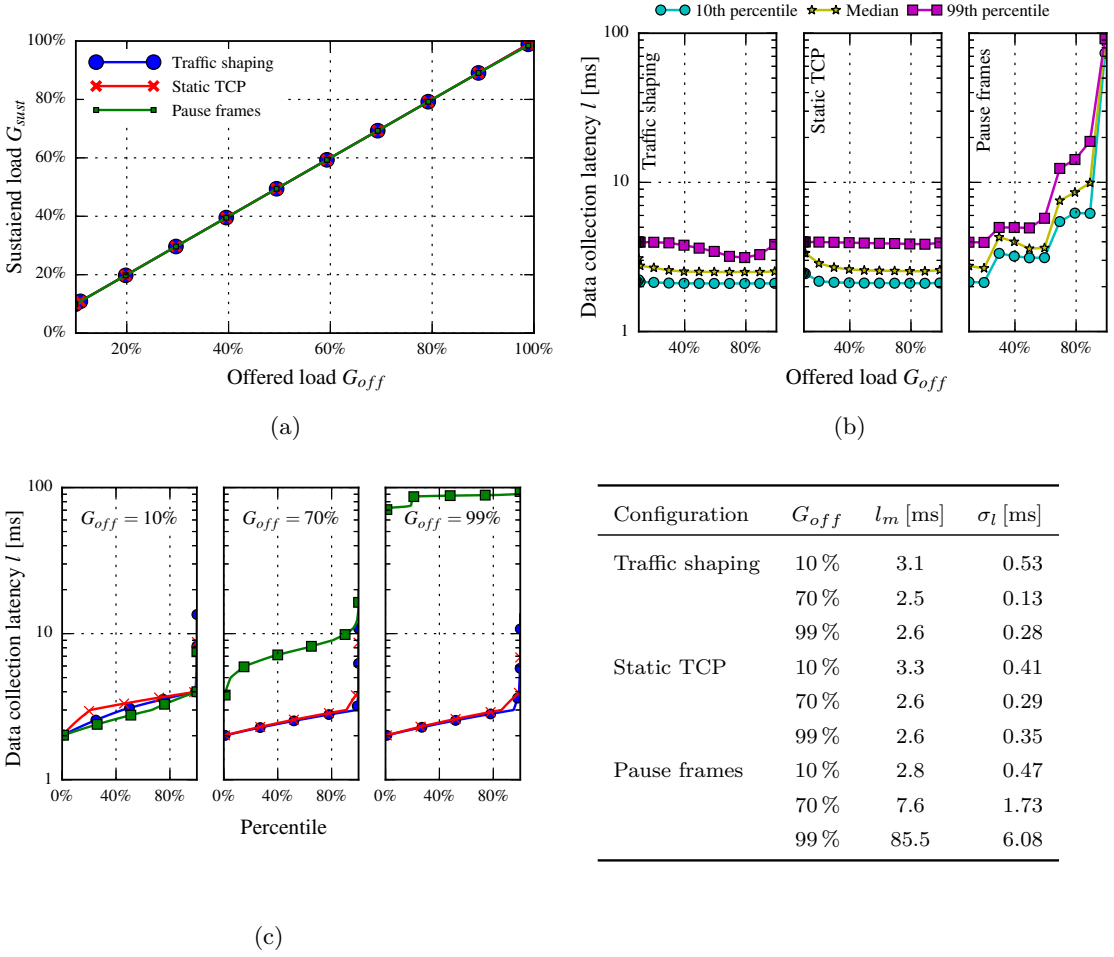
The results confirm again that all three approaches have means to improve the performance under incast congestion. Particularly, traffic shaping and static TCP offer full goodput and low latency. The pause frame mechanism, although achieving theoretical goodput, experiences higher latency and jitter.

4.5.5 *Summary*

Even a simple static configuration of TCP congestion control, which does not require any additional programming overhead, can already effectively improve the performance under incast congestion. Advanced TCP incast avoidance algorithms proposed for datacenter can be also successfully applied in DAQ networks, which we demonstrated on the example of DIATCP. These, however, require at least Linux kernel recompilation and often modifications to the networking hardware. If used, they can save the programming effort of implementing application layer solutions. Software traffic shaping obtains results that match or exceed the alternative proposals, but it is specific to the ATLAS DAQ and not transparently transferable to other environments. On the other hand, the pause frame mechanism guarantees lossless operation even in complex environments, but at the cost of higher latency and, in some configurations, lower sustained load.

4.6 CONCLUSION

Using the example of the TCP/IP-based ATLAS DAQ network we showed the consequences of many-to-one congestion. The throughput of a system can be seriously limited, even to a few percent of the available bandwidth, if the network does not provide enough capacity to accommodate many-to-one data bursts. It is therefore important to understand the problem and provide solutions for many-to-one communication patterns.



Configuration	G_{off}	l_m [ms]	σ_l [ms]
Traffic shaping	10 %	3.1	0.53
	70 %	2.5	0.13
	99 %	2.6	0.28
Static TCP	10 %	3.3	0.41
	70 %	2.6	0.29
	99 %	2.6	0.35
Pause frames	10 %	2.8	0.47
	70 %	7.6	1.73
	99 %	85.5	6.08

Figure 4.17: Comparison of application-layer traffic shaping, static TCP congestion window, and Ethernet IEEE 802.3x pause frame mechanism in the configuration from Figure 4.14. Sustained load (a) and event data collection latency (b) in function of the offered load, and the exact distributions of latency for three different cases (c).

Our original contribution is the identification of the analogous problem in datacenter networks, where it is referred to as TCP incast pathology. On one hand, solutions provided for one can be also tested in the other. But the key characteristics of DAQ networks allow for some simplifications that make it easier to pursue a different strategy. This is the case with software switches, which will be discussed in Chapter 5.

Nevertheless, many proposals for incast congestion in datacenters can be effective also in DAQ. Most of them focus on limiting the packet injection rate into the network. We evaluated some example solutions, at the different protocol layers, and proved that they can be applied in data acquisition, for some configurations at least. We also showed that a similar effect can be achieved with a simple TCP static congestion window. All these approaches, including the ATLAS traffic shaping algorithm, are most effective in avoiding incast congestion in the last hop of a network. Congestion control becomes more complex in the network

core, where flows towards hundreds of DCMs overlap. But even with an effective algorithm, more event data have to be eventually buffered at ROS, which prevents from using simple devices in the readout system.

5

EXTENDING BUFFERS WITH SOFTWARE SWITCHES

In this chapter the core contribution of this thesis is discussed. We evaluate whether incast congestion in DAQ networks can be avoided by using software switches with large packet buffers in the main memory of a server-class computer. We show that potential bottlenecks, that are normally identified for this approach, have less significance in DAQ networks. Next, we present the design and evaluation of a prototype software switch that can be configured to guarantee lossless operation under heavy incast congestion. In the end, we show that software switching can be also used in production systems already and provide good performance. We extend a popular virtual switch, Open vSwitch, with a mechanism allowing programmable buffering of incast-sensitive flows in large, dedicated queues. This chapter contains results published in [88, 89].

5.1 INTRODUCTION

In the previous chapter we showed general avenues to approach the incast congestion problem. Many solutions exist that can significantly delay the onset of incast or even completely avoid it, but at the cost of lower saturation load, like in case of the pause frame mechanism. Most of the proposals focus on controlling the amount of packets that are simultaneously sent over a network. There are two main drawbacks of this approach. Firstly, complexity increases with the size of the network, link speeds, and the number of parallel data collection processes. Secondly, data that are deferred from transmission over a network have to be buffered on the sending nodes — the readout system in DAQ. This implies higher complexity of the readout nodes and prevents the use of simple push architectures.

An alternative way to solve the problem is to use large-enough packet buffers, see Section 4.4.3. Unfortunately, this approach is claimed to be costly. Network switches with large memories are rare and expensive, so even more expensive telecom-class routers are sometimes the only choice. In our work we intend to fill this gap by proposing software switching that can provide cheaper and extensible buffers in the DRAM memory of COTS servers. Buffering capabilities would be then limited only by the amount of DRAM memory that can fit into a single server. Nowadays, this amount is counted in terabytes. Furthermore, the use

of software for control makes it easy to adapt the switch to a target application, so it can be tuned in order to improve the performance.

The performance of software switches has traditionally stood in the way of following this strategy. But the situation has changed, which we showed when presenting the evolution of software switches and their performance as well as the key advances in modern computer architectures over the recent years in Section 2.3. It has now become possible to process hundreds of gigabits of packets on commodity servers, which removes the barrier of performance of software switches for incast avoidance in high-speed data acquisition networks.

This chapter is structured as follows. Section 5.2 gives an overview of packet processing in software with the focus on theoretical performance and potential limitations. This discussion is put into context of data acquisition in Section 5.3, where we also describe our evaluation setup. We present our design and evaluation of the lossless software switch for data acquisition in Section 5.4. Then, in Section 5.5, we analyse whether the same goal can be achieved by adapting the popular virtual switch — Open vSwitch (OVS). In the end, we compare the performance with traditional switches and discuss some other aspects in Section 5.6. We conclude this chapter in Section 5.7.

5.2 SOFTWARE PACKET PROCESSING

As we demonstrated in Section 2.3, history is coming full circle. Switches and routers are being implemented in software running on commodity PCs again and can provide good enough performance. This trend has risen from the popularity of virtualised environments used in datacenters, in which software switches provide connectivity between virtual machines running on the same host and to the outside world as well [51]. But it is not only switching and routing that have become popular network functionalities being implemented in software. Various network appliances, like proxies, firewalls, or load balancers, are now often replaced by their software-based counterparts. This whole new trend is called Network Functions Virtualisation (NFV) and means implementing a generalised component of a network in software [66].

The main advantage of packet processing in software is high flexibility. Modules can be combined according to user's needs without paying for unnecessary features. Furthermore, free and open source projects can be used and optimised for a given application. It is even more compelling for data acquisition. Not only the flexibility of the design in software could improve the performance, but, above all, buffering capabilities could become highly scalable, when using the DRAM memory of a commodity server instead of expensive high-end switches and routers.

5.2.1 *Theoretical performance*

Thanks to the architecture of modern computers and progress in the speeds of buses, it is now possible to consider software switches as a replacement for typical hardware devices. Discussions on key technological advances are available in the references provided in Section 2.3.

A high-level diagram of one of the modern server platforms that we will later use to build a prototype of a software switch is presented in Figure 5.1. It consists of two Intel Xeon EP-2600 processors with integrated memory controller (IMC) and integrated I/O (IIO) controller for PCIe and DMI (Direct Media Interface) [78]. The CPUs are connected over two QPI links. Each CPU offers 40 PCIe 3.0 lanes, which can be used to connect multiple NICs providing Ethernet ports (up to 100GbE interfaces) to build a switch. Each lane offers 8.0 GT/s (gigatransfers per second[1]). The total bandwidth that is offered over the PCIe interfaces is therefore

$$B_{PCIe} = 2 \cdot 40 \cdot 8.0\,\text{GT/s} \cdot 128/130\,\text{bit/symbol} = 630.2\,\text{Gbps}$$

independently in each direction (128/130 encoding). NICs use PCIe and Direct Memory Access (DMA) to move the received packets and pull those be transmitted to/from the main memory. The DMA engine allows those operations to be performed without the involvement of the CPU. The maximum data rate of a single channel to the memory is set by the limits of the Double Data Rate type three (DDR3) DRAM. This platform supports modules up to DDR3-1600, which offers 1600 MT/s. Thus, the total theoretical bandwidth to or from the memory for a platform with two CPUs and four channels each (a single channel is 64 bits wide) is

$$B_{mem} = 2 \cdot 4 \cdot 64\,\text{bit/symbol} \cdot 1600.0\,\text{MT/s} = 819.2\,\text{Gbps}$$

The last bandwidth-limiting bus is the QPI interconnecting CPUs. If a NIC is connected with PCIe lanes of one CPU and has to access packets residing in the memory of the other CPU, QPI bus is used. It offers 8.0 GT/s data rate, which for the bus width of 16 data bits and two QPI links results in the total bandwidth (independently in each direction) of

$$B_{QPI} = 2 \cdot 16\,\text{bit/symbol} \cdot 8.0\,\text{GT/s} = 256\,\text{Gbps}\,.$$

Platform Controller Hub (PCH) is a next generation chipset, connected over DMI with the CPU. It provides centralised platform capabilities, like the main I/O interfaces, display connectivity, storage features or power management [85].

1 Transfers per second is the number of operations transferring data per second in some channel.

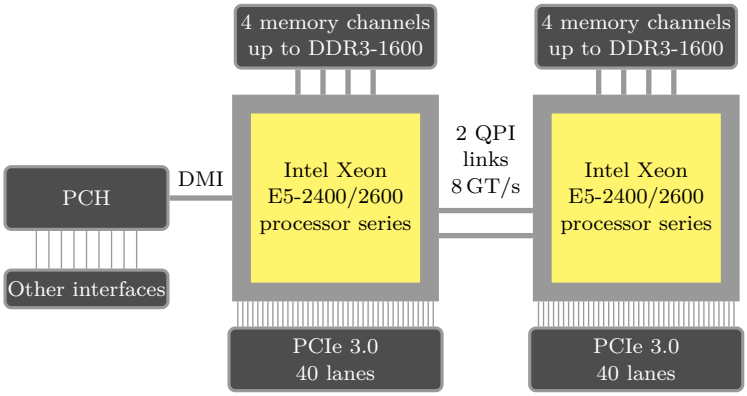


Figure 5.1: Block diagram of one of Intel's platforms (code name Romley) for the Intel E5-2600 processor series.

The data rates provided by the buses of just a single presented platform suffice, bandwidth-wise, the requirements of the ATLAS experiment in Run 2 (approximately 400 Gbps, see Section 3.5.1). This is a good premise to consider a solution based on software switches for future data acquisition networks. Even 384 GB of DRAM memory can be installed for each CPU [83], most of which can serve as a packet buffer. Furthermore, the capabilities of COTS server platforms continuously evolve and, thus, it is highly probable that they will keep providing enough I/O performance for new or subsequent generations of the experiments. For example, with DDR4 DRAMs supported in the last two generations of Intel Xeon processors (code names Haswell and Broadwell) the memory bandwidth increases up to 544 Gbps for a single CPU (with maximum memory size of 1.5 TB), so an aggregate for a dual-CPU platform exceeds 1 Tbps [83].

5.2.2 *Potential bottlenecks*

Although the data rates offered by the components of modern platforms seem enough to build an Ethernet switch with a bandwidth of several hundreds of gigabits, it is important to understand potential bottlenecks in this application. In case of Ethernet, one must take into account that data is chunked into Ethernet frames, each of which can carry typically between 46 B to 1500 B of payload (*basic frames*) [76][2]. This means that data flow into and from the memory not as a continuous byte-stream, but as a stream of packets representing separate transactions (if there are no optimisations like processing packets in bunches) with certain latency. Each of the packets has to be also processed by the software switch independently spending a certain amount of CPU cycles. Thus, in order to take advantage of the available bandwidths

2 An exception are the so called *Jumbo frames*, which are longer (typically 9 kB) than the maximum frame length set by the Ethernet standard [150]. In this work we focus on the lengths allowed by the standard.

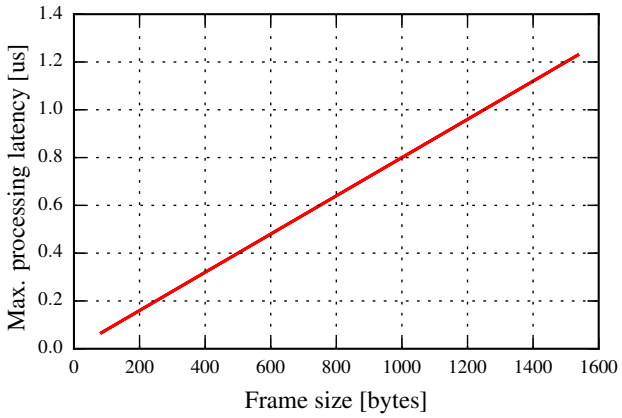


Figure 5.2: Maximum packet processing latency to saturate a 10 Gbps link as a function of the Ethernet frame size (assuming serial processing).

from the previous section, per-packet latency has to be considered. This is why the performance of software switches is normally evaluated in terms of *packets per second* (pps) and not in *gigabits per second*.

Figure 5.2 is a simple illustration of this fact. It shows what is the maximum per packet processing latency, so that it is still possible to saturate a single 10 Gbps link. A serial, packet-by-packet processing is assumed. An obvious conclusion is that it is easier to saturate network links with large packets. Large packets are more frequent in data acquisition than small ones, as we explained in Chapter 3. Typical bottlenecks of software packet processing are therefore less critical and there is more headroom to build a software switch using the available I/O bandwidth of modern platforms. Furthermore, the capabilities of modern server platforms to perform packet processing have been already proved and thoroughly analysed in the literature (see Section 2.3). For these reasons, as we noted already in Section 3.5, we conduct evaluations in this thesis, including software switching, using large packets. We focus our attention on key aspects from the viewpoint of data acquisition, i. e. the application of software switches to avoid incast congestion in large DAQ networks. Nevertheless, we still discuss some of the aspects that could potentially limit the performance of software switches in this context. Our analysis includes the dependencies on the number of the available CPU cores and their frequency in order to reduce power consumption, see Section 5.6.2. We also check the influence of the number of receive queues used by the NICs in Section 5.5.4. Since we propose using all available PCIe slots of a platform across all CPUs, we check whether there are any bottlenecks when using the QPI bus between the processors in Section 5.4.2.1.

In order to avoid the well-known bottlenecks, we follow the recommendations provided by a given packet processing framework (in our case DPDK [80], see the next section for more details). This especially concerns NUMA-aware (Non-Uniform Memory Access) object alloca-

tion in memory and use of huge page tables. The packet send and receive operations are performed by multiple CPU cores on different NUMA nodes. The affinity of the CPU cores is set, where possible, to match the NUMA-node of the corresponding network interface. The kernel is also configured to isolate those CPU cores used by the software switch (the *isolcpus* flag) and further reduce the impact of the kernel with the *nohz_full* and *rcu_nocb_poll* flags [92]. The two latter parameters are used to reduce the periodic timer interrupts on the CPUs and improve their real-time response. Also, we disable the Hyper-Threading technology[3] in order to avoid any dependencies on sharing any physical resources by the sibling cores. The size of the receive and transmit queues on the NICs is set to their maximum allowed values.

5.2.3 *The DPDK packet processing framework*

A boost to the performance of software switches, routers, and other network applications has been given by specialised packet processing frameworks, which we summarised in Section 2.3. The Data Plane Development Kit (DPDK) [80], Netmap [143], PF_RING [138], or Snabb Switch [151] are popular frameworks for software-based packet processing on standard x86 servers. The common goal of these frameworks is to optimise performance, which is achieved by providing their own drivers and libraries, not relying on the standard network stack of the kernel. As shown in [52] on the example of DPDK, those modern frameworks show significant performance improvements when compared to the default Linux kernel implementations.

In this work, we use the DPDK framework to build a dedicated software switch for data acquisition. The decision to use DPDK is motivated by its exhaustive documentation, support for NICs of different vendors, large community, and a broad set of examples. The sample forwarding application provided by the framework (L3FWD), in particular, delivers full forwarding performance without any difficulty and has become a basis for us to design our own switching application.

DPDK is a set of libraries and optimised NIC drivers that replace the default Linux network stack. Those drivers are not placed in kernel, but in user-space and allow direct access to the NICs and the so-called *zero-copy* packet processing. The libraries provide, among others, NUMA-aware memory allocation, lockless first-in-first-out (FIFO) queues, means for manipulation of packet buffers carrying network data, timer facilities, and a network library with a collection of convenience macros and structures for protocol processing (IP, TCP, UDP and others). They build a complete and simple programming framework for constructing extremely high-performance packet processing appli-

3 Intel's Hyper-Threading technology allows a single CPU core to handle two separate sets of instructions simultaneously. To the operating system, it makes the system appear as if the number of cores is doubled [147].

cations. Originally, DPDK drivers have operated in polling mode in order to eliminate the overhead of interrupts. Support for the interrupt mode was added in DPDK release 2.1.

Zhou et al. in [179] named three critical aspects of the DPDK that are particularly relevant to the performance of a DPDK-based packet forwarding application:

1. Memory management: NUMA-aware object allocation in memory using processor huge page table support in order to reduce TLB (Translation Lookaside Buffer[4]) misses. Objects are also aligned properly, so that access to them is spread across all memory channels (see Figure 5.1), which is required to take advantage of the available bandwidth to memory.

2. Polling mode: Polling mode speeds up processing, because the overhead of interrupt handling is eliminated. The drawback is the increased CPU consumption, which can interfere with other tasks on the same machine. In our case, we devote CPU cores entirely for packet processing, so this is not an issue.

3. Batching: Packets in DPDK are delivered in large batches for efficiency. With this technique the overhead associated with accessing NICs (e. g. locks, system calls) occurs once for several packets and not for each single one [16].

More details on DPDK can be found on the project's website [80] and in the literature [16, 110, 179].

5.3 THE CONTEXT OF DATA ACQUISITION

Some aspects of data acquisition simplify the adoption of packet processing in software, as already noted in the previous section. The typical bottlenecks of software switches, in particular, are not critical in the context of DAQ.

In Section 3.3 we discussed the typical requirements that are put on data acquisition networks. We continued this discussion in Section 3.4 focusing on throughput versus latency optimisation. We explained that our attention is not directed towards minimisation of the zero-load network latency. In DAQ, it is important to optimise the network at high loads, where throughput and latency are correlated. For this reason, while evaluating software switching, we focus on maximising the saturation goodput and minimising the jitter of data collection time. In other applications, like in HPC or in datacenters, zero-load latency can be of high importance, so the overheads of software packet processing could become relevant.

4 TLB speeds up the page table lookup in the virtual memory system of a computer operating system by storing the most recent page lookup values in a page table cache [122].

We discussed in more detail the key analogies and differences between the typical traffic patterns found in data acquisition and datacenter networks in Section 4.3. Relatively small network size, the dominance of large packets, static and well-known traffic pattern are other aspects that make the typical bottlenecks of software switches less critical in DAQ.

Furthermore, the fact that the traffic pattern is known in advance and the configuration of the system usually does not change for long periods of time brings advantages while considering software switching. Since queueing is implemented purely in software, it can be optimised to the given configuration in advance, before the start of an experiment. Also, the required amount of buffering can be estimated before the start of the run, so the packet buffers can be easily extended with additional DRAM memory modules, if required.

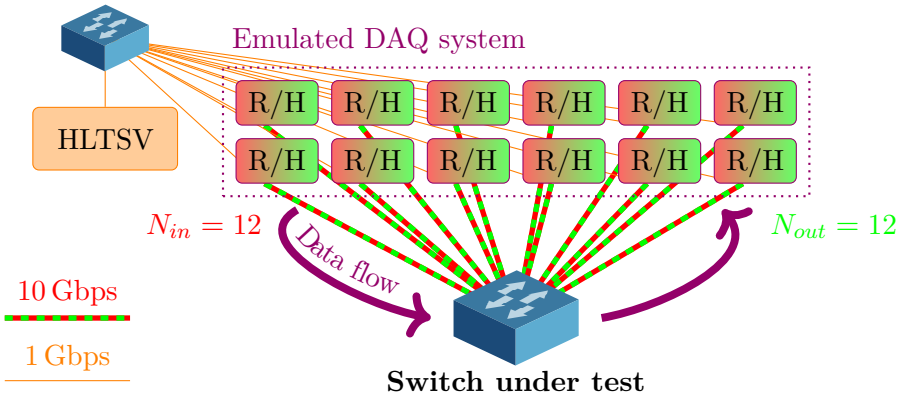
In Chapter 6 we will also show how those aspects help in building and managing a large topology of interconnected software switches to maximise the throughput, if the configuration and the traffic pattern are known in advance.

5.3.1 *Evaluation setup*

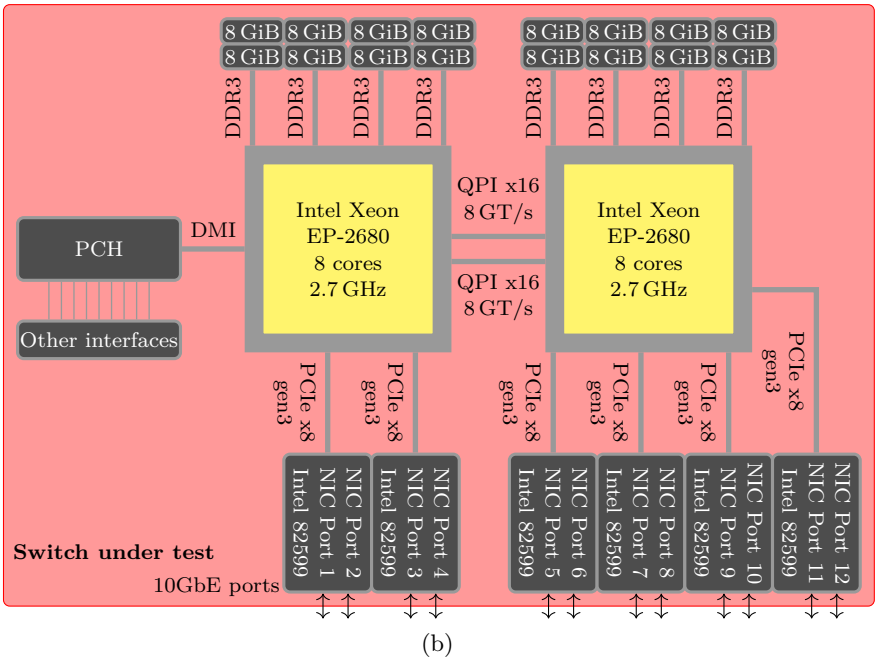
All evaluations presented in this chapter are performed using the same emulated DAQ configuration, unless otherwise stated. The setup, depicted in Figure 5.3, is the same as the one used to evaluate the pause frame mechanism in Section 4.5.3 (see Figure 4.12), but the switch under test is now a COTS server with multiple Ethernet ports running a DPDK-based switching application.

We use one of the server boards based on the Intel C602 chipset (code name Romley), S2600GZ [84], with two Intel Xeon EP-2680 eight-core CPUs, which correspond to two NUMA nodes. We equip the board with six dual-port 10GbE cards (Intel 82599 Ethernet controller [79]) providing a total bandwidth of 120 Gbps. Each of them is connected over eight PCIe lanes (gen2) directly to the CPUs. There is enough bandwidth on the PCIe buses to provide bidirectional data transfer at 10 Gbps between each port and the memory. The first four ports are connected to the first NUMA node, whereas the latter eight to the second NUMA node. This four/eight split results from the physical routing of PCIe lanes on the board. There is also a total of 128 GiB DDR3 memory installed (64 GiB per CPU socket). The operating system is 64-bit Fedora 20, kernel version 3.18.7-100. We configure the system to avoid the well-known bottlenecks, as described in Section 5.2.2. The architecture details and theoretical performance of the Intel Romley platform were described in Section 5.2.1.

Since the emulated DAQ configuration has not changed, the theoretical performance is also the same. With twelve readout nodes providing



(a)



(b)

Figure 5.3: Test setup (a) for evaluation of software switching. The switch under test (b) is a COTS server with twelve 10GbE ports running a switching application (DPDK).

128 KiB event fragments, the total event size is 1.5 MiB and theoretical goodput equals to 113.9 Gbps.

5.4 A DEDICATED SOFTWARE SWITCH FOR DAQ NETWORKS

In order to estimate the performance of software switching in the context of data acquisition we first design, implement and evaluate a basic switching application in the DPDK framework. In our designs, we consider different buffering strategies. Traditionally, in typical switches and routers, they are implemented in hardware and their adjustment is impractical after the product is released. With software switching, implementation of those strategies is often easier than in hardware and

solely requires changes to the software and re-compilation of the application. Therefore, we can evaluate different concepts in buffer designs and reach back to some ideas from late 1980s analysed by Nagle [115] or Tamir and Frazier [156] and also from 1997 by Morris [111]. We will analyse two classical designs and propose a new one that is dedicated for data acquisition.

5.4.1 *Design*

The main rationale behind our design of a software switch for data acquisition networks is to eliminate packet drops caused by incast while maintaining high throughput. Because of the large event size, less effort can be put into minimising processing time of a single packet. The DPDK sample application, L3FWD, which is a layer 3 switch[5], forms a basis for us to design our own switching application. We consider three different queueing strategies, which are depicted in Figure 5.4. *Lcore* refers to a logical execution unit of a processor (also called a hardware thread) [80]. In our case, the Hyper-Threading technology is disabled (see Section 5.2.2), so there is always a single hardware thread associated with a single physical CPU core. There is always a single *master lcore*, which runs a default thread of our switching application and is responsible for initialisation of various objects. Other lcores are responsible for polling packet descriptors from NIC receive queues, packet processing, and/or placing descriptors in transmit queues in order to send those packets back into the network on an appropriate port.

In typical software switches, as Open vSwitch or DPDK example applications, packets are dropped when the available transmit queues fill up. We take a different approach as our goal is to reach lossless operation. In order to achieve this we use the backpressure mechanism [42]. Whenever an lcore tries to place a packet descriptor in a full transmit queue, it will keep retrying until it succeeds. In this way, queues preceding the transmit ones (receive queues or any software queues available in the software switch) have to buffer any excess packets. This backpressure moves eventually to the receive queues of the NICs. Thus, this is the single point where packets can be dropped, if not enough buffers are provided by all queueing stages. Without the backpressure, we experienced occasional packet drops on the transmit side, when the transmit queues filled up because of some fluctuations on the switch itself or even on the end-nodes (if configured to send pause frames). In all our designs we set the size of the receive and transmit queues to the maximum allowed by the NICs.

5 Layer 3 switches select the output port based on the destination IP address of an Ethernet frame.

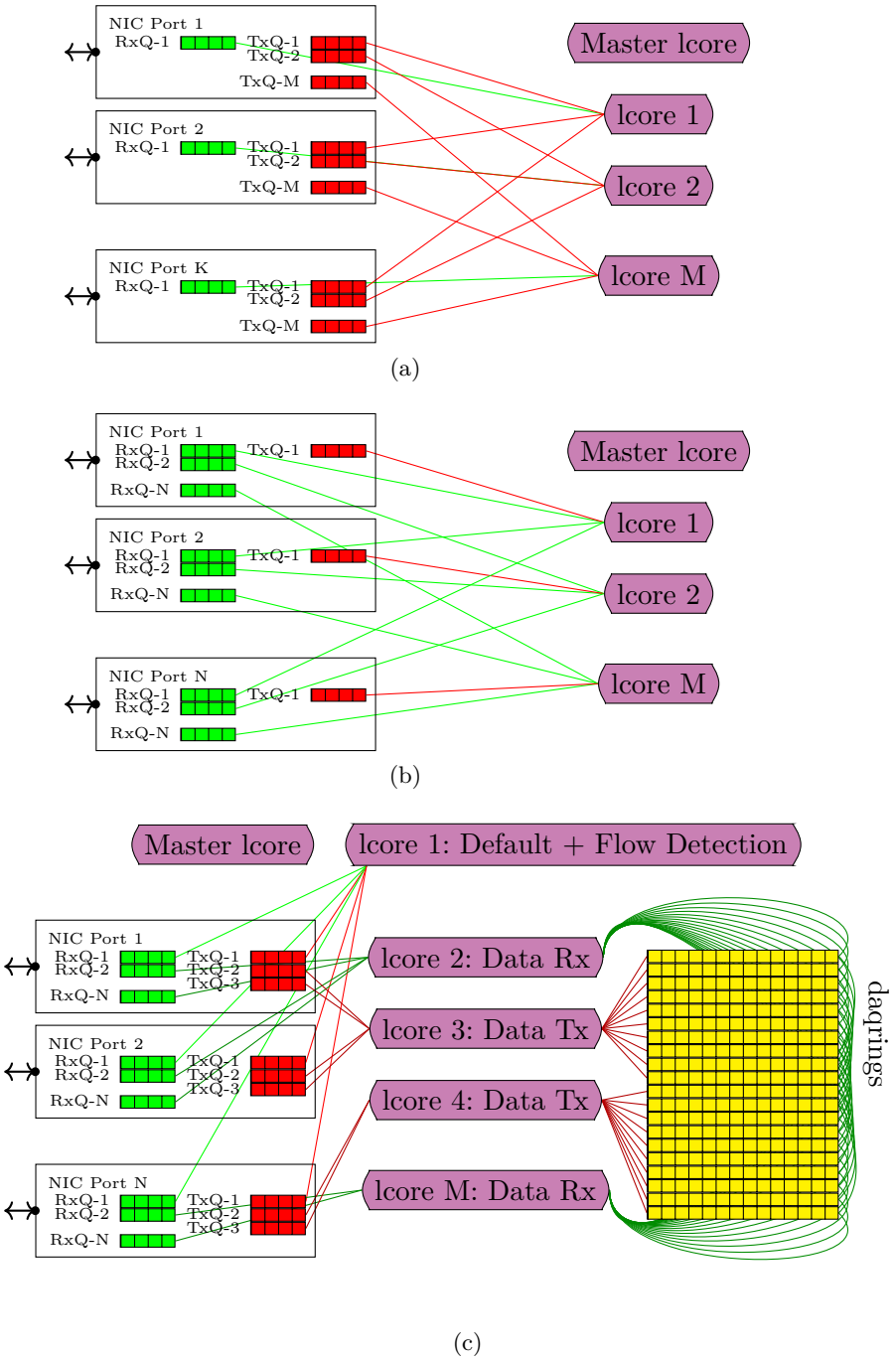


Figure 5.4: Three differing queueing strategies for evaluation of the DPDK-based switching application. Output queueing (OQ) using hardware queues of a NIC in (a), virtual output queueing (VOQ) using hardware queues of a NIC in (b), and DAQ-dedicated queueing using software queues (DPDK rings — *daqrings*) in (c).

5.4.1.1 *Output queueing in hardware (OQ)*

In this queueing regime (Figure 5.4a) only a single receive queue (RxQ) is used on every NIC. These queues are distributed equally across the available lcores, which poll the descriptors of incoming packets. Every lcore is also associated with a dedicated transmit queue (TxQ) on every NIC. After consulting the forwarding table (based on destination IP addresses) the lcores enqueue the descriptors to appropriate transmit queue. This strategy resembles the output-queued switches [99]. Buffering is realised with the receive and transmit queues only. If the NICs do not provide long enough queues, packets can be dropped.

5.4.1.2 *Virtual output queueing in hardware (VOQ)*

Next strategy follows the well-known Virtual Output Queueing (VOQ) scheme, in which each input maintains a separate queue for each output [99]. Therefore, in contrast to the previous strategy, only a single transmit queue is maintained on every NIC and all transmit queues are distributed equally across the available lcores (Figure 5.4b). On the receive side, there is as many queues on a single NIC as there are NICs in total. The internal hardware filters of the NICs are programmed in such a way that the incoming packets are enqueued to those queues based on the destination IP address. For example, the Intel 82599 controller allows the configuration of up to 8194 so-called perfect match filters [79]. This cannot be considered a bottleneck as we can use appropriate masking to share the filters across common flows. If it is still not enough, additional classification in software is required. A similar approach was used by Cerrato, Annarumma, and Risso in [34] or by Tanyingyong, Hidell, and Sjödin in [159, 160]. The ordinal number of the particular receive queue gives directly the appropriate output port, so it is not required for the lcores to consult the software forwarding table. Receive queues associated with a specific output port are bound to this lcore that is assigned to the transmit queue of this specific port. The same as with the previous scheme, buffering is provided only by the receive and transmit queues of the NICs.

5.4.1.3 *Dedicated queueing in software (daqrings)*

We designed this scheme (Figure 5.4c) as a dedicated mechanism for data acquisition. Its core lies in the idea of implementing dedicated buffers to each DCM in the system. It closely resembles the strategy analysed by Nagle in [115] in 1987. He showed that even with infinite buffers in switches a network can drop packets due to increasing delays[6]. In order to avoid this problem the author introduced the term *fairness* — each source host should obtain an equal fraction of the resources at each switch. This could be achieved by maintaining a separate queue

6 The problem of the increasing network delay with larger buffers is known as *bufferbloat*, which we referred to in Section 1.2.

for each source for each outgoing link in each network node. Similar conclusions were made by Morris in [111], who analysed the decreasing TCP performance with the increasing number of competing flows. He proposed a short-term solution, in which routers would be provisioned with buffer space proportional to the maximum number of active flows (even ten times as many buffers as flows). In order to avoid queue buildup, per flow limits on buffer use would be required though.

The complexity of switches with this amount of queues is, however, considerably increased. But with software switches this approach becomes feasible. Particularly in DAQ networks, where packets could be queued on a per data collector basis. Nagle used originally per source queues, because the sources have the control over the generation of packets. In the case of data acquisition networks, generation of packets is associated with a particular destination data collector, so per DCM queueing is more natural. In our design, event data sent from different ROSes, but targeting the same DCM, are put into a single software queue. With this approach buffers can be sized precisely, traffic can be shaped on a per DCM basis, and fairness across all DCMs can be guaranteed. In contrast to the previous schemes, buffering is provided by the receive and transmit queues in the NICs as well as those software queues. The total number and the size of the latter is limited solely by the amount of the available DRAM memory. Thus, lossless operation should be possible even under heavy incast congestion.

We use the hardware filters of each NIC again to assign the incoming packets into different hardware receive queues. All packets which have not been identified as packets carrying event data are placed into the RxQ 0 (*the default RxQ*), whereas other queues (*data RxQs*) are dedicated to packets carrying event data. A set of lcores poll packets from the assigned receive queues as in the two previous schemes. But instead of switching the data packets into one of the output transmit queues, they are temporarily buffered in dedicated queues. This buffering of event data packets takes place in the DPDK lockless ring buffers [80], which we refer to as *daqrings*. A single daqring corresponds to a single destination DCM. This binding can be either set statically or configured on-the-fly by the default thread, which implements flow detection logic. Three hardware queues are used on the transmit side for outgoing packets: TxQ 0 (*the default TxQ*) for the non-data flows and the other two (*data TxQ*) for the data flows.

We define four types of user-level threads executed by the available lcores:

1. management (master lcore),
2. default,
3. data receive,
4. data transmit.

There is always one management and one default thread, and a configurable number of data threads.

THE DEFAULT THREAD This thread polls the default RxQ of all the NICs that are bound to the software switch. It is implemented with the use of the DPDK packet processing pipeline [80]. First, the destination IP address is extracted from the received packet and looked up in the Longest Prefix Match (LPM) table [80] for the output port. Then, from the TCP payload, it determines whether the flow is a new event data flow of the ATLAS DAQ/HLT system. If so a new ring buffer is activated and corresponding rules are created in the hardware filters of the NICs, so that all subsequent packets will be filtered to the data RxQs and handled by the data threads. Finally, the default thread puts the packet directly into the default TxQ (TxQ-1) of the output NIC.

RX FILTERING Each receive data queue is bound to a single output port. The LPM lookup mechanism of the default thread is thus offloaded to the hardware. Within a single receive data queue, a specific ID is assigned to the packet based on the destination DCM by the hardware filter, which is then accessed by the data threads to enqueue the packet on the appropriate daqring. A 1:1 queue-to-DCM mapping would not be scalable because of the hardware limited number of the supported receive queues.

There is no risk of out-of-order packets since packets belonging to a single TCP flow are always filtered to the same hardware queue by the flow director, then handled by a single lcore, and finally enqueued on the same software ring. In theory it limits the data rate of a single flow to what can be processed by one CPU core, which is then the natural performance limit of the proposed architecture. Due to the fact that DAQ networks carry a large number of relatively small TCP flows, it is practically impossible to reach the performance limit of a thread with just a single flow. Furthermore, high forwarding performance of a single CPU core has been already confirmed in [52].

Normally, a DCM is uniquely identifiable by its destination IP in the DAQ network, so the hardware filters can be configured solely with the destination IP field of the packet header.

THE DATA THREADS AND RING BUFFERS Packet descriptors from the receive data queues of the NICs are polled by the data receive threads. Based on the queue number and the filter ID of the packet, they enqueue the packets on the appropriate ring. Since all packets belonging to the flows coming from all the ROSes to one DCM will be queued in the same DPDK ring, the rings are of multi-producer single-consumer type.

On the transmit side, the data transmit threads dequeue packets from the rings and place them into the transmit data queues. Each data

transmit thread serves the transmit data queues of one or more NICs. In our prototype we use two transmit queues for data flows: one for packets with the actual event data directed to DCMs, the second for requests and TCP ACK packets directed to the ROSes. The transmit threads also perform traffic shaping, which is particularly important, if the next hops in a network have limited buffering/bandwidth capabilities. Since the event data targeted to a particular DCM is queued in a dedicated buffer, traffic shaping can be performed very effectively on a per destination DCM basis. For each daqring we can set a specific rate limit by defining the maximum number of packets that can be polled from this queue within some polling interval, which is also controllable by the user. The excess packets are not dropped, but buffered in this daqring, unless its length is exceeded. This algorithm is referred to in the literature as *buffered leaky bucket* [99].

If a particular DCM ring remains empty for a predefined period of time it can be deactivated. In this case, it can be reused by the default thread for any newly detected data flows.

5.4.2 *Evaluation results*

The results of our evaluation for the designs presented in the previous section are gathered in Figure 5.5. In order to verify that the software switch can operate at its full bidirectional bandwidth (resulting from the number and speed of the NICs) we emulated twelve ROSes and 144 DCMs on all available hosts attached to the switch, which results in an all-to-all communication scenario with heavy incast congestion. The details of the evaluation setup were presented in Section 5.3.1. For this comparison the event rate remains unlimited, which means that we measure the saturation goodput. We analyse how the performance is affected when changing the number of CPU cores that are used by the software switch for processing packets carrying event data[7]. In each iteration we increase the number of cores by two, one on each NUMA node.

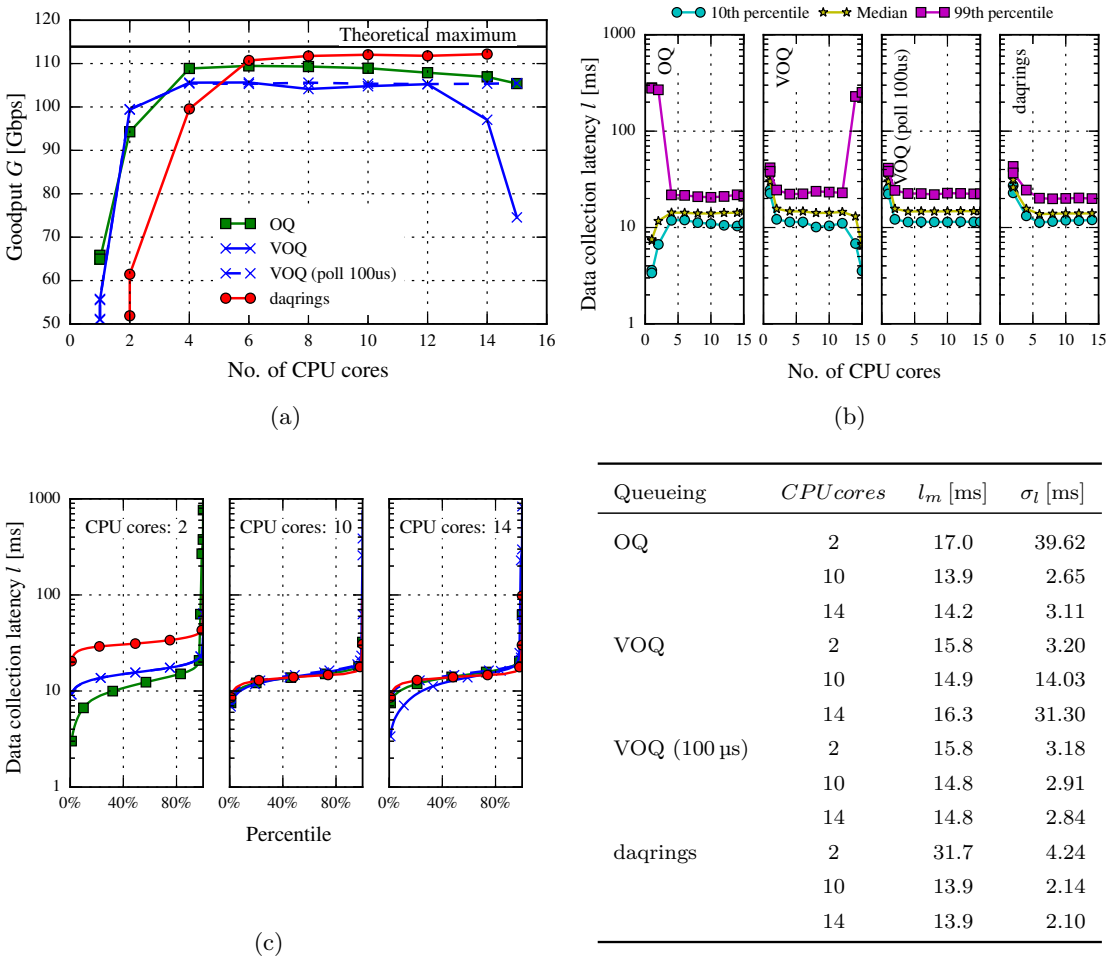
GOODPUT Figure 5.5a confirms that dedicated queueing, daqrings, reaches the best performance. We achieved about 98.5 % of the theoretical throughput, which corresponds to an average bandwidth of 118 Gbps at the software switch. Highest performance is already achievable with only six CPU cores devoted to data threads (three transmit and three receive threads). With the total of 144 DCMs in this configuration 144 ring queues are active in the software switch.

7 For OQ and VOQ there is additionally one master lcore and for daqrings there is one master lcore and one default lcore (see Figure 5.4). These lcores are not taken into consideration because they are not directly involved in switching packets that carry event data.

Output queueing reaches highest performance already with four CPU cores. This is expected as there are fewer operations required by the software switch when compared to daqrings, which requires, for example, additional enqueue-dequeue operations for the dedicated queues. The goodput saturates for OQ at lower value though, reaching 96.1 %. Here, we can also observe an unexpected effect — the performance first improves with the number of CPU cores, but at some point starts to degrade slightly. This effect is even stronger for VOQ, which reaches 92.8 % in the best case, but only 65.5 % when the maximum number of cores is used. The reason for this behaviour is not entirely clear, but it appears to be related to the fact that the NIC queues are continuously polled by the CPUs, even across the QPI links. If a delay of 100 µs is added between consecutive polls at each receive queue, the effect is avoided. We analyse this problem in more detail in Section 5.4.2.1.

At first glance, the results for VOQ seem wrong. In this design the maximum number of receive queues is equal to the total number of ports connected to the software switch, which is dictated by the design. It means that increasing the number of CPU cores beyond the number of ports should not affect the performance as these cores are not used. It contradicts, however, the plot presented in Figure 5.5a. The performance should not change if there is more than twelve cores. The reason is as follows. We increase the number of cores by two in each iteration, one core on each NUMA node. Each port is assigned to this lcore that belongs to the same NUMA node. In our configuration (see Figure 5.3b), there are eight ports connected to NUMA node 2. First with seven cores on NUMA node 1 (eighth core is the master lcore) and eight on NUMA node 2, ports are fully distributed across distinct processing threads. This emphasises the importance of considering the NUMA architecture when designing these schemes.

LATENCY The distributions of the data collection time per event are presented in Figure 5.5b and Figure 5.5c. Daqrings provide lowest latency and jitter. Even at lower core counts there are no packet drops as there is always enough buffers. Latency is increased only because there are fewer CPU cycles available. At the optimum operating point, there is a tail present (see 99th percentile). This is caused by the many-to-many communication and the fact that we did not apply any traffic shaping on the daqrings, which can lead to small queue buildup. The average latency with six data threads and more for all CPU frequencies equals approximately to 13.8 ms, which is comparable to the serialisation delay of twelve events on a 10 Gbps link (14.6 ms). Since twelve DCMs are emulated on a single host and the ROS responds to their requests on a first-come-first-served basis, some events are collected with a minimum latency approaching the serialisation delay of a single event on a 10 Gbps link (1.2 ms). There are also events that suffer from the fact that each ROS responds to all 144 DCMs in the system, so



Queueing	$CPU cores$	l_m [ms]	σ_l [ms]
OQ	2	17.0	39.62
	10	13.9	2.65
	14	14.2	3.11
VOQ	2	15.8	3.20
	10	14.9	14.03
	14	16.3	31.30
VOQ (100 µs)	2	15.8	3.18
	10	14.8	2.91
	14	14.8	2.84
daqrings	2	31.7	4.24
	10	13.9	2.14
	14	13.9	2.10

Figure 5.5: Comparison of the three queueing strategies of a dedicated switching application in the configuration depicted in Figure 5.3. Goodput (a) and event data collection latency (b) in function of the number of CPU cores used by the application for switching packets, and the exact distributions of latency for three cases (c).

their collection latency can increase even above the serialisation delay of data belonging to twelve events.

TCP timeouts are observed for OQ at lower core counts, which manifests itself by the fact that some events suffer from collections latencies exceeding 200 ms. On the contrary, packet drops are first observed with higher core counts in case of VOQ, which is related to the issue indicated earlier in this section. In the optimum operating points, the latencies in both cases are comparable with daqrings. The latter reaches, however, slightly higher load.

5.4.2.1 *Effects of excessive polling*

In the previous section we observed an unexpected behaviour in case of OQ and VOQ designs. Performance degrades when we increase the number of CPU cores that are used to perform packet switching. This

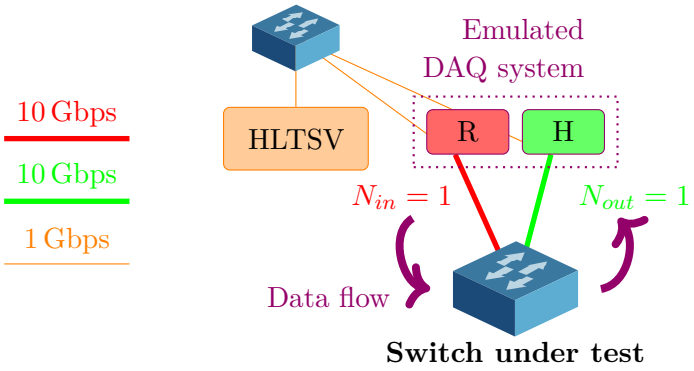
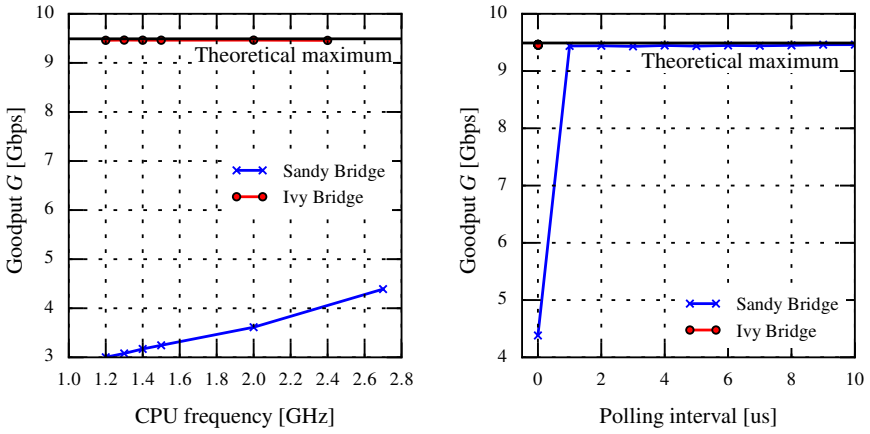


Figure 5.6: Simplified test setup to investigate the effects of excessive polling.

effect is particularly strong in case of VOQ. The reason is not entirely clear, but it appears to be related to the fact that the NIC queues are continuously polled by the CPUs. This is particularly critical, when polling for the received packets across the QPI links. With an increased number of CPU cores there are more threads polling in parallel. If some of the receive queues are empty, there are cores that continuously poll the NICs as there is no other processing required. In parallel, there can be threads that perform packet receive/send operations. If they share the QPI links, the performance can be affected by the cores polling empty receive queues. This effect is particularly visible with the VOQ strategy, as there are always some receive queues that are polled by remote NUMA nodes. This is not the case for OQ and daqrings, where receive queues are always polled by the local NUMA node from the perspective of the particular NIC. CPU caches and direct PCIe lanes to the NICs provide optimum performance even with continuous polling. This theory is supported by the experiment with increased polling interval in the VOQ case, see Figure 5.5a. The performance does not degrade any more. Similar observations were made by Emmerich et al. in [52].

We perform one more experiment to confirm this observation and investigate whether the issue still appears with newer server generations. For this purpose we build a simpler version of the setup from Figure 5.3, as presented in Figure 5.6. Only a single ROS application and a single data collector on separate physical nodes are used (theoretical goodput is 9.49 Gbps). At the software switch, we use two 10GbE ports on two different NUMA nodes. First, we use the same platform as before, with two EP-2600 (code name *Sandy Bridge* [85]) series processors (see Section 5.3.1), and then we change to Intel Server System R2208LT2HKC4 [83] with the next generation EP-4600 v2 (code name *Ivy Bridge* [86]) processors (there are four Intel Xeon E5-4657L v2 processors, but only two are used in our evaluation).

As can be seen in Figure 5.7, the software switch based on the newer generation server achieves the theoretical performance. In case of the older generation, the performance degradation is significant. First when



(a) (b)

Figure 5.7: Goodput in the function of the CPU frequency (a) and polling interval (b) in a small setup comparing the performance of SandyBridge and IvyBridge platforms with the VOQ design in the software switching application.

the polling interval is introduced, see Figure 5.7b, the performance reaches the theoretical maximum.

This experiment confirms that excessive polling of the receive queues over the QPI bus can negatively impact the performance of a software switch. In general, special attention is therefore required when the QPI links are involved. Performance degradation can be mitigated or completely avoided with newer generation processors and/or designs with receive-side polling with proper CPU-affinity.

5.4.2.2 *Traffic shaping in daqrings*

We come back to our original evaluation setup from Section 5.3.1 and evaluate traffic shaping capabilities of daqrings, which we explained in Section 5.4.1.3. We restrict the transmit data threads to put no more than 32 packets from a single ring buffer into a single transmit queue every 500 µs, which limits each DCM to approximately 0.78 Gbps at the software switch. The data receive threads are configured to drop packets, if there is no space available in the ring of the corresponding destination DCM. We still expect no drops if the buffering is large enough for a single event. This approach shows how incast could be avoided in a situation when DCMs are connected with slow links to a ToR switch. By limiting the rate we ensure that the ToR switch buffers are not overrun and the packets are buffered at the software switch instead.

Goodput and latency characteristics as a function of the daqring size are shown in Figure 5.8. A single event requires approximately 990 TCP segments over eleven TCP flows towards a single DCM. Maintaining small packet buffers (i.e. 128 or 512 packets per DCM) triggers incast

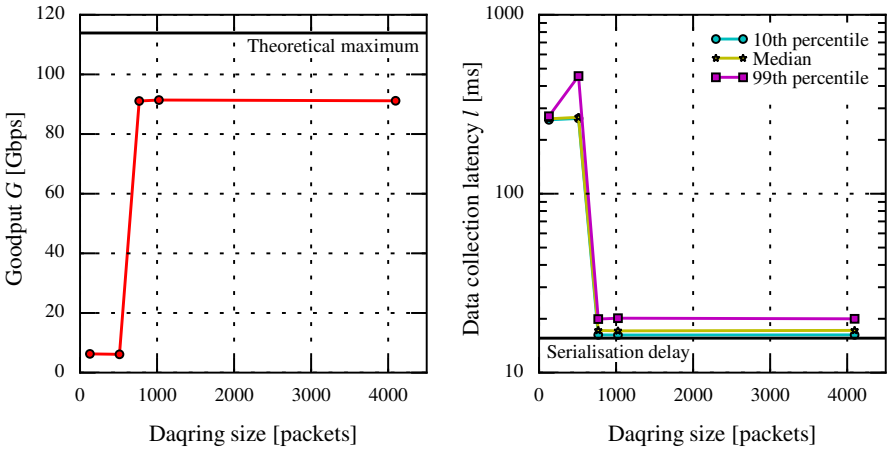


Figure 5.8: Goodput (a) and data collection time (b) when tuning the daqring size, when a rate limit of 0.8 Gbps is applied to each daqring.

and results in a very high mean data collection latency because of high TCP timeout rate. In the optimum operating range, the latency remains at its minimum value of 17.2 ms with low jitter, and no packets are dropped. The serialisation delay of the entire event data on a link limited to 0.78 Gbps, including protocol overheads, equals to 15.6 ms and gives the lower limit on the minimum data collection time.

The direct effect of the collection latency is the maximum achievable event rate of the system. In the optimum switch configuration the DAQ throughput achieves 80 % of the theoretical maximum. The performance from Section 5.4.2 is not reached because of the traffic shaping we have applied and the performance limitations of the traffic generation setup.

With the total of 144 DCMs in this configuration, 144 ring queues are active in the software switch. For the single ring size of 4096 packets and 2048 B of maximum packet size configured at the switch, the total packet buffer space equals to 1.12 GiB.

5.4.2.3 *Increased burstiness with a single DCM*

We now increase the burstiness of the traffic by increasing the number of emulated ROSes to 110. This special configuration is depicted in Figure 5.9. We emulate ten independent ROS applications on a single physical host and there are eleven nodes in total for ROS emulation. Only a single DCM is used in this test and it is rate-limited to 0.78 Gbps at the daqring. The total event size is now 13.75 MiB and the serialisation delay of the entire event, with protocol overheads, is 156 ms. The event size corresponds to approximately 9790 TCP segments over 110 TCP flows.

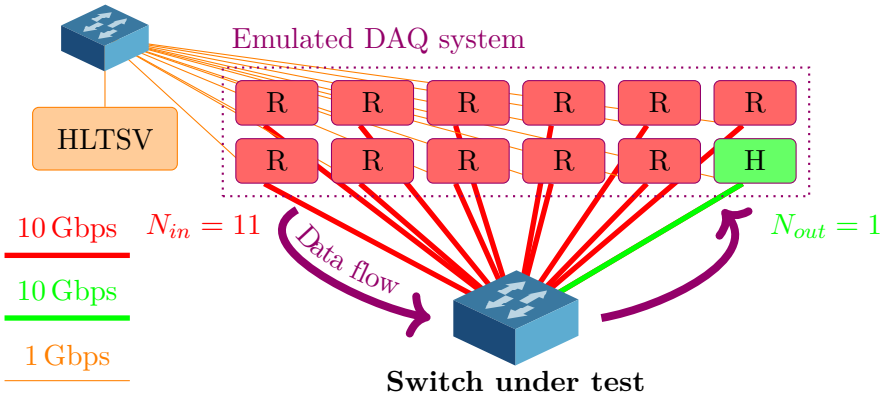


Figure 5.9: The special data taking configuration for a scenario with one data collector and 110 ROS applications. The latter are emulated on eleven physical hosts.

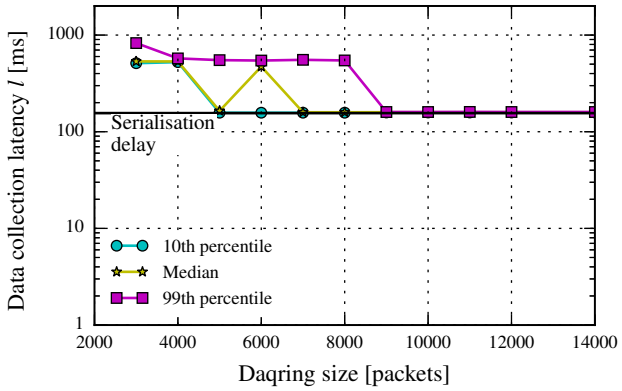


Figure 5.10: Data collection latency when tuning the daqring size for a setup with one data collector and 110 readout applications (see Figure 5.9).

The latency characteristics are presented in Figure 5.10, again as a function of the ring buffer size. Starting with a length of 9000 packets (nearly the number of packets required to carry the entire event) there is no jitter and the mean latency remains at 159 ms, very close to the serialisation delay. Reducing the ring size below 9000 triggers incast with high jitter and mean latency, which are caused by the high number of TCP retransmissions. Because of the large bursts, the DCM queue fills up and packets are dropped by the switch. For queues below 5000 every collected event suffers from at least one TCP timeout of a minimum value of 200 ms.

5.4.3 *Summary*

Our results show that the proposed design of the software switch with dedicated queueing mechanism can be configured to match the specifics of a DAQ system and operate at full available bandwidth with low

jitter and without packet drops, thus eliminating incast. This is also true under heavy congestion and for packets arriving in extremely large bursts. These results are possible thanks to the available amount of system memory that can be used for packet buffers, and the architecture of modern CPUs, which provide the necessary performance. Equally important is the presence of packet processing frameworks, which allow the design of dedicated network applications tailored for a specific configuration, like our lossless software switch for the ATLAS DAQ network.

This application, being a proof-of-concept, lacks typical functionalities required from networking devices, like MAC learning, IP routing, or support for typical management interfaces and protocols. Also, the dedicated data flow detection logic is implemented for the specific application layer protocol — the ATLAS data-flow protocol. These drawbacks make the approach less generic and less likely to be adopted in production networks. In the next section we will evaluate whether a similar queueing system could be implemented in one of the popular, production quality virtual switches.

5.5 OPEN VSWITCH OPTIMISATION FOR DAQ NETWORKS

In the previous sections, we demonstrated that a DPDK switching application with dedicated queueing mechanisms running on a commodity server is a viable option in data acquisition for providing lossless operation under heavy incast congestion. Our next step is to evaluate whether the same can be achieved with the popular virtual switch — Open vSwitch (OVS). As we explained already in Section 2.3, OVS is a production quality, open source virtual switch with built-in support for DPDK. It can be used not only in virtual environments, but also as a software switch/router interconnecting physical ports on a general-purpose server. It supports typical network management interfaces and protocols. In summary, OVS can be considered as fully-fledged switch and, on the other hand, it can be optimised for data acquisition, being an open source software project.

Furthermore, OVS supports also the OpenFlow (OF) [126] protocol and the Open vSwitch Database (OVSDB) [134] protocol. This enables the SDN approach to building networks, which we introduced in Section 2.5. In SDN the networking control and forwarding planes are physically decoupled and the network intelligence is centralised in an SDN controller, which has a global view of the entire network. This controller is responsible for maintaining all of the network paths and programming each device in the network [113]. This is particularly interesting in the context of DAQ, where the global view of the network could be used to optimally distribute data flows across available paths and queues. The logic to assign packets to appropriate daqrings would not need to be integrated within the switch itself. This can be seen

as a drawback of our design presented in Section 5.4.1.3, where we implemented the ATLAS-dedicated mechanism to detect data flows. Therefore, by implementing our daqrings approach in OVS with a centralised management, a more generic solution for large DAQ networks becomes feasible.

We will discuss some aspects of building larger networks topologies with our proposals in Chapter 6. Here, we will focus on the single-switch performance of OVS in data acquisition and adapt our proposed queueing mechanism. We will analyse if there are any performance penalties when compared with the dedicated switching application that we evaluated in the previous sections of this chapter.

5.5.1 *Design*

The core of the idea behind the design of the lossless software switch for DAQ, which we described in Section 5.4.1.3, can be summarised in three points:

- The use of the DRAM memory as a large packet buffer.
- A queueing mechanism, in which a dedicated, large-enough queue is allocated in the switch to every single data collector (see Figure 5.4c).
- A mechanism that maps the packets destined to a particular data collector into its queue.

In the original design all of these items were implemented in a dedicated switching application using the DPDK framework.

5.5.1.1 *Traffic distribution*

Our extension to OVS differs in that the DCM-to-queue mapping mechanism is moved to an external network controller. To achieve this we added a new port type to the virtual switch — a *daqring port*, which represents a single queue, which is implemented as a DPDK ring, similar as in Section 5.4.1.3. Because each of these queues is visible as a logical port in OVS, the network controller can decide which packets should be moved to the daqrings, where they are temporarily buffered, instead of directly switching them to the standard egress ports.

The controller uses OVSDB to instruct the switches to create a single daqring device for every DCM that is identified in the system. Then it installs a set of OpenFlow rules on the switches, so that they are programmed to move specific packets from the ingress ports to appropriate daqrings and later to dequeue them to the egress ports. In the simplest case, if a single DCM is identifiable by its IP address, two OpenFlow rules for every DCM are enough on every switch:

- If the packet's destination IP address matches the one of the DCM, output the packet to the corresponding daqring port.

- If the packet is received from a daqring port, output the packet to the egress port.

The second rule must have a higher priority than the first one to avoid trapping the packets in a loop. These rules can be adjusted at the controller to match the architecture of the given DAQ system. For example, for our evaluation in the following section we are emulating multiple DCMs on a single host, so the destination IP address is not the unique ID of a DCM any more. We are using therefore a set of rules based on the 4-tuple (source/destination IP, source/destination TCP port) for every ROS-to-DCM flow. We will explain the algorithm that is used by the controller to assign the egress port for our proposed topology with multipath in Chapter 6.

5.5.1.2 *Fairness*

Fairness across the DCMs is guaranteed by polling the daqrings in a round-robin fashion. Rate limitation on the daqrings can be enabled by limiting the number of packets that are polled in a single polling cycle (controllable with OVSDB) using the same buffered leaky bucket algorithm as in Section 5.4.1.3.

5.5.1.3 *Alternative approach*

OpenFlow version 1.0.0 [126] already allows the creation of Quality-of-Service (QoS) queues and their assignment to ports in the switch. Potentially our daqrings could be implemented using those QoS queues, so there would be no need to create separate logical ports. However, we decided not to use this approach at this time because of the specific design of the DPDK devices in OVS. The implementation of the new daqring port did not require any significant modifications to the OVS architecture. We would not expect any significant difference in performance, because the underlying mechanisms to move the packets between the internal queues would remain the same.

5.5.2 *Implementation*

We added 791 and removed 11 lines of code in seven files, including the build files and the manual, of the OVS release 2.4.0 (with DPDK 2.0.0) to implement the daqring device and optimise the switch for high throughput. For the SDN controller, we used the OpenDaylight project release Lithium [167]. We used the REST API [60] and Python [140] to automate the installation of the ROS-DCM flows on the switch. For testing purposes or for small deployments, the *ovs-ofctl* tool provided with OVS can be used instead to install the OpenFlow rules.

Some optimisations to the OVS code were required to reach the full performance in our DAQ scenario (for details on OVS's design see [136]). The critical modifications included:

1. Increased number of hardware packet descriptors per network port and larger memory pool to buffer packets.

2. GCC optimisation level increased from O2 to O3.

3. Modification to the OVS flow caching mechanism — Exact Match Cache (EMC). By default, any change in the TCP flags of the packets in a particular ROS-DCM flow was causing a modification of the entry in the EMC. In the case of ATLAS DAQ, each last packet from a ROS response is marked with the TCP PUSH flag, causing systematic EMC updates. Since there were no OpenFlow rules using TCP flags, we removed the flag from the cache key as such rules are not normally needed in DAQ networks.

4. Removal of an intermediate OVS software transmit queue in the DPDK device. Instead the packet descriptors are put directly into the hardware queues of the network cards. Since the software queues were all allocated on a single CPU, costly copy operations were needed to move packet descriptors between these queues and the port hardware queues connected with PCIe to other CPU sockets.

Although optimisation level 2 can be seen as generally applicable, the other modifications are specific to a narrower set of throughput-oriented applications. Therefore, we decided not to send the patches to the maintainers. The source code is available in [43].

In order to achieve the highest performance of our switch, we continue to follow the general recommendations, which we discussed in Section 5.2.2.

5.5.3 *Evaluation*

We evaluate our implementation of the lossless OVS-based software switch using the basis configuration used throughout this chapter (see Section 5.3.1). Figure 5.11 gives the comparison of the performance between the DPDK DAQ-dedicated application that was introduced and evaluated in Section 5.4.2 and OVS with our various optimisations:

- *OVS+sw_tx_queue.* OVS without daqrings, and with the default OVS software tx-queue. Reaches only 30.6 %.

- *OVS+daqrings-EMC_fix.* OVS with daqrings and all optimisations, except the one for the Exact Match Cache. Reaches 98.3 % with ten or more CPU cores.

- *OVS+daqrings+Pause+gcc_02.* OVS with daqrings and all optimisations, but also with IEEE 802.3x pause frame [76] and the default GCC optimisation level. Reaches 98.8 % but requires twelve CPU cores. Pause frames eliminate packet drops for lower core

counts, but increase the mean latency (see the plot on the left-hand side in Figure 5.11b).

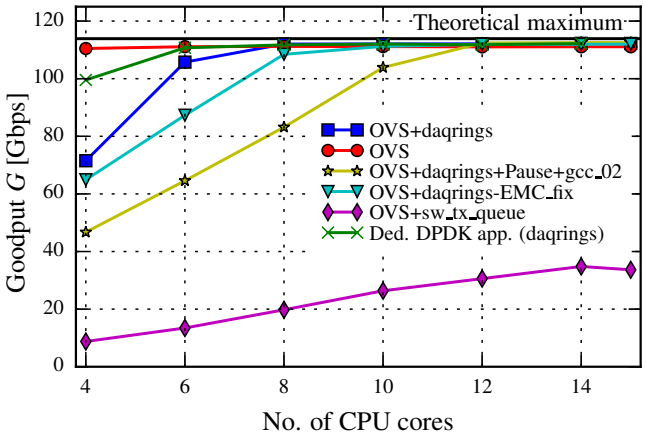
- *OVS.* OVS with all optimisations, but without daqrings. Reaches 97.6 % of the theoretical goodput. No performance degradation even with only four CPU cores.
- *OVS+daqrings.* OVS with daqrings and all optimisations. Reaches 98.6 % of the theoretical goodput. There is significant performance degradation for fewer than eight CPU cores.
- *Ded. DPDK app. (daqrings).* Reaches 98.5 % of the theoretical goodput. There is significant performance degradation when using fewer than six CPU cores.

In *OVS+sw_tx_queue* and *OVS+daqrings-EMC_fix* a significant number of data transfers suffer from TCP timeouts with collection latency exceeding 200 ms (for four CPU cores), which indicates packet drops. For all other cases no timeouts are observed and the latency distribution does not exhibit significant jitter. In case of fourteen CPU cores the median latency is similar for all configurations, but not for *OVS+sw_tx_queue*. When using only four CPU cores the median increases for *OVS+daqrings*, and for *OVS+daqrings+Pause+gcc_02* even more so.

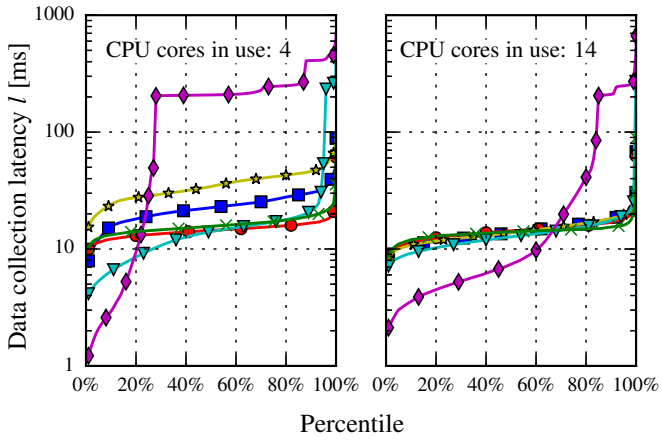
The results show that the optimised OVS delivers near maximum performance because of the large buffering capabilities. Using daqrings, performance increases slightly even further. We predict that this difference will increase for larger topologies, which we will discuss in Chapter 6. OVS with daqrings, similar to the dedicated application from Section 5.4.1.3, offer the users better flexibility to provide fairness across data collectors and exploit rate/burst control to eliminate packet drops, if, for example, subsequent network stages do not provide enough capacity (e.g. see Section 6.3 and Section 7.3). *OVS+daqrings* requires, however, more CPU cores than pure optimised OVS because of the additional port send/receive operations associated with the extra daqring devices. Note, there is only a small performance penalty for lower core counts, when compared to the dedicated DPDK software switch. For optimal core counts the performance remains similar. This confirms that OVS with our extensions is a valid candidate and there is no need to design and implement fully custom, dedicated switching application in order to achieve the same effect. Furthermore, this approach offers even more advantages as discussed already in Section 5.5.

5.5.4 *Detailed performance characteristics*

In our initial design of the dedicated switching application (see Section 5.4.1.3) the exact number of the receive queues configured on the NICs was determined by the queueing strategy in use. In the case of



(a)



(b)

Figure 5.11: Saturation goodput (a) and latency distribution (b) in all-to-all incast scenario with modified Open vSwitch.

OVS, this number is freely configurable by the user. As these are the first queues on the ingress side of the software switch, we analyse how it affects the overall performance. Internally, OVS uses the Receive Side Scaling (RSS) feature available on the NICs to distribute the incoming packets across available receive queues using hashes [79]. These queues are then assigned to the available lcores for polling and packet processing. On the other end, egress, the number of transmit queues per NIC is not directly configurable in any of these cases, but it is often linked with the number of configured lcores. In order to make this analysis exhaustive, we consider here the total number of lcores used by OVS and also different CPU frequencies. The size of the transmit and receive queues remains the same as throughout this chapter, being the maximum allowed by the NICs.

Intuitively, with higher CPU frequencies fewer cores and fewer receive queues should be required to reach the theoretical performance. This is confirmed by Figure 5.12a, which shows the saturation goodput of our

DAQ-optimised OVS for various combinations of the aforementioned parameters. At 2.7 GHz performance close to maximum is reached already with eight CPU cores and three receive queues on each NIC port. At 2.0 GHz the same amount of cores with ten queues or twelve cores and two queues guarantees similar performance. The combination of twelve cores and four receive queues is required for 1.2 GHz. These queues are mainly used to distribute packet processing across available lcores, but also provide some buffering for short bursts, when the lcores are not polling packets fast enough.

Distributions of data collection latency, presented in Figure 5.12b and Figure 5.12c, lead to similar conclusions. Lower goodput values are correlated with increased latencies and in extreme cases TCP timeouts with collection times exceeding 200 ms. In the optimum operating ranges, there are no packet drops, but slightly higher latencies are observed when fewer CPU cores are used to reach the same saturation goodput. This is expected as each packet has to wait longer before being processed, when fewer CPU cycles are available to process the same amount of traffic. The same observation can be made when lowering the CPU speed, but only for lower total core counts. With 15 lcores differences in the latencies are negligible in the optimum range. This may be caused, however, by the fact that the total processing power is increased by a factor of four and when changing the CPU frequency this factor is approximately two.

Summarising, in order to reach full theoretical performance of our evaluation setup only a fraction of the available CPU power is required. This leaves a significant headroom to build software switches providing even higher capacities.

5.6 OTHER ASPECTS

Until now we have focused on the most important aspect when evaluating the feasibility of building data acquisition networks with software switches, mainly the performance. We showed, on the example of a single-switch network, that this approach is indeed feasible. Here, we will also show that it offers good performance advantage when comparing with traditional switches. Nevertheless, there are also other factors that need to be taken into account. First of all, all aspects of building and managing large network topologies need to be considered. This will be discussed in Chapter 6. In this section a short study on energy consumption will be given. Power is an important factor as it affects the costs of running a network for a long period of time. On the other hand, we will also describe how the unused CPU cycles on the software switch nodes can be utilised in order to improve the overall efficiency of a system.

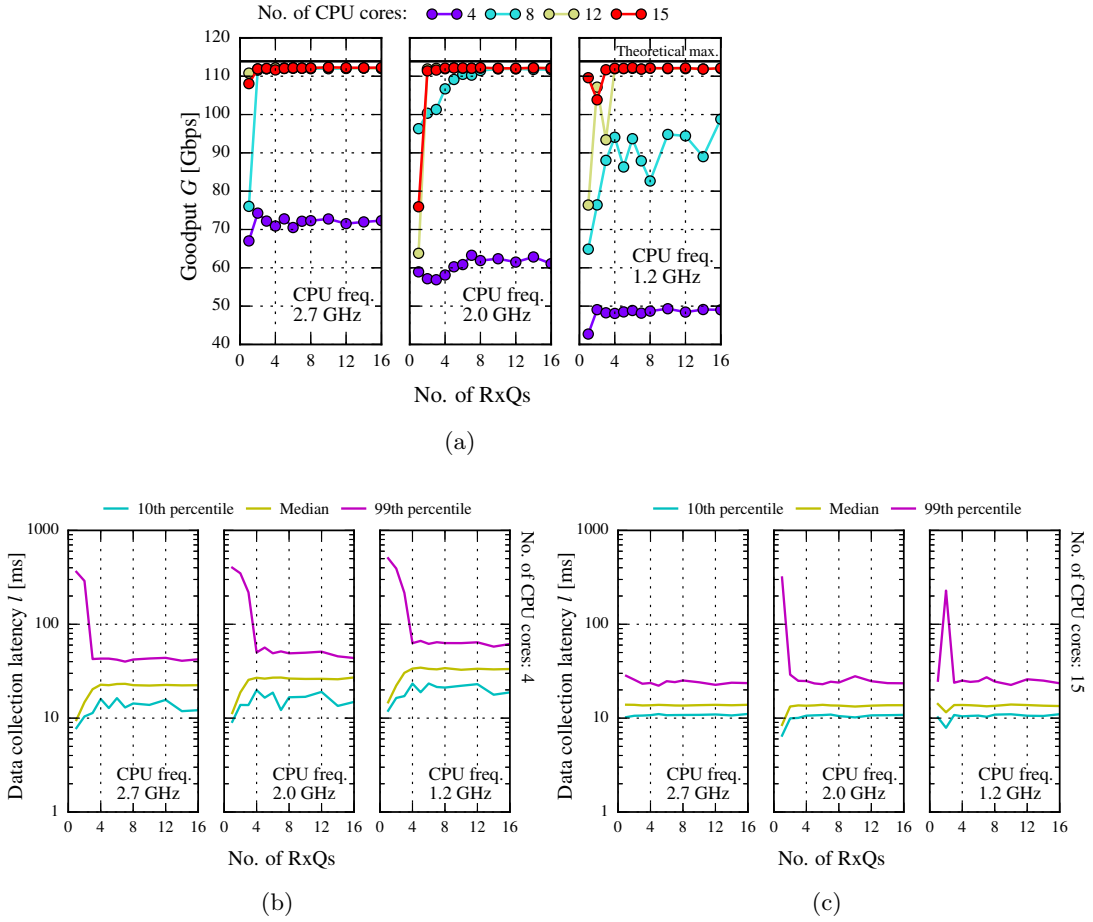
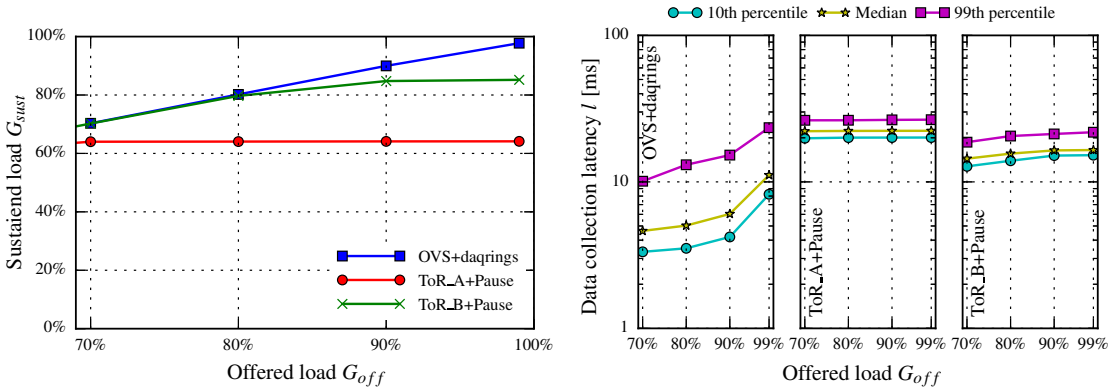


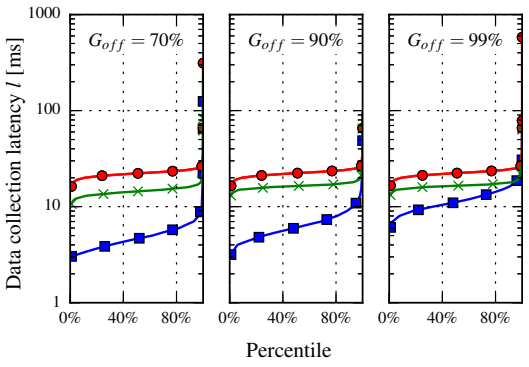
Figure 5.12: Saturation goodput (a) and latency distribution (b), (c) in all-to-all incast scenario with modified Open vSwitch when changing the number of CPU cores, their frequency and the number of receive queues on the NICs.

5.6.1 *Comparison with traditional switches*

We continue our evaluation of software switching for DAQ and extend the study that we performed in Section 4.5.3. We compare the DAQ-optimised OVS with two 10GbE ToR switches from different vendors in the same all-to-all incast scenario (the switch under test in Figure 5.3a acts as one of the ToRs). We enable transmission and reception of pause frames on those ToRs. As it was shown in Section 4.5.3, without this mechanism the performance does not exceed 20 %. In case of OVS, as we noted in Section 5.5.3, only the reception of pause frames is enabled in order to avoid temporal fluctuations at the data collector nodes.

The results in Figure 5.13a show the sustained versus offered load (because of time limitations only values starting from 70 % are tested). The ToR switches reach only 64 % and 85 % compared to nearly 100 % achieved by OVS. Latency characteristics are plotted in Figure 5.13b and Figure 5.13c. Timeouts are avoided in each configuration, but the latencies achieved with ToRs are significantly higher than those of the





Configuration	G_{off}	l_m [ms]	σ_l [ms]
OVS+daqrings	70 %	4.9	2.0
	90 %	6.6	2.6
	99 %	11.8	3.5
ToR_A+Pause	70 %	22.1	2.2
	90 %	22.2	2.2
	99 %	22.3	2.3
ToR_B+Pause	70 %	14.5	1.8
	90 %	16.5	1.7
	99 %	16.6	1.7

Figure 5.13: Performance of OVS with daqrings and regular ToR switches with enabled Ethernet IEEE 802.3x pause frame mechanism. Sustained load (a) and data collection latency (b) as a function of the offered load, and the exact distributions of latency for three different cases (c).

OVS. The increased latency for OVS at the higher load of 99 % is expected and caused by the volume of data flowing through the switch. It is not seen for the ToRs, since they do not offer the same load and saturate at lower values.

5.6.2 *Energy consumption*

Figure 5.14 gives an estimate for power consumption of the server used as a replacement for the conventional switch. We compared the maximum power per port. The values for the ToRs were taken from their data sheets, whereas the ones for OVS are plotted for different CPU frequencies as well as CPU core counts and correspond to full achievable load (at least 95 % in all cases). Although the server running OVS provides better performance, the power consumption is higher, especially when compared with ToR B. The situation can be improved in two

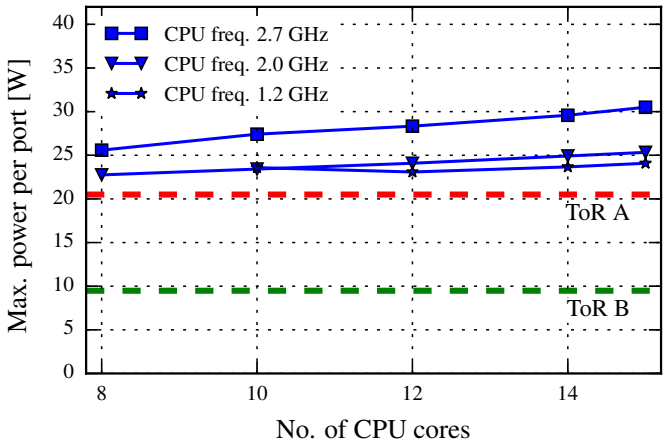


Figure 5.14: Average per port power consumption in all-to-all incast scenario with DAQ-optimised Open vSwitch for various CPU frequencies. In all cases goodput is at least 95 % of the theoretical maximum.

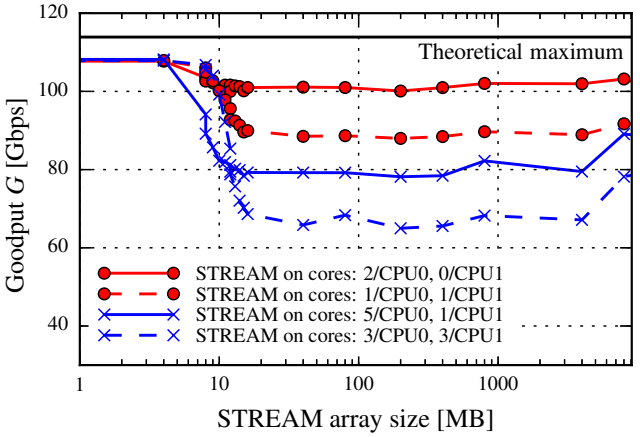
ways. First, we use continuous polling of all the OVS ports for incoming packets. This could be improved by reducing the number of empty polling cycles, and, in effect, reducing the power consumption. Second, remaining unused CPU cores could be used to perform different tasks, e.g. event filtering or experiment readout, optimising the utilisation of the entire system. We provide an initial evaluation in the following section.

5.6.3 *The use of the remaining cores*

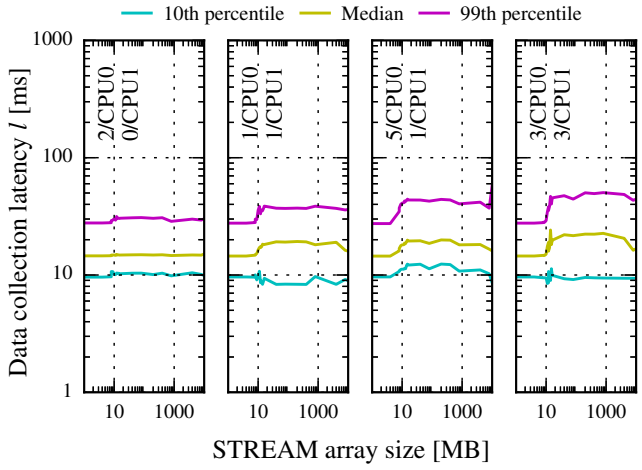
Our results from the previous sections indicate that there is always a specific number of CPU cores that can guarantee a desired load that our switch sustains. Hence, there is the possibility to use the remaining cores to perform other jobs, provided that they do not degrade the switching performance.

The obvious resource, which could be the source of this negative interaction, is the memory. In order to emulate a situation in which this interference occurs we ran the STREAM benchmark [104] on some of the unused cores. This benchmark measures sustainable memory bandwidth, thus stressing the memory subsystem. We did not focus on the results of the benchmark themselves, but we observed the impact on the goodput and the data collection latency of our evaluation setup.

The results are presented in Figure 5.15. It is clear that switching performance is affected, particularly if the STREAM array size (used to perform operations defined in the benchmark) exceeds the threshold of a few megabytes. The degree of the degradation depends on the number of cores and their CPU-socket allocation. Although no TCP timeouts are observed, see Figure 5.15b, data collection latency is increased when the STREAM array size exceeds the threshold. In this



(a)



(b)

Figure 5.15: Goodput (a) and data collection latency (b) in all-to-all incast scenario, if some of the CPU cores that are not allocated to OVS are used to run the STREAM benchmark.

case, the more negative impact is evident when more cores are assigned to the STREAM benchmark on NUMA node 1. Similarly as in Section 5.4.2, it can be explained by the fact that more NICs are bound to this NUMA node, so they need to share the memory bandwidth with more STREAM threads. For the case of three cores devoted to STREAM on each NUMA node, lowest goodput and highest latency are observed. There are also packet drops, but all of them are resolved with fast TCP retransmissions. For a real application, careful analysis and tuning would be required to guarantee the desired level of performance.

5.7 CONCLUSION

In this chapter we analysed and proposed different designs of a lossless software-based switch targeting high bandwidth data acquisition networks with the aim of preventing TCP throughput collapse due to incast congestion. Our prototypes proved that saturation and lossless operation can be reached on real hardware providing the total bandwidth of 120 Gbps in the all-to-all incast scenario, where traditional ToR switches perform poorly. And, importantly, controls on the injected traffic are not required. Furthermore, our performance analysis as well as results reported by other researchers show that higher bandwidths are achievable.

First, we characterised DAQ networks and showed that the potential impediments of software switching, like latency or the offered bandwidth for small packets, are not critical to the performance of data acquisition systems. Instead, the nearly limitless memory and the flexibility of a software switch allows us to design a dedicated software switch with enormous packet buffers that could be considered as a potential replacement for the expensive feature-rich core routers, typical for large DAQ networks. We verified the correctness of this hypothesis on real hardware providing 120 Gbps bandwidth. We effectively prevented incast maintaining the system bandwidth with 144 data collectors receiving data on a total of 1584 TCP flows through a single software switch.

Second, we optimised the production quality Open vSwitch for DAQ traffic characteristics and implemented analogous queueing mechanism as in the design of the dedicated software switch. We reached saturation and lossless operation also in this case. Although slightly more CPU power is required, we gain the benefits of using a fully-fledged switch supporting standard protocols, which is an important factor in production systems.

We also showed that there is significant headroom in terms of available CPU power. This can be used either to increase switching capacity, reduce power consumption or perform different tasks. It is possible to combine the functions of a software switch and, for example, readout or filtering node, on a single server. Special attention and tuning is, however, required for shared access to the resources, memory in particular.

The small prototypes already reached, bandwidth-wise, figures comparable to the requirements of the ATLAS DAQ network in Run 2. We have not demonstrated yet that such an architecture will scale by two or more orders of magnitude for the future upgrades of the LHC experiments, which we will discuss in the following chapter along with other aspects such as administration, port density or costs of building a network based on software switches.

6

SOFTWARE-DEFINED DATA ACQUISITION NETWORKS

In the previous chapters we focused our attention on small-scale prototypes in order to evaluate potential approaches to improve the performance of data acquisition networks under heavy incast congestion. We demonstrated that optimised software switches are good candidates. We have not showed yet, how large-scale networks could be built and managed. This is discussed in this chapter. First, we remind our motivations behind taking the software-defined approach with a centralised control plane for building a larger topology of interconnected software switches. Next, we propose to use the leaf-spine topology, which is already a popular solution in datacenters. We show how it could be implemented and optimised specifically for data acquisition, focusing on aspects as bandwidth scalability, packet routing, load balancing, resilience, costs, and space constraints. In the end, we present a prototype leaf-spine data acquisition network consisting of eight software switches running on separate physical servers. This chapter contains results published in [89].

6.1 INTRODUCTION

In Chapter 5 we demonstrated that software switches with large buffers perform significantly better than typical switches under heavy congestion. A server with twelve 10 Gbps Ethernet interfaces acting as a prototype switch reached its maximum bandwidth of 120 Gbps. It completely avoided throughput degradation, while hardware switches reached no more than 85 % of the requested load under similar conditions.

Here, we intend to show that a number of software switches can be interconnected to build terabit networks. In this context, it is important not to focus on the performance only, but consider aspects related to building and operating a large DAQ network. Among them are bandwidth scaling, routing, management, or load balancing. No less important is to discuss the costs. In the end, they are among the decisive factors when choosing one networking technology over the other for the next data acquisition network.

The fundamental aspect is the choice of a topology that could scale to terabits under the specific traffic pattern of data acquisition. It is useful to take advantage of the lessons learnt in datacenters. The leaf-spine (Clos-based) network architecture has become a popular, cost-

effective, and efficient alternative for traditional network designs with large routers in the core of the network, which we discussed in Section 2.4. It is interesting to analyse the bandwidth scaling aspect of the leaf-spine network with the assumption of predefined communication pattern, characteristic of data acquisition. Although the leaf-spine topology has been already discussed and applied in data acquisition (see Section 2.4), the literature misses more detailed studies on DAQ-oriented load balancing to maximise the offered event bandwidth. In this context, it is straightforward to discuss the usability of Software-Defined Networking (SDN) technologies to centrally manage and optimise a data acquisition network. SDN, together with software switches and with our mechanism for DAQ-oriented buffering, can potentially provide an entire solution that can become an alternative for the future data acquisition networks.

We gave a brief introduction to SDN in Section 1.1.4 and we extended it with literature review in Section 2.5, taking into account the context of data acquisition. The most attractive feature is the ease of programming flows across a network and utilising its global view in order to optimise the performance of a DAQ system. Although this approach is also possible with traditional networks, it is highly dependent on the hardware vendors. Nevertheless, there are also some aspects of SDN that are widely disputed. Among them are the single point of failure with central network controller, flow installation latency, or the size of OpenFlow tables in switches. These, however, are not critical in the context of data acquisition, as it was shown in Section 2.5.

6.2 THE LEAF-SPINE TOPOLOGY FOR DAQ NETWORKS

Instead of combining hundreds of ports in a single core device, we evaluate the concept of the leaf-spine topology, which has become a popular choice in datacenters to build non-blocking fabrics. It is made with a large number of devices, but of a lower port density. In this architecture the core of the network is distributed over several spine switches. It is possible to use the same ToR switches both in spine and leaf layers. In our case, we focus on building the topology with the loss-limited software switches, proposed in the previous chapter, or a combination of software and ToR switches to overcome the incast congestion problem. We follow a similar approach as in the datacenter fabric designed by Facebook [7, 21], but adapt the topology and analyse its performance specifically for data acquisition.

6.2.1 *Design*

An example of how a DAQ network could be built using the leaf-spine topology is depicted in Figure 6.1. The basic network unit, a *pod*, is in our case a subset of ROS nodes (R) and filtering racks (H). A single

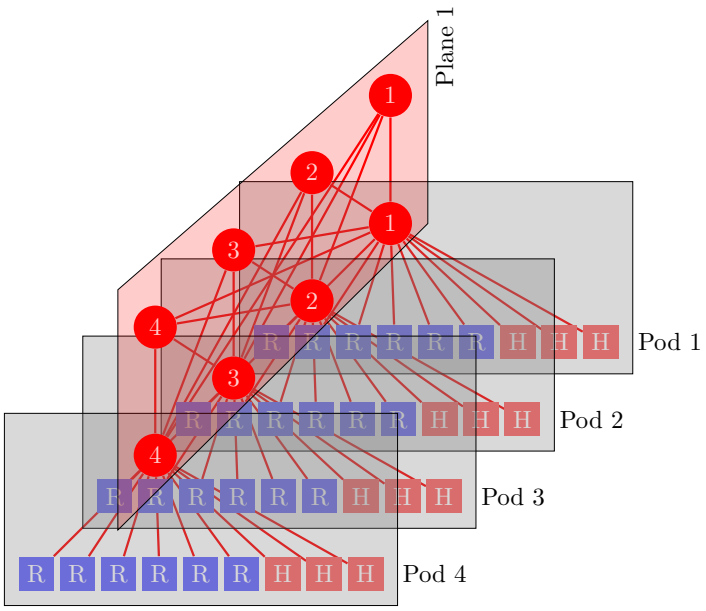


Figure 6.1: Architecture of a DAQ network in the leaf-spine topology. Note that leaf switches are not connected with each other.

filtering rack is a ToR switch that connects multiple servers. Each server runs a data collector and multiple event filtering processes, like in the normal DAQ system. The nodes in each pod are connected to the first stage of the network – the leaf stage. Typically, in DCNs, these nodes are the end-hosts only. In our case, however, we treat an HLT rack as a single entity, so the links within each rack are not considered (there is no traffic between data collectors). The leaf switches are used to connect the nodes in each pod and to load balance the traffic across the links of the spine stage. The spine switches provide connectivity between the pods. ECMP [75] (equal-cost multi-path) is often used in datacenters to provide hash-based load balancing. In Section 6.2.2 we will propose a method to optimally distribute all flows across available paths in a given DAQ scenario.

When designing the topology for DAQ, we need to consider though that, in general, the readout nodes can have multiple links to the network[1]. This becomes critical for providing resilience to link failures and increasing the overall DAQ bandwidth. Here, we introduce the term of an independent *plane* — sometimes called a leaf-spine plane. The leaf and spine switches in Figure 6.1 form a single plane. The ROS nodes and HLT ToRs can connect to a number of independent planes, depending on the number of available uplinks and bandwidth requirements. An example of this topology with four pods, six ROSes and three filtering racks in a pod, and four planes is depicted in Figure 6.2. The pods are interconnected in every plane by an independent leaf-spine plane.

1 In the present ATLAS DAQ network ROS PC's are directly connected to the core with four 10GbE links [39].

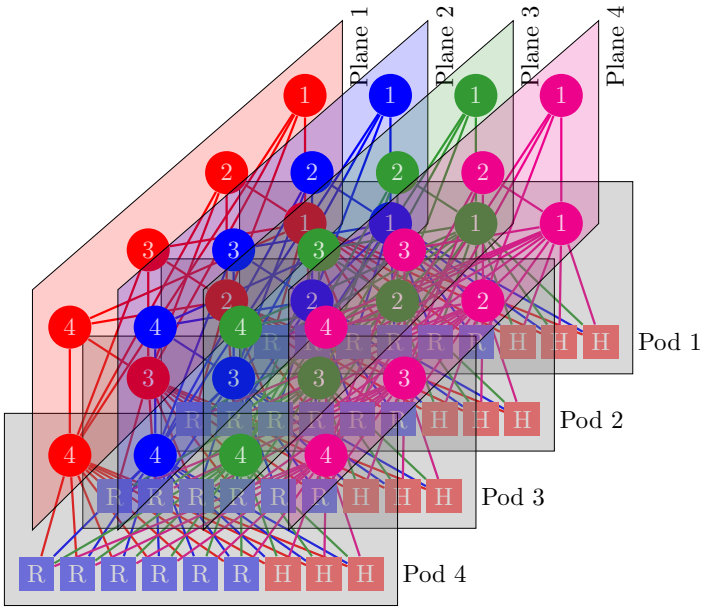


Figure 6.2: Architecture of a DAQ network in the parallel leaf-spine topology.

We call this topology a *parallel leaf-spine topology* because the planes provide independent connectivity.

6.2.2 *Flow optimisation and bandwidth scaling*

From the viewpoint of a DAQ system, it is important to estimate what is the offered bandwidth of the proposed topology. Normally, the bisection bandwidth[2] of a given network is given as an indicator, but in our case we can calculate the maximum bandwidth of this topology for the traffic pattern typical for DAQ. For this purpose we need to exploit the multiple network paths that are available between ROSes and DCMs. We can profit from the fact that we possess the knowledge about the entire network and the specific traffic pattern.

The most obvious approach to maximise the network's performance is to assign all flows of a particular DCM to use a single plane and a single spine switch in that plane. Two examples are given in Figure 6.3:

1. One of the DCMs of the HLT rack 2 in pod 3 is assigned to spine switch 1 in plane 4. All flows from ROS to this DCM are programmed in such a way that all packets go first to the leaf switches in plane 4 and then to spine switch 1 in the same plane. They finally reach the destination pod 3 and HLT rack 2, where this DCM is located.

2. One of the DCMs of the HLT rack 1 in pod 4 is assigned to spine switch 3 in plane 2. All flows from ROS to this DCM are

2 Bisection bandwidth is defined as the maximum capacity between any two servers [170].

programmed in such a way that all packets go first to the leaf switches in plane 2 and then to spine switch 3 in the same plane. They finally reach the destination pod 4 and HLT rack 1, where this DCM is located.

All DCMs can be then spread across available planes and spine switches, so that the traffic is equally distributed. This is under the assumption that all the links have the same bandwidth, and all the nodes are symmetrically connected to every plane. The pseudo-code is presented in Algorithm 6.1. This approach can be considered as an example of the waterfilling algorithm, well-known in communication [129], as it allocates new flows to the least loaded spine switch. In the next section we will show how this can be achieved in real network using the OpenFlow protocol. An example implementation is available in [43]. In more complex situations the round-robin allocation of spine switches can be replaced with a full waterfilling-based solution.

Algorithm 6.1 General algorithm to distribute DAQ flows across the parallel leaf-spine fabrics.

```
spines ← list of spine switches in the topology
nspines ← number of spine switches in the topology
dcm ← the first data collector in the topology
i ← 0
while dcm exists do                          ▷ iterate over all data collectors
    spine ← spines[i mod nspines]            ▷ use one spine switch for each
    INSTALL_FLOWS(dcm,spine)
    i ← i + 1
    dcm ← the next data collector in the topology
end while
```

In the following we will derive the theoretical bandwidth of such a DAQ network under these assumptions. The DAQ bandwidth can be optimised by assigning flows to particular paths depending on the available capacity. As we will see, for a particular DAQ configuration, the topology can be easily adapted to reach the maximum performance that is defined solely by the available bandwidth at the input and output of a DAQ network, and not the network itself (see equation (A.7)).

For full event building, the total bandwidth available for DAQ data flows can be calculated with

$$B = N_P \cdot N_H \cdot N_R \cdot b_{RH} \,,$$

where N_P is the number of planes, N_H is the total number of HLT racks, N_R is the total number of readout nodes, and b_{RH} is the bandwidth of a single ROS-to-HLT flow. Bandwidth of this single flow is limited either by the bandwidth of the leaf-to-HLT, ROS-to-leaf, or leaf-to-spine link, which can be expressed with

$$b_{RH} = \min\left(\frac{b}{N_R}, \frac{b}{N_H}, b_{RH_{interpod}}\right) . \tag{6.1}$$

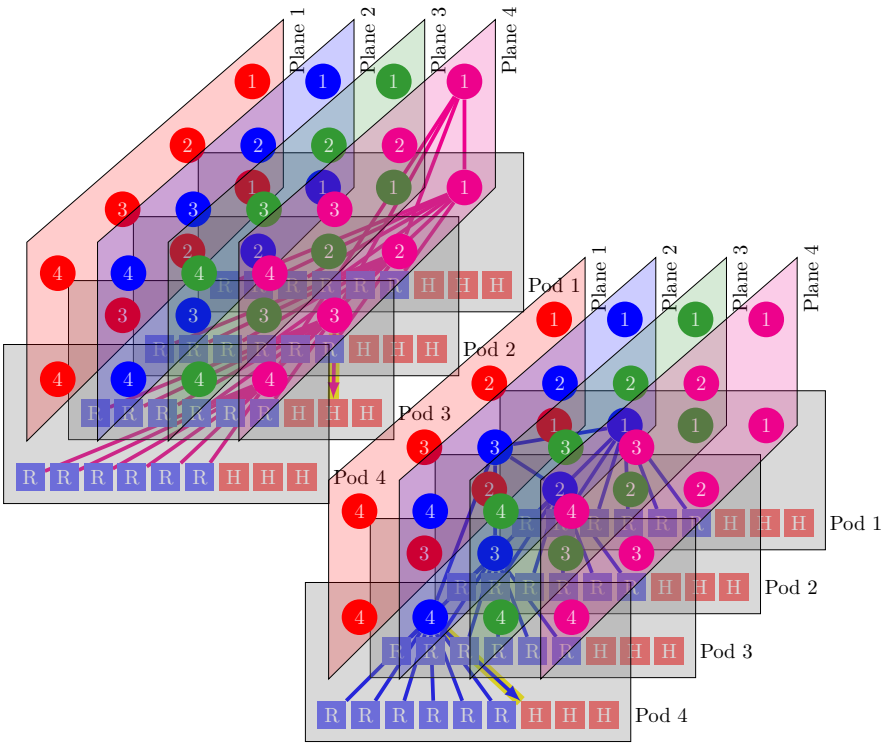


Figure 6.3: The method for load balancing by assigning a particular plane and spine switch for all flows belonging to a DCM. Two examples are presented: a DCM in the second HLT rack in pod 3 is assigned to spine switch 1 in plane 4 and a DCM in the first HLT rack in pod 4 is assigned to spine switch 3 in plane 2.

The base bandwidth of a single link is denoted with b. For the flows traversing the spine layer, the bandwidth is limited by the total number of flows that use the same plane. Note that flows from ROSes in the same pod as a DCM do not traverse the spine layer:

$$b_{RH_{interpod}} = \frac{b \cdot N_S}{N_{R_{pod}} \cdot N_{H_{pod}} \cdot (N_{pod} - 1)} \,. \tag{6.2}$$

The numerator is the total bandwidth in/out of a pod through the spine layer, with N_S as the number of spine switches in a plane. The denominator is the number of inter-pod flows that traverse the spine layer. $N_{R_{pod}}$ as the number of readout nodes in a pod, $N_{H_{pod}}$ as the number of HLT racks in a pod, and N_{pod} denoting the total number of pods. The total number of ROSes and HLTs is given by

$$\begin{aligned} N_R &= N_{R_{pod}} \cdot N_{pod} \\ N_H &= N_{H_{pod}} \cdot N_{pod} \,. \end{aligned} \tag{6.3}$$

These simple equations can give a quick estimate of the offered DAQ bandwidth for various configurations. They can be also used to derive the minimum number of spine switches in order to guarantee the requested DAQ bandwidth in a given configuration.

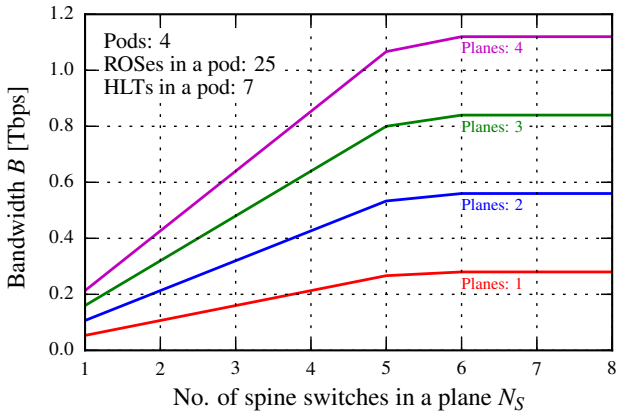


Figure 6.4: An example for bandwidth scaling for a DAQ network (as defined in Section 3.2) in the parallel leaf-spine topology.

An example for a DAQ system with 100 readout nodes, 28 HLT racks, and 10 Gbps of base link bandwidth is given in Figure 6.4. This design allows for flexible adjustment of the network to the requirements by changing the number of spine switches, planes or even the number of nodes in a pod.

In order to ensure that the only limiting factors are the input or output links, not the network itself, $b_{RH_{interpod}}$ from equation (6.1) has to fulfil the condition

$$b_{RH_{interpod}} \geq \frac{b}{N_R}\,,\ b_{RH_{interpod}} \geq \frac{b}{N_H}\,. \tag{6.4}$$

Using equation (6.2) and (6.3), $b_{RH_{interpod}}$ can be written as

$$\begin{aligned} b_{RH_{interpod}} &= \frac{b \cdot N_S}{\frac{N_R}{N_{pod}} \cdot \frac{N_H}{N_{pod}} \cdot (N_{pod}-1)} \\ &= \frac{b \cdot N_S \cdot {N_{pod}}^2}{N_R \cdot N_H \cdot (N_{pod}-1)}\,. \end{aligned}$$

so condition (6.4) becomes

$$\frac{b \cdot N_S \cdot {N_{pod}}^2}{N_R \cdot N_H \cdot (N_{pod}-1)} \geq \frac{b}{N_R} \qquad \frac{b \cdot N_S \cdot {N_{pod}}^2}{N_R \cdot N_H \cdot (N_{pod}-1)} \geq \frac{b}{N_H}$$

$$\frac{N_S \cdot {N_{pod}}^2}{(N_{pod}-1)} \geq N_H \qquad \frac{N_S \cdot {N_{pod}}^2}{(N_{pod}-1)} \geq N_R$$

$$N_S \geq N_H \frac{N_{pod}-1}{{N_{pod}}^2} \qquad N_S \geq N_R \frac{N_{pod}-1}{{N_{pod}}^2}$$

or

$$N_S \geq N_{H_{pod}} \frac{N_{pod}-1}{N_{pod}}\,,\ N_S \geq N_{R_{pod}} \frac{N_{pod}-1}{N_{pod}}\,, \tag{6.5}$$

when taking into account per pod counts again, as in equation (6.3). Equation (6.5) can be used to calculate the required number of switches

in each spine plane in order to achieve the maximum network bandwidth as results from the available bandwidth at the ROS and DCM sides. What is also interesting, network utilisation is independent of the number of parallel leaf-spine planes (assuming that all ROS nodes and HLT racks are connected to each plane).

This is also confirmed in Figure 6.4. Independently of the number of planes used, all plots saturate for six spine switches per plane. In this configuration the network reaches its maximum performance, defined by the available input and output bandwidth. If fewer switches in the spine stage are used, the ROS and HLT links remain underutilised as there is not enough bandwidth available in the leaf-spine fabrics. On the contrary, when more than five spine switches are used, the fabrics is underutilised because the HLT racks do not provide enough bandwidth:

$$N_{H_{pod}} \frac{N_{pod}-1}{N_{pod}} = 7\frac{4-1}{4} = 5.25\,.$$

In all cases the ROS links are underutilised:

$$N_{R_{pod}} \frac{N_{pod}-1}{N_{pod}} = 25\frac{4-1}{4} = 18.75\,,$$

which require at least 19 spine switches in each plane in this configuration. This can be seen, however, as means to provide resilience to link failures, if fewer switches are used. We will discuss the failover mechanism in Section 6.2.4.

6.2.3 *Flow assignment and packet routing*

In the previous section we showed how the bandwidth of a DAQ network, which is based on the proposed parallel leaf-spine architecture, can be maximised. In our derivations, it was assumed that all paths throughout the network are statically assigned to particular ROS-DCM flows.

This stands in contrast to the popular load balancing methods. Traditional approaches include LAGs (IEEE 802.3ad [76]) and ECMP (IEEE 802.1Qbp [75]) that use hashing to distribute packets across available links. These algorithms cannot guarantee though that the traffic is equally distributed across available paths. This depends on the hash functions used and the flows to be hashed. It can provide the best performance when the traffic is not known a priori or it is highly dynamic, as in datacenters. In DAQ, the workloads are known a priori, so deterministic load balancing is possible in DAQ networks and we can understand its performance as shown in the previous section. A similar approach was taken in [29, 116]. Furthermore, each single flow is always assigned to the same path, so packet reordering is not possible, which otherwise could decrease the performance of the end-nodes.

The OpenFlow protocol is the perfect means to achieve this static flow assignment. We use this protocol to monitor and optimally distribute the traffic across available paths. OpenFlow is used between a

centralised network controller and the switching nodes in the software-defined network. We described SDN and OpenFlow in Section 2.5 in more detail. The controller uses OpenFlow to install packet-handling rules on OpenFlow-enabled switches. Specific actions, like forwarding to some egress port, are applied to packets belonging to particular flows. The flows can be defined by a number of different protocol fields, like source/destination MAC addresses, IP addresses, TCP ports, etc.

In our design, a set of default rules is first installed throughout the leaf-spine fabric to provide basic connectivity through the network. The controller then uses the OVSDB protocol to instruct the switches to create a single daqring device (see Section 5.5) for every DCM that is identified in the system. Then it installs a set of OpenFlow rules on the switches, so that they are programmed to move specific packets from the ingress ports to appropriate daqrings and later to dequeue them to the egress ports, the same way as we described already in Section 5.5.1.1. For our evaluation in Section 6.3, we are emulating multiple DCMs on a single host, so we are using therefore a set of rules based on the 4-tuple (source/destination IP, source/destination TCP port) for every ROS-to-DCM flow. The controller uses the algorithm presented in Section 6.2.2 and assigns the optimum egress port for the parallel leaf-spine topology. In a typical DAQ network though, it is enough to base all flows solely on destination IP addresses. Moreover, the switches do not need to use layer 2 MAC addresses to forward packets.

Furthermore, the end-nodes that connect to the network with multiple interfaces, like the ROSes, can be also treated as switching nodes. An instance of OVS is, therefore, run on these nodes as well. The SDN controller can then assign the flows outgoing from some ROS to the appropriate plane. In this way, the entire network is programmed by the controller and the OVSDB and OF protocols are the only protocols in use. The Address Resolution Protocol (ARP) is not needed as every host is configured to represent a distinct network (*/32 addressing*). This approach differentiates us from the original design in [7], using traditional network protocols with ECMP routing and flow-based hashing. OVS at the end-nodes was also used by Al-Najjar, Layeghy, and Portmann in [116], but with additional means for ARP handling. This concept is expanded by Ballani et al. in [15], who proposed a general framework to implement network functions at the end-hosts.

6.2.4 *Resilience*

Constant monitoring of path status throughout the network can be implemented at the SDN controller to provide fault tolerance. As can be seen from Figure 6.2 and Figure 6.4 failure of a single switching node or a link does not have a significant impact on the overall system performance because of the large number of independent paths available

between the ROS and DCM nodes. Furthermore, as it was indicated in Section 6.2.2, multiple links from the ROS to the parallel planes of the DAQ network can be used to increase the system's redundancy.

Path status has to be monitored, though, in order to trigger redistribution of the flows according to the algorithm presented in Section 6.2.2. The entire process at the controller is given by Algorithm 6.2 and can be summarised in three steps:

1. Detection of link or interface failure.
2. Recalculation of the flow assignment throughout the network.
3. Installation of the new flows at the switching nodes.

Algorithm 6.2 Example algorithm to redistribute data acquisition flows across the parallel leaf-spine fabrics after link failure.

```
spines ← list of spine switches in the topology
nspines ← number of spine switches in the topology
link_state_changed ← true
while true do
    if link_state_changed then          ▷ recalculate flow assignment
        i ← −1
        dcm ← the first data collector in the topology
        while dcm exists do
            repeat
                i ← i + 1
                spine ← spines [i mod nspines]
            until IS_PATH_UP(dcm,spine)
            INSTALL_FLOWS(dcm,spine)          ▷ install new flow
            dcm ← the next data collector in the topology
        end while
        reinstall ← false
    end if
    link_state_changed ← CHECK_LINKS      ▷ detect link changes
end while
```

For the time it takes to perform these steps, the data packets can be still being sent over faulty paths, which degrades the performance. For this reason, it is important to optimise the entire process. The reallocation process will be faster, if there are fewer flows, so one of the most effective means is to reduce the overall number of flows that are installed in the network. In the typical case, as it was described in the previous section, it is enough to use flows based on the IP address of each DCM only. In this case, it becomes straightforward to reinstall the flows belonging to the affected DCMs. In Section 2.5 we pointed to the literature on the techniques of providing resilience in SDN networks. In Section 6.3, in turn, we will perform initial evaluation in a prototype DAQ network. An example implementation is available in [43].

In SDN one more aspect has to be taken into account. As we explained already in Section 2.5, reliability of the control plane is critically important. A single controller is a single point of failure and can bring down the entire network, if a default behaviour of the switching nodes is not defined. Backup controllers can be used to recover from these failures. Section 2.5 also contains references to the relevant work in this area.

6.2.5 *Cost comparison*

In the introduction to this chapter (see Section 6.1) it was remarked that the total cost of building a DAQ network is one of the key factors when considering a technology. As a matter of fact, this cost also includes the cost of developing better traffic shaping techniques in the given configuration, the cost of the inefficiencies introduced by the network congestion, and the cost of operating a network. The latter includes also the cost of hiring or training experts for the technology of choice. These aspects were discussed in Section 2.1.1 and Section 3.3.

In this section we focus on the cost advantage when using the parallel leaf-spine topology with software switches instead of the traditional approach with expensive telecom-class routers in the core of a network (see Section 4.2). Since both solutions are based on Ethernet, it can be assumed that the total cost of the network hardware is the main differentiator.

6.2.5.1 *Cost estimation*

In case of the parallel leaf-spine topology, subsequent leaf and planes are added incrementally (starting with a single plane with one leaf switch) in order to provide the required port count. The number of spine switches is predefined and determines the oversubscription[3] factor at the leafs. In this case, it is the ratio of the number of end-nodes to the number of spine switches connected to a single leaf switch.

The oversubscription value of 1:1 is not required in order to provide full performance for the specific traffic pattern. For example, in Section 6.2.2, we showed that only six spine switches are needed, if there are seven HLT racks and 25 ROS nodes in each pod. It means that even an oversubscription of approximately 5:1 ((7+25):6) is not going to reduce the performance of data acquisition.

The overall cost of the parallel leaf-spine network based on software switches is determined by the total number of servers used to build

3 Oversubscription can be explained as the ratio of the worst-case achievable aggregate bandwidth among the end-hosts to the total bisection bandwidth of a particular network. For example, an oversubscription of 1:1 indicates that all hosts may potentially communicate with arbitrary other hosts at the full bandwidth of their network interface, whereas a factor of 5:1 means that only 20 % of available host bandwidth is available for some communication patterns [56].

this network. We assume here that the same server is used for all the switches, which is the Supermicro SuperServer 6037R-TXRF system with ten PCIe 3.0 slots [155]. This server can be equipped, for example, with two E5-2697 v2 Intel Twelve-Core Xeon 2.7 GHz processors and ten quad-port 10GbE network adapters [81]. In this configuration it should be possible to provide a total of 40 10GbE ports and a bandwidth of 400 Gbps with a single software switch. The prices are catalogue prices from [27, 77]. The total cost of a single server-switch is approximately £11 000.

The required number of servers $N_{servers}$ to provide N_{ports} ports to connect to the network is given by the total number of leaf switches $N_{L_{total}}$ and spine switches $N_{S_{total}}$. The number of servers is therefore given by

$$N_{servers} = N_{L_{total}} + N_{S_{total}} \,. \tag{6.6}$$

The number of spine switches N_S in each plane is predetermined and defines the oversubscription factor. Another predefined value is the number of network ports available on every server-switch $N_{ports_{server}}$.

In order to fulfil the requirement a number of parallel leaf-spine planes is required. Each plane can offer no more than

$$N_{ports_P} = N_{pods} \cdot N_{ports_{pod}}$$

ports to connect the end-nodes. N_{pods} is the maximum number of pods per plane and $N_{ports_{pod}}$ is the number of available ports in each pod. Since each leaf switch in a plane is connected to every spine switch in this plane (see Figure 6.2), it is given by

$$N_{ports_{pod}} = N_{ports_{server}} - N_S \,.$$

A single leaf switch connects also to a single pod, which allows us to calculate the oversubscription factor at the leafs, $N_{ports_{pod}} : N_S$, as $(N_{ports_{server}} - N_S) : N_S$. Also, the maximum number of pods in a plane is therefore equal to the maximum number of leaf switches in a plane N_L. Since each spine switch has to be connected to every leaf switch in a plane, the maximum number of leaf switches in a plane N_L is determined by the number of ports available on a single switch. The maximum number of pods in a plane can be therefore expressed as

$$N_{pods} = N_L = N_{ports_{server}}.$$

The number of fully filled leaf-spine planes is then calculated with

$$\begin{aligned} N_{P_{full}} &= \left\lfloor \frac{N_{ports}}{N_{ports_P}} \right\rfloor \\ &= \left\lfloor \frac{N_{ports}}{N_{pods} \cdot N_{ports_{pod}}} \right\rfloor \\ &= \left\lfloor \frac{N_{ports}}{N_{ports_{server}} \cdot (N_{ports_{server}} - N_S)} \right\rfloor \,. \end{aligned} \tag{6.7}$$

Another plane provides the remaining ports. They are provided by a subset of $N_{L_{rem}}$ leaf switches in this plane, but the number of spine switches remains the same as the predefined value N_S. The number of leaf switches is equal to the number of pods required to provide the remaining ports $N_{P_{rem}}$ and can be calculated with

$$\begin{aligned} N_{L_{rem}} &= \left\lceil \frac{N_{ports_{rem}}}{N_{ports_{pod}}} \right\rceil \\ &= \left\lceil \frac{N_{ports_{rem}}}{N_{ports_{server}} - N_S} \right\rceil , \end{aligned} \tag{6.8}$$

where the number of remaining ports is given by

$$\begin{aligned} N_{ports_{rem}} &= N_{ports} - N_{P_{full}} \cdot N_{ports_P} \\ &= N_{ports} - N_{P_{full}} \cdot N_{ports_{server}} \cdot (N_{ports_{server}} - N_S) . \end{aligned} \tag{6.9}$$

Equation (6.6) can be now expressed as

$$\begin{aligned} N_{servers} &= N_{L_{total}} + N_{S_{total}} \\ &= N_{L_{rem}} + N_S + N_{P_{full}} (N_L + N_S) \\ &= N_{L_{rem}} + N_S + N_{P_{full}} (N_{ports_{server}} + N_S) \\ &= N_{L_{rem}} + N_S \left(N_{P_{full}} + 1 \right) + N_{ports_{server}} , \end{aligned} \tag{6.10}$$

considering the servers used to build the planes with maximum number of pods and the plane providing the remaining pods. In the end, equations (6.7) to (6.10) are sufficient to calculate the total number of servers required to build a parallel leaf-spine fabrics offering N_{ports} ports with N_S spine switches in each plane and $N_{ports_{server}}$ ports on each server-switch.

6.2.5.2 *The reference solution*

The current ATLAS DAQ network architecture (see Section 3.5.1) is used as a reference solution. We assume that at least one Brocade MLXe 32 [28] chassis is used. Subsequent chassis and 10GbE modules (24-port) are added incrementally when increasing the total number of available network ports. The ROSes and HLT racks are connected directly to the router, so the oversubscription cannot be altered and equals 1:1. The prices are catalogue prices from [106]. The cost of the chassis is £98 698, the single 24-port module is £51 948, and the single switch fabric module is £7399 (seven modules are included in the cost of chassis). The maximum number of ports in a single chassis is 768. The required number of chassis $N_{chassis}$, modules $N_{modules}$, and fabric modules $N_{fmodules}$ can be calculated with

$$\begin{aligned} N_{chassis} &= \left\lceil \frac{N_{ports}}{768} \right\rceil \\ N_{modules} &= \left\lceil \frac{N_{ports}}{24} \right\rceil \\ N_{fmodules} &= N_{modules} . \end{aligned}$$

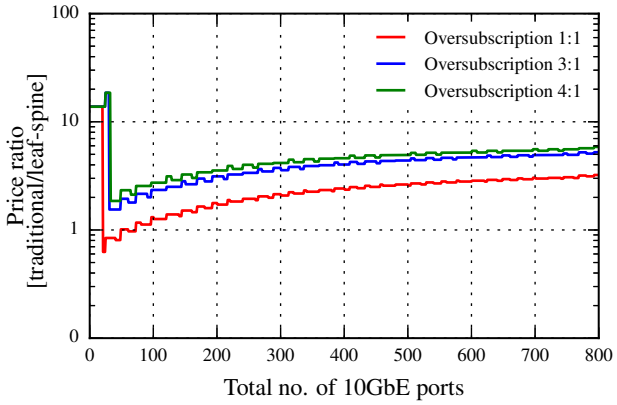


Figure 6.5: Costs comparison for building a DAQ network with the traditional approach (large routers in the core) and the parallel leaf-spine topology (software switches) with different oversubscription factors at the leaf switches. The cost of cabling is not included.

6.2.5.3 *Comparison*

Figure 6.5 shows that a DAQ network of the size of the ATLAS experiment using software switches is generally few times cheaper than the current solution. The advantage grows when increasing the oversubscription factor, which can be followed to some extent without any performance degradation in the DAQ use case, as explained in Section 6.2.2. For lower port counts, the ratio is significantly larger due to the high cost of the router chassis.

The only exception to this rule is for the oversubscription of 1:1 and lower port counts. In few configurations, the traditional approach can be cheaper. This is caused by the fact that a lot of spine switches are used to interconnect a small number of pods. This is not, however, a realistic configuration as the network would remain highly underutilised.

6.2.6 *Physical space requirements*

The physical space required to fit the entire networking hardware can be also an important factor when designing a DAQ network. This is particularly important in space-constrained environments, example of which are the LHC experiments. For this reason, we provide an estimate on the physical dimensions of a DAQ network based on software switches.

In Section 6.2.5 we compared the costs of building a network based on the parallel leaf-spine topology based on server-switches to the costs of a reference solution, following the current ATLAS DAQ network. Similar methodology can be used to compare the space requirements of both solutions. Figure 6.6 shows the ratio of the physical area required by the reference solution with the Brocade core routers to the area required by the parallel leaf-spine topology based on server-switches. For the

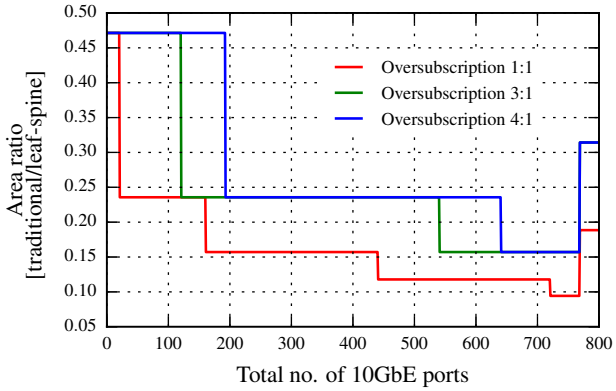


Figure 6.6: Comparison of the physical space required (total area) for building a DAQ network with the traditional approach (large routers in the core) and the parallel leaf-spine topology (software switches) with different oversubscription factors at the leaf switches.

former, the area is given by the dimensions of the Brocade MLXe 32 chassis [28]. For the latter, the dimensions of racks required to fit all the server-switches determine the total physical area needed. We use a reference rack from [1].

In this case, the traditional approach has the advantage. For higher oversubscription factors, server-switches in the leaf-spine fabrics require two to four times more physical area. Although the ratio remains in single-digit range, this aspect has to be considered when designing the entire TDAQ system. In Chapter 7 we will propose a solution to improve this metric.

6.3 A PROTOTYPE OF AN SDN-BASED DAQ NETWORK

In order to evaluate the potential usage of the proposed design in building a centrally managed DAQ network with DAQ-optimised Open vSwitches we prepared an IP-only parallel leaf-spine network. Figure 6.7 gives an overview of this prototype.

6.3.1 *Evaluation setup*

SWITCHING NODES This network consists of eight software switches (see Figure 6.7a) running on separate physical servers. The servers acting as switches follow the same specification as in Section 5.3.1. We continue to use OVS optimisations described in Section 5.5. Pause frame mechanism remains disabled for all the following tests. The OpenDaylight's [167] REST API [60] is used to distribute the flows across available paths with the algorithms presented in Section 6.2.2 and Section 6.2.3. The only difference is that we need to use both IP addresses and TCP port numbers in order to distinguish the DCMs. This comes from the fact that we emulate small HLT racks on each physical edge

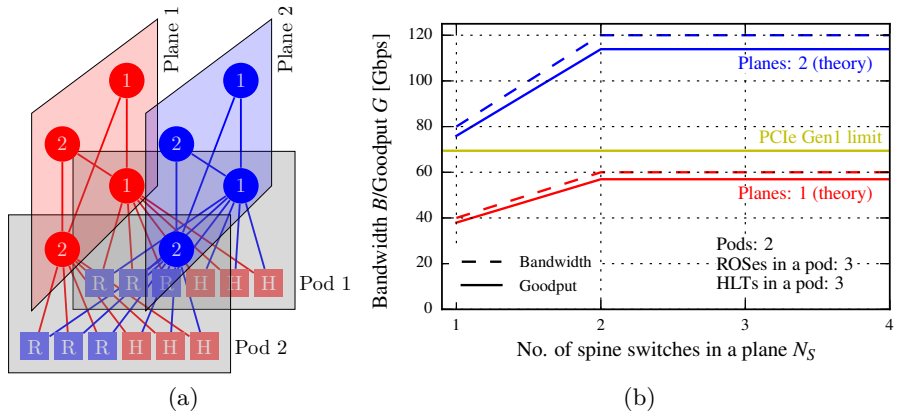


Figure 6.7: The prototype of the parallel leaf-spine topology (a) and its theoretical performance for DAQ traffic pattern (b).

host, so the DCMs running on the same host share the IP address. All flows are therefore based on IP addresses and TCP port numbers. Layer 2 MAC addresses are not used to forward packets at any switching node.

EDGE HOSTS Six ROSes and six HLT nodes run on separate physical hosts (twelve total). Eight DCMs run on each host to emulate a small HLT rack. Each host connects to two planes, and an OVS instance is run on every host, so that the network controller can assign the flows to the appropriate plane. Each ROS provides a single event fragment of 128 KiB, so the total event size is 768 KiB. The theoretical bandwidth for the DAQ-specific traffic pattern, calculated with the formulas derived in Section 6.2.2, is given in Figure 6.7b. This plot also includes the values converted to the theoretical goodput, using the equations from Appendix A.2. Note, our edge hosts in this testbed use PCIe gen1. Consequently, we know in advance that we will not be able to reach the network capacity. However, we will be able to use this as an opportunity to show the advantage of daqrings to shape traffic to suit limited edge devices (see Section 5.4.1.3 for details). The estimated upper limit that can be reached is also drawn in Figure 6.7b. This is estimated as the sum of the theoretical effective data rate available on the PCIe gen1 interfaces connecting the 10GbE network ports on the edge hosts [65]. It is then recalculated into the DAQ theoretical goodput with equation (A.12).

NETWORK The entire network, including switching nodes and edge hosts, is programmed by the OpenDaylight controller. The OVSDB and OF protocols are the only protocols in use. ARP is not needed as every host is configured to represent a distinct network (*/32 addressing*). This follows our generalised design that was presented in more detail in Section 6.2.3.

6.3.2 *Evaluation results*

As throughout the thesis, the goal of our designs and optimisations is a lossless network for DAQ that provides optimum throughput for the specified traffic pattern. The event rate remains unlimited for the following evaluation, which means that we measure the saturation goodput.

6.3.2.1 *Daqrings tuning*

As we noted already in Section 6.3.1, it is not possible to reach the theoretical performance with our prototype DAQ topology because of the edge hosts. The bandwidth is limited by PCIe gen1 bus used in the hosts emulating ROS and HLT racks, which is not enough for the dual-port 10GbE network interfaces. Furthermore, their on-chip buffers are small, so they are easily overflown with bursty many-to-one traffic. This situation resembles, to some extent, the real-world ATLAS configuration. The ToR switches have 10GbE uplinks to the core, but only 1GbE to the edge hosts (see Section 3.5.1). This allows us to evaluate the use of per-daqring traffic shaping.

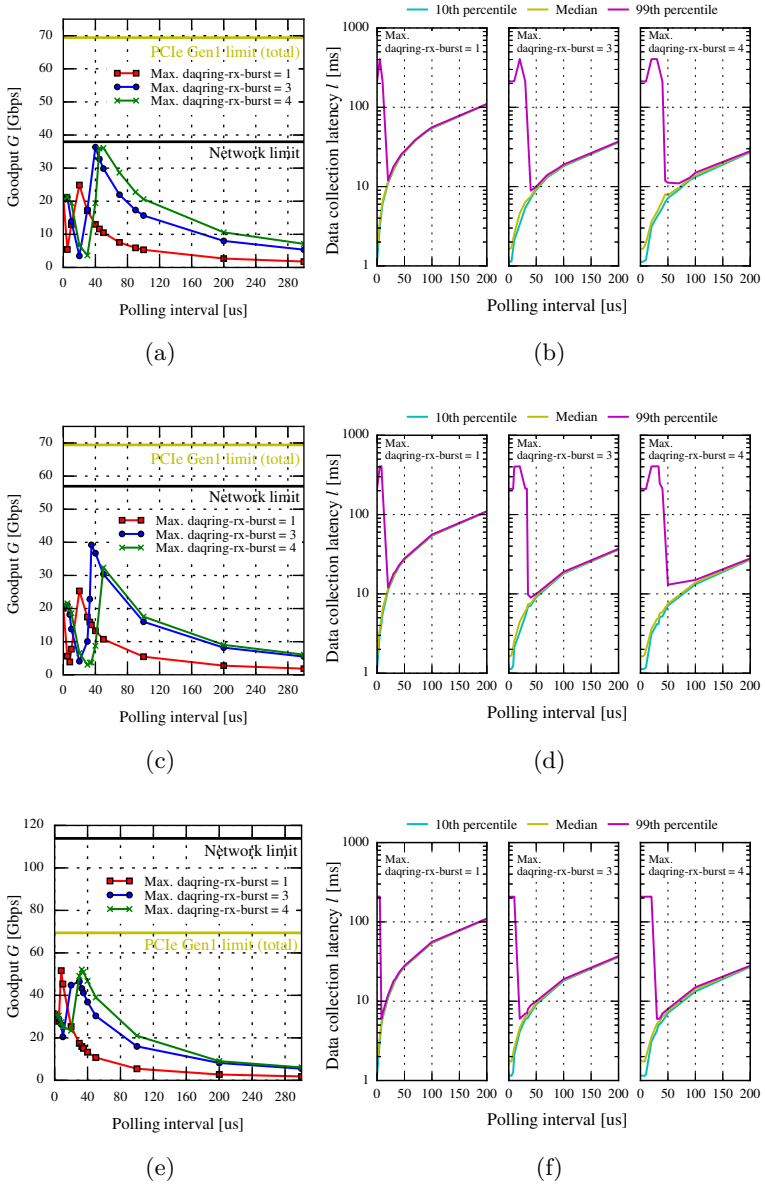
For every daqring device we use the OVSDB protocol to limit the number of packets (*burst size*) than can be dequeued from this ring in a single polling cycle as already explained in Section 5.4.1.3 and Section 5.5.1. Both the burst size and the polling cycle can be tuned for every configuration of the parallel leaf-spine fabrics, as illustrated in Figure 6.8. Careful fine-tuning gives us the best combination of those values, which eliminates packet drops (lossless operation) and maximises the goodput by adapting to the particular limitations of the PCIe/NIC combination. The larger the maximum burst size, the longer polling interval is required. This is natural as the ratio of burst size and polling interval translates directly into a specific rate limit.

Specifically, every combination of the maximum burst $daqring_{burst}$ within a polling cycle $daqring_{polling}$ can be also expressed as shaping the outgoing packet streams towards each DCM as not to exceed a specific rate limit $daqring_{limit}$. For the largest Ethernet frames of 1518 B (including the 4 B for the checksum and 14 B for the Ethernet header) [55], it can be estimated as

$$daqring_{limit} = \frac{daqring_{burst} \cdot 1518\,\mathrm{B}}{daqring_{polling}}\,.$$

For our configurations, the optimum settings are summarised in Figure 6.8g. This table contains also the maximum total rate towards an HLT node, considering that there are eight DCMs emulated on a single host.

For the full topology with two planes and two spine switches in each of them, this total limit is 11.8 Gbps. This matches the theoretical effective data rate of PCIe gen1 (that is shared by two 10GbE ports on each host) of 12.19 Gbps [65]. What can be seen in Figure 6.8e and



Configuration	Burst	Polling [µs]	Limit [Gbps]	Towards an HLT node [Gbps]
1-plane-1-spine	3	40	0.91	7.3
1-plane-2-spines	3	35	1.04	8.3
2-planes-2-spines	4	33	1.47	11.8

(g)

Figure 6.8: Tuning traffic shaping at the daqring queues for the setup depicted in Figure 6.7a, using one plane with one spine switch (a) and (b), using one plane with two spine switches (c) and (d), and using two planes with two spine switches (e) and (f). Optimum settings for each configuration are summarised in (g).

Figure 6.8f, for the optimum traffic shaping configurations, the goodput is maximised and the data collection latency is minimal without indications of packet drops. All percentiles are below 6 ms, while the serialisation delay of the entire event and protocol headers at 1.472 Gbps is 4.5 ms.

Similar observations can be made for the incomplete configurations (one plane with two spines in Figure 6.8c and Figure 6.8d or one plane with one spine in Figure 6.8a and Figure 6.8b). Data collection latencies are slightly higher, because a lower rate is used.

6.3.2.2 *Goodput and latency*

Using the optimal settings for each configuration, we focus now on the goodput and latency characteristics in more detail. They are presented in Figure 6.9 (*OVS+rate_limited_daqrings*). As a reference, results for the cases with non-limited daqrings (*OVS+daqrings-CC*) and using TCP Cubic instead of daqrings (*OVS+TCP_Cubic*) are also plotted.

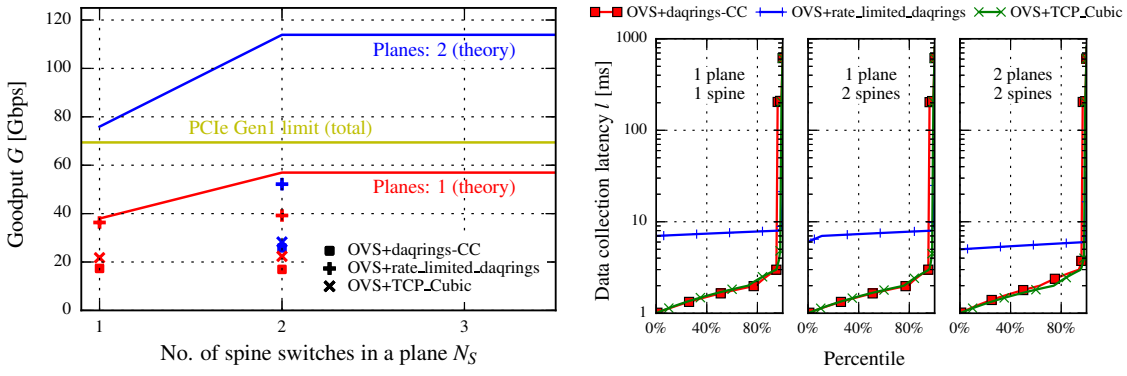
1-PLANE-1-SPINE OVS with rate-limited daqrings reaches the theoretical goodput without any signs of TCP timeouts in the latency distribution. Without rate-limitation, timeouts occur and goodput is significantly lower. The same occurs for the configuration without daqrings, but with default TCP Cubic congestion control.

2-PLANES-1-SPINE Goodput for rate-limited daqrings increases slightly, but remains lower than the theoretical value in this case. The other configurations perform similarly to one plane. The reason is that we are still using only a single port of the dual-port interface on every host, which is the bottleneck in this configuration.

2-PLANES-2-SPINES Goodput of all configurations improves. All possible paths through the parallel leaf-spine network are used. OVS with rate limitation gives the best goodput (about 46 % of the theoretical limit of the fabric and 75 % of the theoretical PCIe gen1 limit). No packet drops are observed. Also the data collection latency is lower than with *1-plane-1-spine* because the DCMs of the same HLT node are now distributed across independent paths.

6.3.2.3 *Failover*

To conclude our evaluation of the prototype parallel-leaf spine fabric, we conduct an initial study on the failover mechanism. We analyse a case when just a single HLT node with eight DCMs is active and all fabric switches are enabled. Figure 6.10 shows how the goodput and the number of retransmitted TCP segments change over time, when a link between the leaf switch (connecting this particular HLT node) and a single spine switch breaks.



Configuration	G_{sat}	l_m [ms]	σ_l [ms]
1-plane-1-spine			
OVS+daqrings-CC	46.7 %	13.0	62.93
OVS+rate_limited_daqrings	95.7 %	7.6	0.59
OVS+TCP_Cubic	57.3 %	5.6	38.84
2-planes-1-spine			
OVS+daqrings-CC	22.3 %	13.1	63.99
OVS+rate_limited_daqrings	51.7 %	7.5	0.79
OVS+TCP_Cubic	29.4 %	5.4	38.06
2-planes-2-spines			
OVS+daqrings-CC	22.7 %	9.0	42.60
OVS+rate_limited_daqrings	45.8 %	5.5	0.19
OVS+TCP_Cubic	24.8 %	4.63	30.36

Figure 6.9: Comparison of three different approaches to incast congestion in the parallel leaf-spine topology (see Figure 6.7a). Saturation goodput (a) and the distribution of the event data collection latency (b) in function of the number of parallel planes and spine switches. The values for goodput in the table are relative to the theoretical limit of the given network configuration (not the PCIe limit).

At first, this HLT node performs near its theoretical performance, without any packet drops. Then, after around 120 s, a link to one of the spine switches breaks. It triggers the SDN controller to disable all flows related to daqrings in the fabric, so just the default path, using just one spine switch, is used. This results in low performance.

The SDN controller rescans now all the edge-hosts in order to identify all ROS-to-DCM flows with changed state. This is required as the DCMs may reinstate their TCP connections if no data are received for some period of time. The TCP port numbers change, so new flows have to be installed on the switches (we need to use both IP addresses and TCP number to distinguish individual DCMs, as described in Section 6.3.1). We did not optimise this process and the controller simply uses the SSH protocol to log into each HLT node and retrieve the ROS-DCM connections. With the updated list of connections, the controller

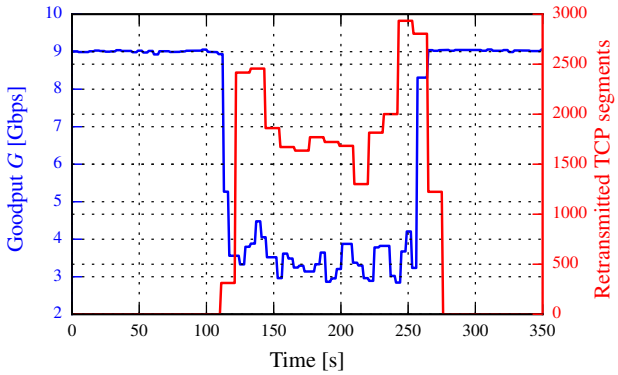


Figure 6.10: Simple failover scenario. A link in the topology from Figure 6.7a is broken around 120 s, and comes up again around 240 s.

can now distribute the flows across available paths using the same algorithm as before (see Section 6.2.2), excluding the broken path. Since only a single HLT is active all flows can be moved to other paths and operate at the same goodput again.

Unfortunately, the time it takes to rescan and reinstall the flows coincides with the time when the broken links becomes available again. Because of that we cannot distinguish these two events in Figure 6.10. After around 250 s from the start of the test, the HLT node operates at its full theoretical bandwidth again.

Although this evaluation cannot prove the effectiveness of the failover mechanism, we show basic principles how to achieve resilience and indicate important aspects that must be considered. First of all, scanning the entire system for ROS-DCM connections for each change in the network cannot be effective. In real world, it would be enough to use just the IP addresses of the data collectors for all flows, including redirections to daqrings. In this situation, there is no need to scan the DCMs at all, so the failover can be performed faster. Second, upon link failure detection, it is enough to redistribute the flows impacted by the failed link, so the performance degradation will be significantly lower. Further optimisation is possible, if backup flows are installed on each node in the network. In case of link failure, packets could be immediately redirected to the backup paths.

A general discussion on providing resilience in software-defined networks is available in [12].

6.4 CONCLUSION

In this chapter we showed how software switches, which we optimised for data acquisition in the previous chapter, can be connected together to build larger network topologies. Taking advantage of the optimised parallel leaf-spine topology these software switches prove to be a viable

alternative for the expensive feature-rich core routers in the future upgrades of the experiments at the LHC and other DAQ systems.

First, we presented how the leaf-spine topology, already popular in datacenters, can be adapted to the specifics of data acquisition. In particular, we concentrated on optimising the flow distribution across the fabric for the specific traffic pattern. We showed that it is feasible to build terabit data acquisition networks using dedicated software switches. This approach can offer cost and performance advantage over the traditional network designs, where expensive telecom-class devices are required. This comes, however, at the cost of slightly larger physical space required to accommodate the hardware. In the following chapter we will discuss, whether this could be alleviated with a new type of Ethernet devices.

Second, we proposed and evaluated on real hardware a method to manage and optimise the network using software-defined technologies, OpenFlow and OVSDB. We showed that the network can be centrally programmed using solely IP and TCP addressing. Using traffic shaping at the daqring ports, we demonstrated how the network can be tuned, in various configurations, to maximise the system's performance and reach lossless operation with high throughput and low latency. We concluded our evaluation with a preliminary discussion on providing resiliency.

Thus, we have shown that this design, including the optimised software switches, the adapted leaf-spine topology and the software-defined control plane, can be a viable solution for future data acquisition networks.

7 MULTI-HOST ETHERNET CONTROLLERS

Software switches were presented in the previous chapters as an alternative to the existing approaches to handle incast congestion in data acquisition. We have already shown performance of software switching is good with the possibility to scale to terabit networks. There is significant cost advantage over traditional solutions at the same time. However, we also see space can be an issue, as discussed in the previous chapter. Space usage can be reduced using a new class of Ethernet devices — multi-host Ethernet controllers. These devices combine a traditional network interface controller used on servers with an advanced Ethernet switch. However, since their performance characteristics are slightly different, we recheck in this chapter that performance is still sufficient. Furthermore, we explain how the shortcomings of software switching, which can prevent this approach from application in datacenter networks, could be overcome with multi-host Ethernet controllers. These devices provide advantages of both: flexible software switches with extreme buffering capabilities and high-performance traditional switch ASICs. This chapter contains results published in [90].

7.1 INTRODUCTION

An interesting alternative to using multiple network interfaces on commodity servers as software switches is replacing the former with a new class of Ethernet devices — *multi-host* Ethernet controllers. An example is Intel's FM10000 chip family [82], which offers up to 36 10GbE and four PCIe gen3 interfaces. As such it provides the features of both a traditional hardware switch, with a fast ASIC offering high bandwidth for packet forwarding, and a flexible software switch, with large, but slower memory, that can be run in parallel.

The advantages in approaching incast congestion in this way are twofold. First, better port density can be reached. As we have seen in Section 6.2.5, an example server can be equipped with 40 10GbE ports and requires three rack units. In contrast, a device based on the FM10000 chip can fit 72 10GbE interfaces with four servers in four rack units [3]. Second, the possibility to offload only some of the packet processing tasks to the dedicated switching ASIC can make our approach to incast congestion more suitable for datacenter networks. In Section 5.2.2 we pointed to the potential limitations of software

switches for general workloads, which normally do not apply to DAQ. If incast-sensitive flows were the only ones redirected to the software switch for buffering and traffic shaping, other flows would not be susceptible to increased latency, bufferbloat, or worse performance for small packets.

7.2 ADVANTAGES IN INCAST-AVOIDANCE

In the following we will discuss in more detail what are the potential advantages in incast-avoidance when using multi-host Ethernet controllers. We consider the potential improvements in port density, NUMA-tuning, and new possibilities for hardware acceleration.

7.2.1 *Towards higher port density*

In Section 6.2.6 we compared the physical space required when building a traditional DAQ network with large routers in the core and the proposed parallel leaf-spine topology based on optimised software switches. The former had an advantage, requiring two to four times less physical area.

In the following evaluation we consider the CSA-7400 computing platform for building software switches. This platform combines four compute nodes with dual Intel Xeon processors and two switch modules, the latter being the FM10840 multi-host Ethernet controllers [3]. Each compute node connects to every switching node with a PCIe (gen3) link. In the end, 72 10GbE interfaces with four servers require four rack units, whereas server-switches from Figure 7.1 occupy three rack units and provide 40 10GbE ports. Unfortunately, we do not consider how the costs change when using the CSA-7400 computing platform because the prices are not known at the moment.

Figure 7.1 shows that improvement in port density can be achieved, if server-switches with traditional NICs are replaced with devices using multi-host Ethernet controllers. Up to two times more ports can fit the same physical space. Although the density offered by traditional network designs (see Section 6.2.6) is still not reached, space utilisation is improved.

Furthermore, four dual Xeon processors, eight total, exceed by far the computational requirements for 72 10GbE ports. In Section 5.5.4, we showed that eight CPU cores are enough to provide full performance in a DAQ scenario with twelve 10GbE ports. Using a simple extrapolation, 48 cores would be required for two FM10840 devices. Assuming twelve-core processors, four Xeon CPUs out of eight available on the CSA-7400 platform should provide enough processing power. The remaining CPUs can be used for other tasks without limitations. This would not be constrained by resource sharing with the software switch, which we explained in Section 5.6.3, because they are physically sepa-

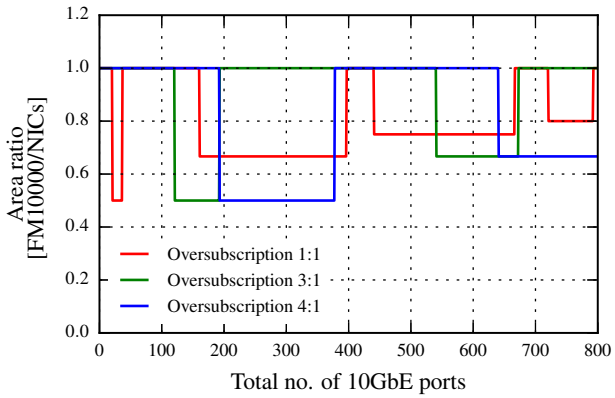


Figure 7.1: Comparison of the physical space usage (total area) for building a DAQ network based on software switches using traditional NICs or multi-host controllers for various oversubscription factors at the leaf switches.

rate processors. Thus, the area occupied by the network based on multi-host Ethernet controllers provides also significant computing resources for event data processing and filtering. As we will see in the following sections, they can be even increased, if hardware acceleration offered by these multi-host devices is used to further reduce the computational requirements of the software switch.

7.2.2 *Overcoming QPI limitations*

In Section 5.2.1 and Section 5.4.2.1 it was indicated that the QPI bus interconnecting CPUs on the same server platform can negatively impact the performance of a software switch. For this reason it is important to consider appropriate, NUMA-aware, allocation of NICs in the slots available on the server platform and affinity control of the CPU cores executing packet processing threads (see Section 5.2.2).

This considerations can be avoided with multi-host Ethernet controllers. When comparing the architecture presented in Figure 5.3b (using NICs) and Figure 7.2 (using multi-host controllers), it becomes clear that in case of the former each CPU has direct access over PCIe to a subset of network ports only. The remaining ports have to be accessed using the QPI bus, which requires careful performance tuning. For multi-host Ethernet controllers, there is no need to use QPI. Each CPU has direct access to the entire switching ASIC over its own PCIe lanes and, thus, can forward packets to every output port of this switch without the need to use QPI.

7.2.3 *Open vSwitch acceleration*

Multi-host controllers, being both a NIC and an Ethernet switch, open the possibility to offload some of the packet processing to the switching

ASIC. This can include, among others, applications like ARP handling or flow caching. The switching ASIC has the capabilities to define forwarding rules based on layer 2, layer 3, or layer 4 protocol headers [82]. Furthermore, in case of the DAQ-optimised software switch, only those packets that need to be buffered in dedicated queues, could be redirected to the software switch. Thus, the remaining traffic would not need to be processed by the CPUs and, in effect, reduce the number of the required processing CPU cycles.

In case of the CSA-7400 computing platform [3], hardware acceleration for Open vSwitch and OpenFlow is declared as a built-in feature, so the advantages are available for use without additional programming efforts.

7.2.4 *Application in datacenter networks*

As we already mentioned in the introduction to this chapter, the possibility to offload only some of the packet processing tasks to the dedicated switching ASIC can make our approach to incast congestion more suitable for datacenter networks. The generic performance of software switches can prevent them from application in many networks. In Section 5.2.2 we indicated that latency and throughput for small packets is often not enough to consider software switches with large buffers as a valid option for incast-avoidance in datacenters. Normally, those limitations do not apply to DAQ.

Multi-host controllers can adapt this solution to incast for the application in datacenter networks. The incast-sensitive flows should be the only ones redirected to the software switch for buffering and traffic shaping. All other flows can be then handled by the switching ASIC and, as a result, they would not be susceptible to increased latency, bufferbloat, or worse performance for small packets.

7.3 PERFORMANCE EVALUATION

In the previous section we showed that the physical space usage can be optimised when multi-host controllers are used instead of traditional NICs to build software switches. This advantage is particularly important in applications with constraints on the physical area. An example here are DAQ systems, like those at the LHC at CERN, where physical space is often limited on the experiment's site. This is not the only advantage of using multi-host controllers. As we explained in Section 7.2.4, the limitations of software switches could be largely avoided by offloading solely the incast-sensitive flows to the software switches for traffic shaping and buffering. The remaining packets can be switched by the traditional high-performance ASIC. In this section we evaluate the performance and discuss whether these devices are indeed suitable as a solution for incast-avoidance. We continue to use the ATLAS DAQ/HLT

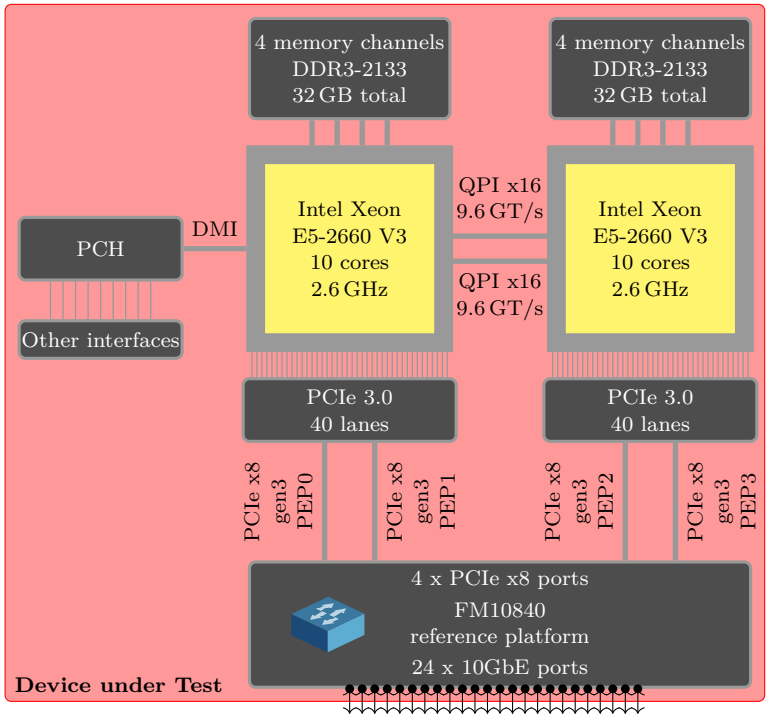


Figure 7.2: Block diagram of the device under test (DuT) that consists of a COTS server (Supermicro Superserver 7048R-TR) connected over four PCIe x8 links with the reference platform of the Intel FM10840 chip. It offers 24 10GbE ports.

software in emulation mode for the following evaluations. As a matter of fact, we consider the same or similar configurations, that were already evaluated in Chapter 4 and Chapter 5.

7.3.1 *Device under test*

In this chapter we evaluate a reference platform of the Intel FM10840 multi-host controller, see Figure 7.2. This platform offers 24 10GbE ports and four PCIe ports — *PEPs*. PEPs are connected with PCIe cables to four PCIe x8 gen3 slots of a COTS server. The maximum bandwidth on each PEP is 50 Gbps. The server is a Supermicro SuperServer 7048R-TR [155] with two Intel Xeon EP-2660 v3 (code name *Haswell*) ten-core CPUs. There is also a total of 32 GiB DDR3 memory installed (16 GiB per CPU socket). The operating system is 64-bit Fedora 20, kernel version 3.19.8-100. We configure the system to avoid the well-known bottlenecks, as described already in Section 5.2.2. This arrangement is used in the following configurations.

TRADITIONAL ETHERNET SWITCH (*peps:0*) Ethernet ports are the only ports used and switching is performed by the FM10840 chip. PEP ports are not used. This configuration is used as baseline (typical Ethernet switch). Congestion avoidance is realised with static TCP

congestion window (see Section 4.5.2.1), the pause frame mechanism of the switch and NICs (see Section 4.5.3), or with application layer traffic shaping, which limits the number of parallel data requests (see Section 4.5.1).

TRADITIONAL ETHERNET SWITCH WITH EXTRA SOFTWARE QUEUES (*peps:1/2/3/4*) Those queues, being the same *daqrings* introduced in Chapter 5, are used to accommodate many-to-one traffic bursts. In this configuration packets incoming on Ethernet ports that match some predefined rules are redirected to the PEP ports (one, two, three or four PCIe ports can be used) and handled by a software switch running on the COTS server. This is the same Open vSwitch 2.4.0 with custom patches as in Section 5.5, but with DPDK 2.2.0 that provides the user-space drivers for the FM10000 devices. This software switch is configured to enqueue the packets in those daqrings in order to avoid incast congestion. This approach provides large buffering capabilities in the DRAM memory of the host and rate limitation on a per daqring basis, so that a DAQ network is optimised and can provide lossless operation.

Our goal is to evaluate the performance of the second configuration, compare it to the performance of traditional switches under heavy incast congestion (first configuration), and demonstrate a new potential application of multi-host Ethernet controllers.

7.3.2 *Test configuration A*

This is the same configuration that was used to tune and evaluate the traffic shaping mechanism, static TCP congestion window, and pause frames in Section 4.5.4, where the details on this evaluation setup can be found. The left-hand side switch in Figure 4.14 was the DuT configured to operate as a traditional ToR switch (PEPs:0). Here, we extend this comparison and enable the offload of the incast-sensitive flows to the software switch over the PCIe ports (PEPs:3). There are twelve ROS nodes connected with 10GbE links to the DuT (120 Gbps total), so three PEPs are required (150 Gbps total) in order to guarantee that all traffic from ROS can be redirected to the software switch.

The approaches to incast that require tuning, namely application-layer traffic shaping and static TCP congestion window, were tuned already in Section 4.5.4. Before extending the comparison, we will also optimise the configuration of daqrings for the case when the optimised software switch is used for incast-avoidance.

DAQRINGS TUNING Figure 7.3 is used to tune the dedicated queues of the software switch running on the host connected to the DuT with three PCIe ports (PEPs:3). There is a single daqring for each of the 30 DCMs and each of these daqrings is rate-limited in order to maximise

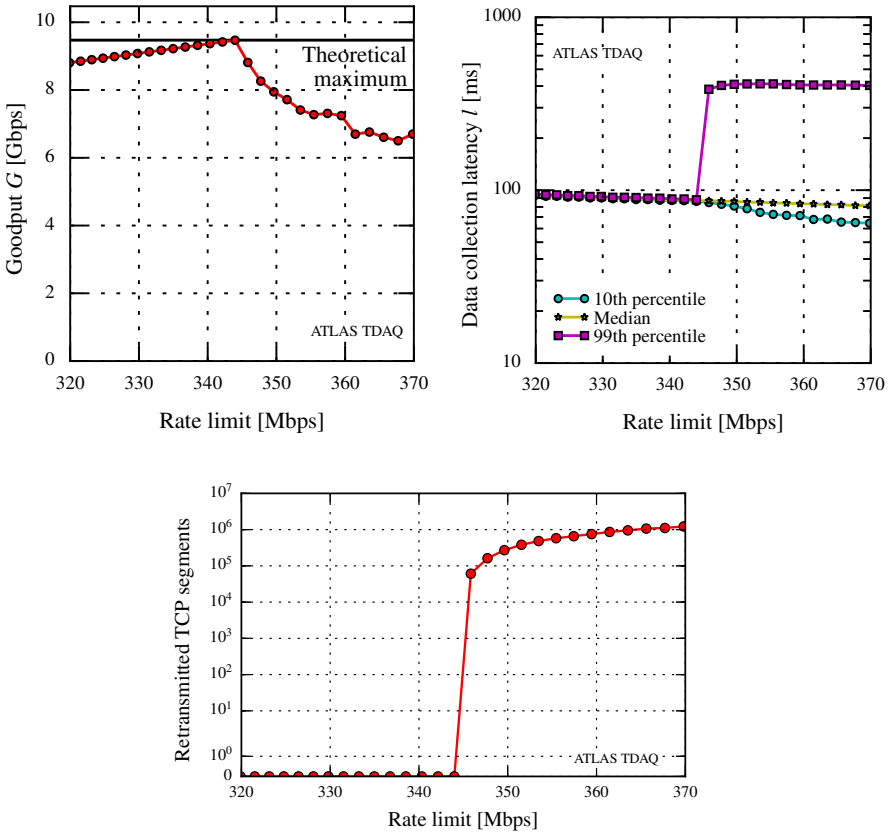
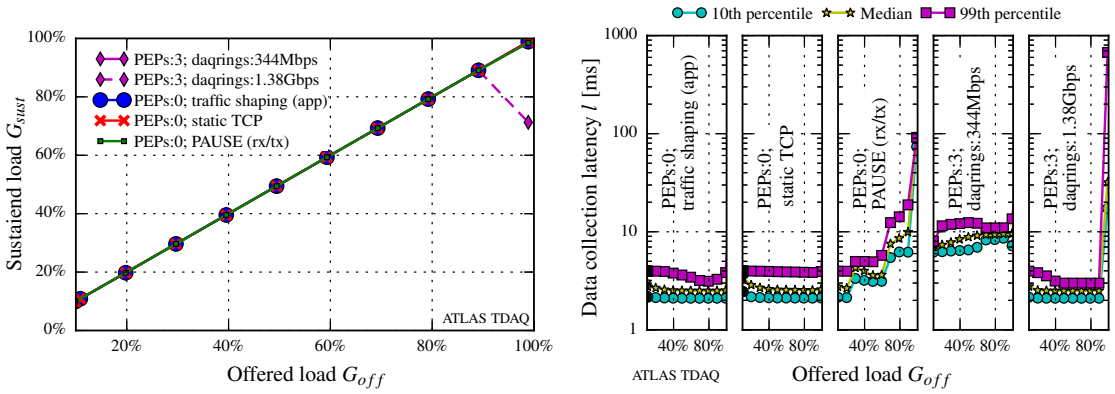
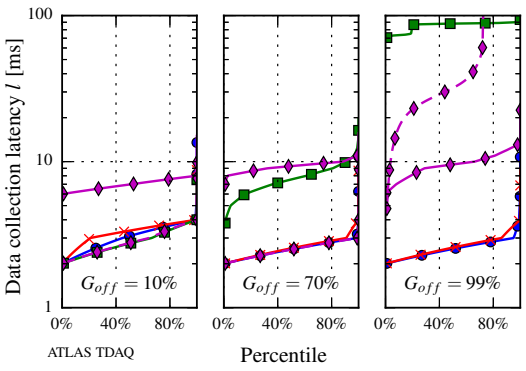


Figure 7.3: Goodput (a), data collection latency (b), and the total number of TCP retransmissions (c) when tuning rate limits at daqrings for the setup depicted in Figure 4.14.



(a) (b)



(c)

Configuration	G_{off}	l_m [ms]	σ_l [ms]
Traffic shaping	10 %	3.06	0.53
	70 %	2.51	0.13
	99 %	2.55	0.27
Static TCP	10 %	3.29	0.41
	70 %	2.59	0.29
	99 %	2.64	0.35
Pause frames	10 %	2.83	0.47
	70 %	7.63	1.73
	99 %	85.5	6.09
daqrings (344 Mbps)	10 %	6.99	0.53
	70 %	9.27	0.73
	99 %	9.59	1.67
daqrings (1.38 Gbps)	10 %	2.85	0.48
	70 %	2.5	0.08
	99 %	115.01	163.06

Figure 7.4: Comparison of application-layer traffic shaping, static TCP congestion window, Ethernet IEEE 802.3x pause frame mechanism, and daqrings in the configuration depicted in Figure 4.14. Sustained load (a) and data collection latency (b) in function of the offered load, and the exact distributions of latency for three cases (c).

the performance under full load. Maximum performance is guaranteed with a rate limit of 344 Mbps, for which the goodput reaches the maximum value, data collection is at its minimum, and no TCP retransmissions are observed, which confirms lossless operation. For different loads, the process needs to be repeated.

RESULTS As already discussed in Section 4.5.4, all solutions provide similar performance in incast-avoidance, which can be explained by a less demanding configuration in terms of incast congestion. More detailed study in different configurations is required to show larger potential differences. In the following section we will present another evaluation, emulating stronger incast congestion.

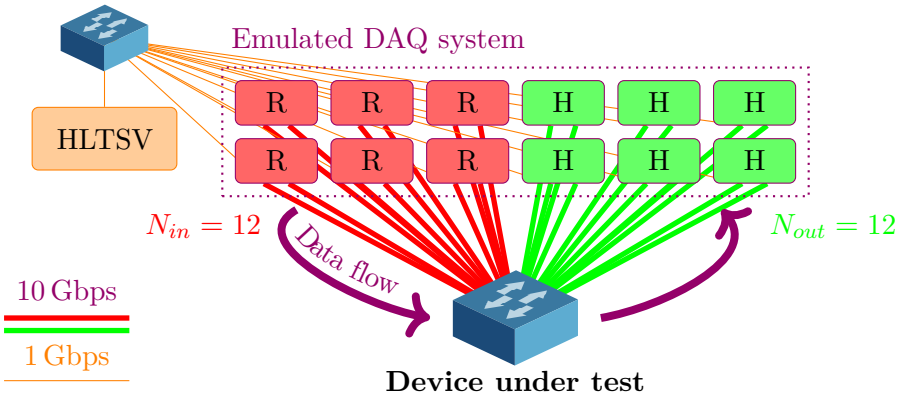


Figure 7.5: Second test configuration for evaluation of the FM10840 reference platform. Six nodes are used to emulate ROSes, each providing single event data fragment of 96 KiB. Eleven DCMs are emulated on each of the remaining six nodes.

The comparison including the proposed approach is presented in Figure 7.4. We consider application-layer traffic shaping (PEPs:0), static TCP (PEPs:0, here the sender congestion windows is tuned to maximise the performance), pause frames (PEPs:0), and daqrings with traffic shaping (PEPs:3) in their optimum configurations. All approaches mitigate incast congestion and sustain the requested load (Figure 7.4a). For daqrings, a rate limit of 344 Mbps is optimal to reach full load.

Latency and jitter (Figure 7.4b and Figure 7.4c) are significantly increased for pause frames. Higher latency is also true for daqrings, but with low jitter (flatter curve). Furthermore, at lower loads daqrings can be tuned and the rate limit can be increased to minimise latency. With the rate limit of 1.38 Gbps the lowest latency and jitter is achieved at the load of 70 %. In this configuration, daqrings provide lowest latency and jitter. At full load, a rate limit of 344 Mbps is required to sustain the load, but here the latency is higher than in case of traffic shaping and static TCP. As we will see in the following section, significant performance improvement can be first achieved with daqrings in a more demanding configuration in terms of incast congestion.

7.3.3 *Test configuration B*

The second test configuration is depicted in Figure 7.5. It is analogue, in terms of the available bandwidth, to the setups used to evaluate the pause frame mechanism in Section 4.5.3 (see Figure 4.12) and software switching in Chapter 5 (see Figure 5.3a). Here, the servers used to emulate the DAQ system are less performant and their configuration is adjusted to reach their maximum performance. It would be more desirable to test with exactly the same setup, but for practical reasons we have had to resort to this setup. The DuT remains the same reference platform as in the previous evaluation in this chapter.

This configuration is significantly more demanding than the previous one, see Section 7.3.2, in terms of incast congestion. Here, we try to emulate a larger data taking configuration with heavy incast congestion. There is a total of twelve nodes connected with two 10 Gbps links to the DuT. Traffic is distributed equally between those links with static routing. Six nodes run ROS applications and the other six nodes are used to emulate HLT racks with eleven independent data collectors on each of them. Each of the DCMs is connected with a single processing unit. Otherwise, the end-nodes tend to operate at lower performance, which is caused by the increased incoming traffic volume over the two 10 Gbps links. Each ROS provides a single event data fragment of 96 KiB, which again maximises the performance of the end-nodes. The total event size is therefore 576 KiB. Using equation (A.12), the theoretical goodput is $G_{theory} = 113.84\,\text{Gbps}$.

In this configuration there are two bottlenecks. The first one is the buffer size of the DuT, if operated as a normal switch (PEPs:0). This is a typical bottleneck under heavy incast congestion, similar as in evaluations in Section 4.5.3 and Chapter 5. The second bottleneck is the performance of the end-nodes, which need to handle the incoming traffic on two 10 Gbps links. This limitation did not affect the performance of the entire setup in the previous evaluations in this chapter because each DCM was connected with a single 1 Gbps link to the network.

Traffic shaping is not tested in this configuration as there is only six event fragments, which limits tuning capabilities. For static TCP, there is only a single value ($cwnd = 2$) that provides stable operation with low packet loss rate.

DAQRINGS TUNING Figure 7.6 is used to tune traffic shaping at the daqrings level. Because of the special configuration with multiple data collectors emulated on a single node, we do not set a single daqring for a single DCM. We choose to configure a single daqring dedicated to a single 10 Gbps output port on the DuT (static routing). Fine-tuning for a smaller number of daqrings with higher rate limits is easier, so it also easier to achieve higher utilisation of the 50 Gbps PCIe links between the switching ASIC and the software switch on the DuT (see Section 7.3.1).

Also, because of the low performance of the end-nodes, we enable transmission of pause frames from the end-nodes to the DuT only, similarly as in Section 5.5.3. Otherwise, they can drop packets because of temporal fluctuations in the available CPU power and negatively affect the throughput of the entire setup. The transmission of pause frames from the DuT is disabled as we aim to mitigate incast congestion with daqrings. This combined approach provides the best performance and avoids overflowing the end-nodes because of any temporary fluctuations in performance-limited data collectors.

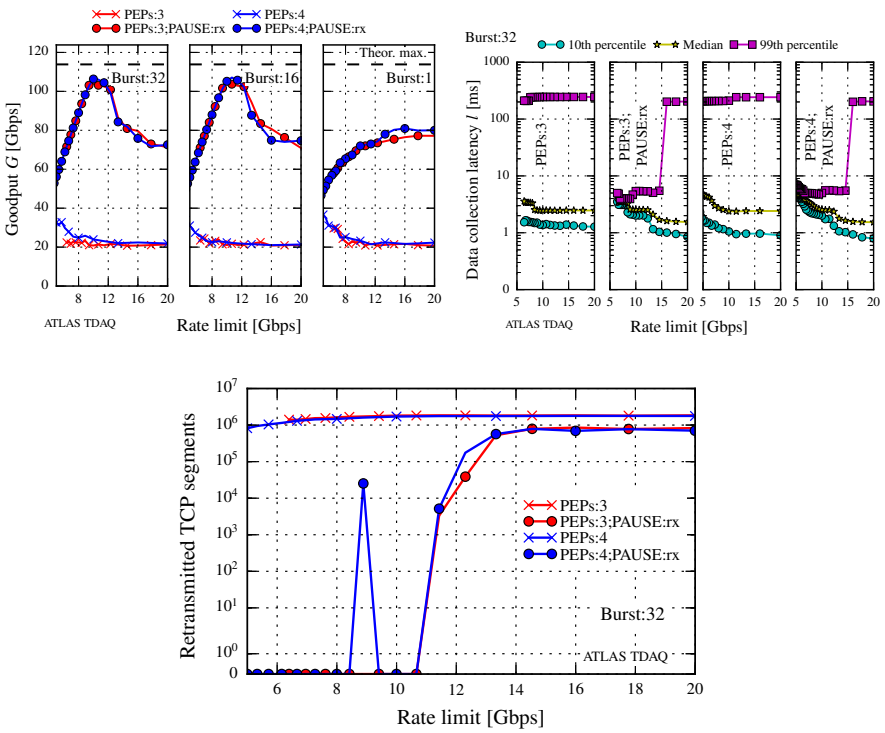


Figure 7.6: Goodput (a), data collection latency (b), and the total number of TCP retransmissions (c) when tuning daqrings for the setup depicted in Figure 7.5.

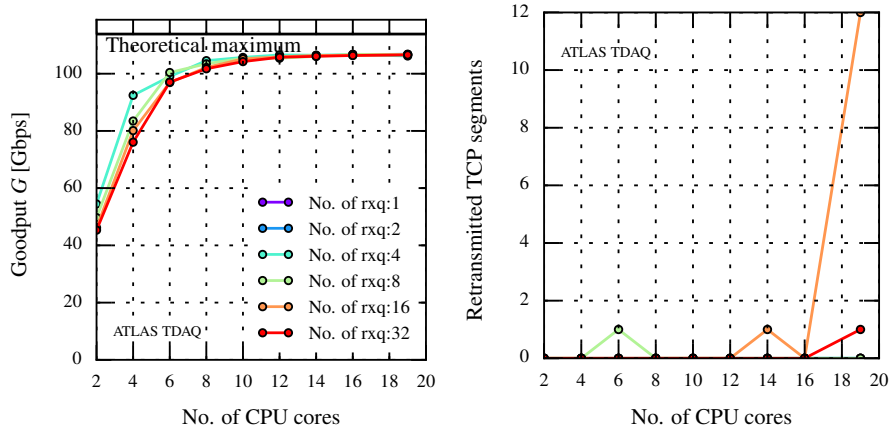


Figure 7.7: Goodput (a) and the total number of TCP retransmissions (b) when changing the number of the rx-queues and CPU cores used by OVS for the setup depicted in Figure 7.5.

Here, we also extended the diagrams with different burst sizes for the same value of the rate limit (it is defined as a maximum number of packets, burst size, that can be received from a daqring in a given poll interval). Burst sizes of 32 and 16 provide the best performance under the rate limit of 10 Gbps, with enabled reception of pause frames. There is a spike in the TCP retransmissions diagram for the rate-limit of approximately 9 Gbps. A probable reason is a temporal high degradation of the performance at the end-nodes due to some CPU-intensive task.

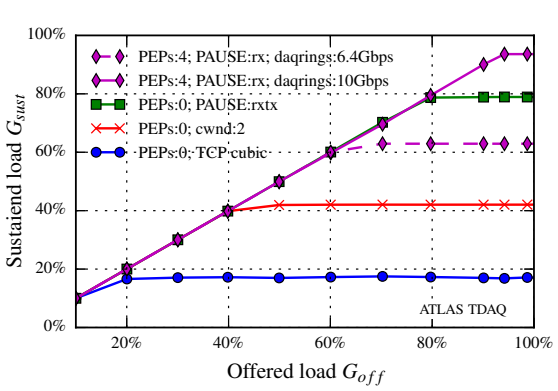
Additionally, in Figure 7.7, we present the performance for the optimum rate limit, but for different configurations of the software switch. We check what is the effect of the number of the hardware receive queues of the DuT and the number of the CPU cores used by the software switch. It is clear that it is enough to use ten CPU cores to reach full goodput. The number of receive queues has only a minimal effect in this setup. This demonstrates that there is a high performance margin at the software switch.

RESULTS The comparison of the various incast-avoidance techniques with this evaluation setup is presented Figure 7.8. In this demanding scenario, the advantage of using daqrings in combination with partial pause frame mechanism is clearly visible. For the optimum rate limit of 10 Gbps at full load, goodput saturates at 94 % of the theoretical maximum, whereas the second best, full pause frame configuration (transmit and receive), reaches only 79 %. The mean data collection latency is slightly higher for daqrings, but this is expected because of higher sustained load. Jitter remains minimal in both cases. At lower loads daqrings can be tuned again, just as in Section 7.3.2, in order to minimise the latency.

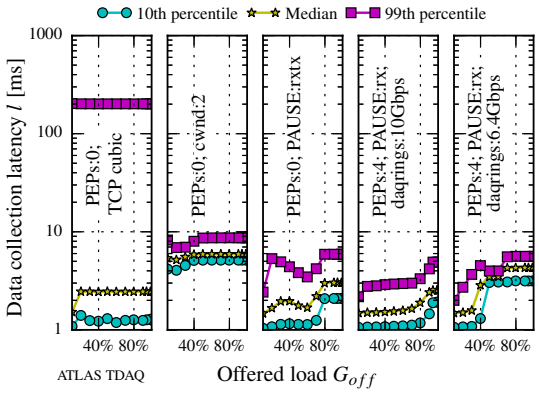
Static TCP congestion window saturates just above 40 % of the theoretical goodput. For this evaluation setup, tuning capabilities are limited too much. Congestion window of two packets on each TCP connection is too low to optimise the goodput, whereas a value of three already triggers incast.

The effects of incast congestion without avoidance techniques can be observed for the default congestion control, TCP cubic. In this case, the load does not exceed 20 % and some events need more than 200 ms to be collected, which indicates packet losses and TCP timeouts.

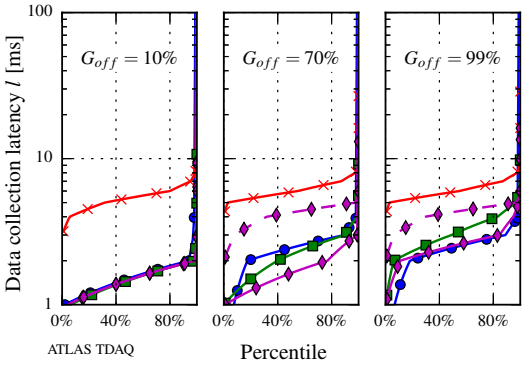
The optimum performance is slightly worse than in the similar evaluations performed with traditional NICs in Chapter 5. The reason is, however, the limited performance of the emulated DAQ configuration. The difference is minimal, which confirms that multi-host Ethernet controllers can be indeed considered as a valid replacement for traditional NICs when building software switches.



(a)



(b)



(c)

Configuration	G_{off}	l_m [ms]	σ_l [ms]
TCP Cubic	10 %	8.15	57.57
	70 %	6.32	45.69
	99 %	6.19	44.4
Static TCP	10 %	5.91	17.44
	70 %	6.08	0.84
	99 %	6.08	0.87
Pause frames	10 %	1.46	0.34
	70 %	2.22	0.69
	99 %	3.16	0.93
daqrings (10 Gbps)	10 %	1.47	0.24
	70 %	1.98	7.76
	99 %	2.6	0.68
daqrings (6.4 Gbps)	10 %	1.47	0.23
	70 %	4.12	0.67
	99 %	4.17	0.64

Figure 7.8: Comparison of TCP Cubic, static TCP congestion window, Ethernet IEEE 802.3x pause frame mechanism, and daqrings in the configuration depicted in Figure 7.5. Sustained load (a) and event data collection latency (b) in function of the offered load, and the exact distributions of latency for three different cases (c).

7.4 CONCLUSION

In this chapter we evaluated whether traditional network interfaces can be replaced with multi-host Ethernet controllers when building software switches. We indicated what advantages they offer, including better space utilisation. We also pointed out that these devices can be potentially used as an alternative to other incast-avoidance techniques in datacenters, where limitations of typical software switches have stronger consequences than in data acquisition.

Finally, we performed an initial evaluation of a multi-host controller, using a reference platform. We confirmed that similar or better performance to other incast-avoidance techniques can be achieved. Also,

we did not observe any drawbacks when using this device instead of multiple traditional network interfaces to construct a software switch. Thus, our evaluation showed that this method for incast-avoidance is valid and can be a new application area for the multi-host Ethernet devices, both in data acquisition and datacenter networks.

8 CONCLUSIONS AND OUTLOOK

This chapter concludes this Ph.D. and highlights the personal contributions to the area of network design and optimisation in data acquisition systems as well as alternative approaches to incast congestion. Finally, we give directions for future research in the area.

8.1 INTRODUCTION

The data acquisition networks of the LHC experiments at CERN have unusually demanding requirements in terms of bandwidth and loss rate. They are particularly challenging because of the bursty many-to-one communication pattern. The ports connected to receiving data collectors are temporarily overcommitted by the data sent from the readout nodes. Switches with insufficient buffers drop some or most of the packets, which degrades the performance or prevents normal operation of an entire data acquisition system. This will become even more critical, as the planned upgrades of the LHC experiments will require higher bandwidth networks for their DAQ systems. The network congestion will become even more demanding.

Such unusually demanding environment provided the motivation for the work presented in this Ph.D. We studied whether a leaf-spine topology of software switches running on commercial-off-the-shelf servers could replace expensive, telecom-class devices in data acquisition systems, using the ATLAS experiment as a case study. We demonstrated that this approach offers the flexibility of design in software, enough performance in packet forwarding, and buffering capabilities constrained solely by the amount of DRAM memory. The latter is the key aspect in incast-avoidance. We also showed how a lossless, high-throughput network based on commodity software switches can be designed and optimised for a target application.

8.2 REVIEW OF THE RESEARCH

In this thesis we discussed our original contributions to the understanding of incast congestion in data acquisition, performance and optimisation of software switches for incast-avoidance as well as their applicability in designing high-bandwidth networks.

Chapter 4

First, we have established that there is an analogous problem in datacenter networks. This problem results from the similar many-to-one traffic pattern. It is particularly critical for TCP communication and therefore referred to as the TCP incast pathology. With this analogy, we derived simple equations to estimate the onset of incast congestion for DAQ networks. Also, we showed that advanced solutions proposed for DCN at different layers, like DIATCP or IEEE 802.3x pause frame, can be also considered in DAQ. On the other hand, we also proposed a simple congestion control mechanism for DAQ. The total amount of packets injected into a network can be controlled by a static congestion window of each TCP flow. This can be a simple and effective solution in some configurations. Nevertheless, all of these approaches have their limitations, and, in case of data acquisition in particular, still require large packet buffers in the core of the network. Also, even with an effective algorithm, more event data have to be eventually buffered at the sender side. This prevents the use of simple devices in the readout system.

Chapter 5

These conclusions turned our attention to one of the most effective ways of incast-avoidance in data acquisition — large packet buffers. The key characteristics of DAQ networks allow for some simplifications that make it easier to pursue this strategy. Particularly, we have identified that software-based switches optimised for a target application can be a valid alternative to expensive, telecom-class routers. The former provide almost limitless memory for packet buffers. We designed a mechanism for application-dedicated queueing, which can be programmed in software and which provides a number of queues that is not limited by hardware resources, as in traditional switches and routers. Our prototypes proved that saturation and lossless operation can be reached on real hardware providing the total bandwidth of 120 Gbps in the all-to-all incast scenario, where traditional ToR switches perform poorly. And, importantly, controls on the injected traffic are not required. Our extension for incast-avoidance in DAQ for the popular software switch, Open vSwitch, has been made available and is ready for use for other researchers or network engineers.

Chapter 6

Although these small prototypes have already reached bandwidths comparable to those of ATLAS DAQ network in Run 1 and Run 2, we continued the study and focused on building a larger topology of interconnected software switches. This is required to provide full connectivity within a DAQ system and scale by two or more orders of magnitude for the future upgrades of the LHC experiments. We took advantage of the parallel leaf-spine topology and we optimised flow distribution across available paths through the fabrics for the specific traffic pattern. We showed that it is feasible to build terabit data acquisition networks using dedicated software switches. This approach, together with separate queues for every data collector for incast-avoidance, can offer cost and performance advantage over the traditional network designs, where

expensive telecom-class devices are required. In order to strengthen the arguments, we proposed and evaluated, on real hardware, a method to manage and optimise such a network using software-defined technologies, OpenFlow and OVSDB. We showed that the network can be centrally programmed using solely IP and TCP addressing. Using traffic shaping at the dedicated queues, we demonstrated how the network can be tuned, in various configurations, to maximise the system's performance and reach lossless operation with high throughput and low latency. Thus, we have shown that this design, including optimised software switches, adapted leaf-spine topology and software-defined control plane, can be a viable solution for future data acquisition networks.

Finally, we considered replacing traditional network interfaces with a new class of devices, multi-host Ethernet controllers, to build software switches. We demonstrated that not only the physical space utilisation can be improved, but also that these devices can be potentially used as an alternative to other incast-avoidance techniques in datacenters. Limitations of typical software switches have more significant consequences there than in data acquisition. Our initial evaluation of a reference platform confirmed that similar or better performance to other incast-avoidance techniques can be achieved. *Chapter 7*

8.3 FUTURE DIRECTIONS

The work described in this thesis has covered most aspects of using software switching for building lossless networks for data acquisition systems with strong incast congestion. Nevertheless, further refinements in some areas can be achieved. Our work can also provide a basis for new research studies.

First, further investigation on the queueing algorithms could be beneficial. In our work on software switches with dedicated queues we have already demonstrated that throughput is maximised. The study could be extended with more focus on the data collection latency. One avenue is to explore service disciplines for daqrings. The question here is whether and at which network stages it would be profitable to serve daqrings on per-event basis. More specifically, the software switch would give preference to those daqrings where some event is already being transported, rather than daqrings which have just started to buffer a new event (e. g. earliest deadline first).

In this work we focused solely on event data flows. This is, however, not the only type of traffic that occurs in DAQ networks. The straightforward examples of others are control and storage traffic. The former is often handled by a separate network. With the Priority-based Flow Control (IEEE 802.1Qbb [75]) it could be possible to use a single network for all types of traffic. Separate hardware queues can be the assigned to different traffic classes and event data flows would additionally use software queues — daqrings.

In Chapter 5 we gave estimates on power consumption of a software switch. The comparison gave advantage to the traditional ToR switches. There is, however, room for improvement. First of all, it would be worth considering interrupts instead of continuous polling when receiving packets on the software switch. This should significantly reduce CPU utilisation, and, thus, also the power consumption. The effects on the overall switch performance should be carefully analysed though as the packet reception mechanism would be changed considerably. A less invasive approach is to automatically regulate the polling interval. When the network load decreases, the software switch should increase the polling interval and it should be reduced when the load increases.

Also better insight into the failover mechanism in the proposed parallel leaf-spine topology with optimised software switches is of high relevance. Our initial study from Chapter 6 could be extended with different link failure scenarios. Furthermore, the performance could be improved by implementing a dedicated service for flow distribution and failover in the controller framework instead of using the REST API. This approach could reduce the time it takes to detect link failures and redistribute the flows. The failover study could also include a scenario when the network controller becomes unavailable. Here, it would be of interest to study controller redundancy or flow caching at the switching nodes.

It would be also beneficial to consider a generic network controller for data acquisition networks. Instead of time-consuming scanning of the entire network for data flows, this generic service could implement logic to detect and distribute the flows automatically. Moreover, this could become independent of the specifics of an experiment, if a proper abstraction of a DAQ network is also defined. It is also of importance to study non-heterogeneous networks, in which slower and faster nodes are present or topology is not consistent. In this case, it would be worth to consider dynamic, instead of our proposed static, flow assignment, like the Hedera flow scheduler [57]. With this generic network controller, feedback on link utilisation in the farm could be also used to optimise the process of event assignment to processing nodes. In the longer term, it could be the network, not the readout node or the supervisor, that takes decision which processing node to use for a particular event.

While considering possible integration with the network controller framework, it would be also of interest to study even higher level of system integration. Open vSwitch is already heavily used in the cloud computing frameworks like OpenStack [127], which are gradually replacing traditional datacenters [51]. OpenStack software controls large pools of compute, storage, and networking resources throughout a datacenter, which can be managed through a dashboard or an API. OpenStack could be considered to control a data acquisition system, including the readout nodes, the event building and filtering farm, the storage as well as the network.

This approach could also make it easier to potentially integrate networking and event data processing on the same physical nodes. Our studies in Chapter 5 showed that this approach is feasible, although there are limitations that need to be considered. Particularly, the shared access to memory could be avoided when using different NUMA nodes for different processes. This is feasible with devices like the CSA-7400 computing platform that we described in Chapter 7. Another alternative is to consider a mechanism to allocate different limits on memory bandwidth usage to different processes. One example is the MemGuard memory bandwidth reservation system proposed by Yun et al. in [176].

Finally, software switches with high buffering capabilities and flexible queueing options can be considered as a valid option not only in data acquisition systems, but other networks suffering from many-to-one communication as well. In Chapter 7 we have already shown that they could be considered for incast-avoidance in datacenters. Another application area is potentially the emerging Internet of Things (IoT), which is a network of interconnected objects that senses information from the environment, interacts with the physical world, and provides services for information transfer, analytics, applications and communications [91]. IoT can be seen, to some extent, as an unstructured data acquisition network, and as such can be vulnerable to incast congestion. This observation was also made by Jin et al. in [91], who stated that the most common operation of an IoT network is to collect data from hundreds of thousands of nodes and congestion can occur near the data sinks. The methods proposed in this thesis could be therefore also effective in the IoT area.

A

FORMULAS

This appendix gives formulas and some of their derivations that are used to evaluate the performance of DAQ networks in this thesis.

A.1 A SIMPLE MODEL FOR BANDWIDTH

Bandwidth of a DAQ network under test (Figure 3.2), as defined in Section 3.2, gives the theoretical raw bit rate that can be achieved for data flows from the readout to the filtering farm. Traffic to storage and control traffic are neglected as explained in Section 3.3.

We assume that event data are evenly distributed across N_R readout nodes (denoted as R in Figure 3.2), which in turn are connected with N_{in} Ethernet links to the DAQ network. Node i has n_{R_i} links. The total number of input links is therefore

$$N_{in} = \sum_{i=1}^{N_R} n_{R_i} \, .$$

DCMs request all fragments of an assigned event from all ROSes (event building). DCMs are organised in racks of the HLT farm (denoted as H in Figure 3.2), which are connected over a ToR switch to the DAQ network. It is assumed that internal configuration of each HLT rack is not limiting the overall theoretical bandwidth[1]. An HLT rack can be also emulated on a single node. In each case, there are N_H HLT racks. Rack j is connected with n_{H_j} Ethernet links to the network. The total number of output links from the DAQ network to HLT is

$$N_{out} = \sum_{j=1}^{N_H} n_{H_j} \, .$$

For full event building, the total bandwidth available for DAQ data flows can be calculated with

$$B = \sum_{j=1}^{N_H} N_R \min_i \left(b_{ij} \right) \, , \tag{A.1}$$

where each data collection process is limited by the bandwidth b_{ij} of the slowest ROS-to-HLT flow. It is explained by the fact that every collector waits to collect all fragments before proceeding with new requests, which effectively means that data collection throughput is determined by the slowest flow. Speed of this single flow is limited here either by

1 The saturation goodput though is often limited by the ToR switches. This is one of the subjects to optimisation in this thesis.

the speed of the ROS-to-network, network-to-HLT, or some other link inside the network:

$$b_{ij} = \min\left(b_{ij\,R}, b_{ij\,H}, b_{ij\,net}\right) . \tag{A.2}$$

On the ROS-to-network side the limitation is set by the number of parallel transfers to HLT:

$$b_{ij\,R} = \frac{n_{R_i} b}{N_H} , \tag{A.3}$$

whereas on the network-to-HLT side by the number of parallel transfers from ROS:

$$b_{ij\,H} = \frac{n_{H_j} b}{N_R} , \tag{A.4}$$

where b is the speed of a single input or output link to the network. In equations (A.3) and (A.4) we assume perfect load balancing across flows. Combining equations (A.1), (A.2), (A.3), and (A.4), the bandwidth can be expressed as:

$$B = \sum_{j=1}^{N_H} N_R \min_i \left(\frac{n_{R_i} b}{N_H}, \frac{n_{H_j} b}{N_R}, b_{ij\,net} \right) . \tag{A.5}$$

If the system is homogeneous, i. e. the number of links to the network per ROS or HLT rack is the same across all nodes ($n_{R_i} = n_R$, $n_{H_j} = n_H$) and the network is perfectly fair ($b_{ij\,net} = b_{net}$) it simplifies to

$$\begin{aligned} B &= \sum_{j=1}^{N_H} N_R \min_i \left(\frac{n_R b}{N_H}, \frac{n_H b}{N_R}, b_{net} \right) \\ &= N_H N_R \min \left(\frac{n_R b}{N_H}, \frac{n_H b}{N_R}, b_{net} \right) \\ &= \min\left(N_R n_R b, N_H n_H b, b_{net}\right) . \end{aligned} \tag{A.6}$$

If the only limiting factors are the input or output links, the theoretical maximum bandwidth is given by

$$B_{inout} = \min\left(N_R n_R b, N_H n_H b\right) . \tag{A.7}$$

A.2 THEORETICAL GOODPUT

The network bandwidth defined in the previous section gives the raw bit rate available for data acquisition between ROS and HLT. It neglects, however, the overheads of the upper layer protocols. In order to estimate the theoretical event data bandwidth we use the theoretical goodput, which takes those overheads into account. Using the ATLAS TDAQ software with Ethernet, TCP/IP, and ATLAS data flow as underlying protocols, it can be calculated as follows.

Let the total average event size be e_{avg}. Event data is spread equally across all ROS nodes and there is $e_R = \frac{e_{avg}}{N_R}$ of event data bytes on each node[2], like in the previous section. The bandwidth, as defined by equations (A.5), (A.6), or (A.7), is used to carry bits of event data and protocol headers. Let the total number of bytes r that are required to transport all bytes of an event from all ROS nodes be

$$r = e_{avg} + o\,, \tag{A.8}$$

where o is the total number of overhead bytes. The maximum theoretical event rate is then given by

$$L1r = \frac{B}{r}\,.$$

Goodput, as defined in Section 3.2, refers though to pure event bytes transferred to the data collectors. The theoretical goodput is therefore

$$G_{theory} = L1r \cdot e_{avg} = B\frac{e_{avg}}{r}\,, \tag{A.9}$$

where $\frac{e_{avg}}{r}$ is called protocol efficiency.

For TCP/IP-based ATLAS DAQ network, the overhead can be calculated with

$$o = n_{frames}\,(o_{Ethernet} + o_{IP} + o_{TCP}) + N_R \cdot o_{ATLAS}\,.$$

Ethernet, IP, and TCP overheads appear in every Ethernet frame, whereas additional bytes of the ATLAS data flow protocol are added for just a single ROS response. Particular overheads, including the Ethernet frame format as defined originally in the IEEE 802.3 Standard for Ethernet [76], are as follows [62]:

$$\begin{aligned}
o_{Ethernet} &= \text{InterFrame Gap} + \text{Preamble} \\
&\quad + \text{Start Frame Delimiter} \\
&\quad + \text{Ethernet header} + \text{padding} \\
&= 12\,\text{B} + 7\,\text{B} + 1\,\text{B} + 18\,\text{B} + \mathit{padding} \\
o_{IP} &= \text{IP header} = 20\,\text{B} \\
o_{TCP} &= \text{TCP header without options} = 20\,\text{B} \\
o_{ATLAS} &= \text{ROS response header} = 12\,\text{B}\,.
\end{aligned}$$

Padding is required, if TCP payload in a single frame is smaller than 6 B. The total overhead is now given by

$$o = n_{frames} \cdot 78\,\text{B} + N_R \cdot 12\,\text{B} + \mathit{padding}\,.$$

2 If ROS and HLT nodes are emulated on the same nodes, the total event bytes traversing the physical network links are less, because one ROS is using the loopback link to the HLT running on the same host. This correction should be applied in the calculations.

The maximum TCP segment size (MSS) for a single Ethernet frame with an MTU of 1500 B (see Section 3.5.1) is

$$MSS = 1500\,\mathrm{B} - \text{IP header} - \text{TCP header} = 1460\,\mathrm{B}\,.$$

Single response from ROS requires therefore

$$n_{frames_R} = \left\lceil \frac{e_R + 12\,\mathrm{B}}{MSS} \right\rceil = \left\lceil \frac{\frac{e_{avg}}{N_R} + 12\,\mathrm{B}}{1460\,\mathrm{B}} \right\rceil$$

frames. Total overhead can be written as

$$\begin{aligned} o &= n_{frames} \cdot 78\,\mathrm{B} + N_R \cdot 12\,\mathrm{B} + padding \\ &= N_R \cdot n_{frames_R} \cdot 78\,\mathrm{B} + N_R \cdot 12\,\mathrm{B} + padding \\ &= N_R \left(\left\lceil \frac{\frac{e_{avg}}{N_R} + 12\,\mathrm{B}}{1460\,\mathrm{B}} \right\rceil \cdot 78\,\mathrm{B} + 12\,\mathrm{B} \right) + padding\,. \end{aligned} \tag{A.10}$$

Frames with less than minimum 64 B for Ethernet are padded to 64 B. Therefore *padding* can be calculated as

$$\begin{aligned} padding &= \begin{cases} 0, & \text{if } e_R \bmod MSS \geq 6\,\mathrm{B} \\ N_R\,(6\,\mathrm{B} - e_R \bmod MSS)\,, & \text{otherwise} \end{cases} \\ &= \begin{cases} 0, & \text{if } \frac{e_{avg}}{N_R} \bmod 1460\,\mathrm{B} \geq 6\,\mathrm{B} \\ N_R \left(6\,\mathrm{B} - \frac{e_{avg}}{N_R} \bmod 1460\,\mathrm{B}\right), & \text{otherwise}\,. \end{cases} \end{aligned} \tag{A.11}$$

Combining equations (A.8), (A.9), (A.10), and (A.11), the theoretical goodput is given by

$$G_{theory} = B \frac{e_{avg}}{e_{avg} + N_R \left(\left\lceil \frac{\frac{e_{avg}}{N_R} + 12\,\mathrm{B}}{1460\,\mathrm{B}} \right\rceil \cdot 78\,\mathrm{B} + 12\,\mathrm{B} \right) + padding},$$

$$padding = \begin{cases} 0, & \text{if } \frac{e_{avg}}{N_R} \bmod 1460\,\mathrm{B} \geq 6\,\mathrm{B} \\ N_R \left(6\,\mathrm{B} - \frac{e_{avg}}{N_R} \bmod 1460\,\mathrm{B}\right), & \text{otherwise}\,. \end{cases} \tag{A.12}$$

In case that traffic from HLT to ROS (TCP ACKs and requests for event data) and from ROS to HLT (event data flow) traverses some link in the network in the same direction, additional overhead caused by the former traffic should be normally considered. Since TCP ACKs and requests from HLT to ROS are small compared to actual event data flow, we neglect this type of overhead.

A.3 MEAN AND JITTER

Mean of the data collection latency, as defined in Section 3.2, is a simple arithmetic mean of a set of sample taken during data-taking period:

$$l_m = \frac{1}{N} \sum_{i=1}^{N} l_i \, .$$

Jitter of the data collection latency can be estimated using standard deviation:

$$\sigma_l = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (l_i - l_m)^2}$$

In this thesis also we use percentile plots to analyse the distribution and jitter of the data collection latency.

BIBLIOGRAPHY

[1] *42U Rack Dimensions, Cabinet Size, and Specifications*. Online; accessed 2016-11-18. URL: http://www.42u.com/42U-cabinets.htm.

[2] *6WIND*. Online; accessed 2016-03-01. URL: http://www.6wind.com.

[3] *Adlink CSA-7400*. Online; accessed 2016-12-17. URL: http://www.adlinktech.com/PD/web/PD_detail.php?cKind=&pid=1624&seq=&id=&sid=&category=Server_Network-Appliance&utm_source=#.

[4] I. F. Akyildiz et al. "Research Challenges for Traffic Engineering in Software Defined Networks." In: *IEEE Network* 30.3 (2016), pp. 52–58. DOI: 10.1109/MNET.2016.7474344.

[5] M. Alizadeh and T. Edsall. "On the Data Path Performance of Leaf-Spine Datacenter Fabrics." In: *2013 IEEE 21st Annual Symposium on High-Performance Interconnects*. 2013, pp. 71–74. DOI: 10.1109/HOTI.2013.23.

[6] Mohammad Alizadeh et al. "Data Center TCP (DCTCP)." In: *SIGCOMM Comput. Commun. Rev.* 40.4 (2010). DOI: 10.1145/1851275.1851192.

[7] A. Andreyev. *Introducing Data Center Fabric, the Next-Generation Facebook Data Center Network*. Online; accessed 2016-11-07. URL: https://code.facebook.com/posts/360346274145943/introducing-data-center-fabric-the-next-generation-facebook-data-center-network/.

[8] Gianni Antichi et al. "Time Structure Analysis of the LHCb DAQ Network." In: *Journal of Physics: Conference Series* 513.6 (2014), p. 062009.

[9] Guido Appenzeller, Isaac Keslassy, and Nick McKeown. "Sizing Router Buffers." In: *SIGCOMM Comput. Commun. Rev.* 34.4 (2004), pp. 281–292. DOI: 10.1145/1030194.1015499.

[10] Katerina Argyraki et al. "Can Software Routers Scale?" In: *Proceedings of the ACM Workshop on Programmable Routers for Extensible Services of Tomorrow*. PRESTO '08. ACM, 2008, pp. 21–26. DOI: 10.1145/1397718.1397724.

[11] InfiniBand Trade Association et al. *InfiniBand Architecture Specification 1.2.1*. Online; accessed 2016-06-27. URL: http://www.infinibandta.org/.

[12] Benjamin J van Asten, Niels L. M. van Adrichem, and Fernando A Kuipers. "Scalability and Resilience of Software-Defined Networking: An Overview." In: *CoRR* abs/1408.6760 (2014).

[13] Wei Bai et al. "PAC: Taming TCP Incast Congestion Using Proactive ACK Control." In: *Proceedings of the 2014 IEEE 22nd International Conference on Network Protocols (ICNP)*. IEEE. 2014, pp. 385–396. DOI: 10.1109/ICNP.2014.62.

[14] F. Baker. *Requirements for IP Version 4 Routers*. RFC 1812. RFC Editor, 1995.

[15] Hitesh Ballani et al. "Enabling End-Host Network Functions." In: *SIGCOMM Comput. Commun. Rev.* 45.4 (2015). DOI: 10.1145/2829988.2787493.

[16] Tom Barbette, Cyril Soldani, and Laurent Mathy. "Fast Userspace Packet Processing." In: *Proceedings of the Eleventh ACM/IEEE Symposium on Architectures for Networking and Communications Systems*. ANCS '15. IEEE Computer Society, 2015, pp. 5–16.

[17] Rainer Bartoldus et al. *Technical Design Report for the Phase-I Upgrade of the ATLAS TDAQ System*. Tech. rep. Geneva: CERN, 2013. URL: https://cds.cern.ch/record/1602235.

[18] Tomasz Bawej et al. "Boosting Event Building Performance Using Infiniband FDR for the CMS Upgrade." In: *Proceedings of the 3rd International Conference on Technology and Instrumentation in Particle Physics (TIPP 2014)*. Vol. TIPP2014. 2014, p. 190.

[19] Tomasz Bawej et al. "The New CMS DAQ System for Run-2 of the LHC." In: *Proceedings of the 2014 19th IEEE-NPSS Real Time Conference (RT)*. 2014, pp. 1–1. DOI: 10.1109/RTC.2014.7097437.

[20] Stephen Bensley et al. *Datacenter TCP (DCTCP): TCP Congestion Control for Datacenters*. Internet-Draft draft-ietf-tcpm-dctcp-01. IETF Secretariat, 2015. URL: http://www.ietf.org/internet-drafts/draft-ietf-tcpm-dctcp-01.txt.

[21] Gary Berger. *Facebook Fabric Networking Deconstructed*. Online; accessed 2016-11-07. URL: http://firstclassfunc.com/facebook-fabric-networking.

[22] Alessandro Bettini. *Introduction to Elementary Particle Physics; 2nd ed.* Cambridge University Press, 2014. URL: http://cds.cern.ch/record/1611164.

[23] Andrea Bianco et al. "Click vs. Linux: Two Efficient Open-Source IP Network Stacks for Software Routers." In: *Proceedings of the 2005 Workshop on High Performance Switching and Routing (HPSR)*. IEEE. 2005, pp. 18–23.

[24] Mark S Birrittella et al. “Intel® Omni-Path Architecture: Enabling Scalable, High Performance Fabrics.” In: *Proceedings of the IEEE 23rd Annual Symposium on High-Performance Interconnects (HOTI 2015)*. IEEE. 2015, pp. 1–9.

[25] Raffaele Bolla and Roberto Bruschi. “Linux Software Router: Data Plane Optimization and Performance Evaluation.” In: *Journal of Networks* 2.3 (2007), pp. 6–17.

[26] Raffaele Bolla and Roberto Bruschi. “PC-based Software Routers: High Performance and Application Service Support.” In: *Proceedings of the ACM Workshop on Programmable Routers for Extensible Services of Tomorrow*. PRESTO ’08. ACM, 2008, pp. 27–32. DOI: 10.1145/1397718.1397725.

[27] *Broadberry*. Online; accessed 2016-11-14. URL: https://www.broadberry.co.uk.

[28] *Brocade MLX Series*. Online; accessed 2016-07-21. URL: http://www.brocade.com/.

[29] Daniel Campora, Niko Neufeld, and Rainer Schwemmer. “Improvements in the LHCb DAQ.” In: *Proceedings of the 19th IEEE-NPSS Real Time Conference (RT 2014)*. IEEE. 2014. DOI: 10.1109/RTC.2014.7097512.

[30] F Carena et al. “Preparing the ALICE DAQ Upgrade.” In: *Journal of Physics: Conference Series* 396.1 (2012), p. 012050.

[31] F Carena et al. “The ALICE Data Acquisition System.” In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 741 (2014), pp. 130–162. DOI: http://dx.doi.org/10.1016/j.nima.2013.12.015.

[32] *CERN - European Organization for Nuclear Research*. Online; accessed 2016-02-10. URL: home.cern.

[33] *CERN openlab*. Online; accessed 2017-01-28. URL: openlab.cern.

[34] Ivano Cerrato, Mauro Annarumma, and Fulvio Risso. “Supporting Fine-Grained Network Functions Through Intel DPDK.” In: *Proceedings of the 2014 Third European Workshop on Software Defined Networks*. EWSDN ’14. IEEE Computer Society, 2014. DOI: 10.1109/EWSDN.2014.33.

[35] Benjie Chen and Robert Morris. “Flexible Control of Parallelism in a Multiprocessor PC Router.” In: *USENIX Annual Technical Conference, General Track*. 2001, pp. 333–346.

[36] Yanpei Chen et al. *Understanding TCP Incast and Its Implications for Big Data Workloads*. Tech. rep. UCB/EECS-2012-40. EECS Department, University of California, Berkeley, 2012.

[37] Hyunjeong Cho, Saehoon Kang, and Younghee Lee. "Centralized ARP Proxy Server over SDN Controller to Cut Down ARP Broadcast in Large-Scale Data Center Networks." In: *2015 International Conference on Information Networking (ICOIN)*. 2015, pp. 301–306. DOI: 10.1109/ICOIN.2015.7057900.

[38] C. Clos. "A Study of Non-Blocking Switching Networks." In: *The Bell System Technical Journal* 32.2 (1953), pp. 406–424. DOI: 10.1002/j.1538-7305.1953.tb01433.x.

[39] Tommaso Colombo and ATLAS Collaboration. "Data-flow Performance Optimisation on Unreliable Networks: the ATLAS Data-Acquisition Case." In: *Journal of Physics: Conference Series* 608.1 (2015), p. 012005.

[40] Robert M Crovella. "Sensor Networks and Communication." In: *Measurement, Instrumentation, and Sensors Handbook; 2nd ed.* Ed. by John G Webster and Halit Eren. CRC Press, 2014. ISBN: 978-1-4398-4891-3. DOI: 10.1201/b15664-17. URL: http://dx.doi.org/10.1201/b15664-17.

[41] L. Cui, F. R. Yu, and Q. Yan. "When Big Data Meets Software-Defined Networking: SDN for Big Data and Big Data for SDN." In: *IEEE Network* 30.1 (2016), pp. 58–65. DOI: 10.1109/MNET.2016.7389832.

[42] William James Dally and Brian Patrick Towles. *Principles and Practices of Interconnection Networks*. Elsevier, 2004.

[43] *daq-software-switching*. GitHub repository. URL: github.com/gjerecze/daq-software-switching.

[44] *Data Center Bridging Task Group*. Online; accessed 2016-03-15. URL: http://www.ieee802.org/1/pages/dcbridges.html.

[45] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters." In: *Commun. ACM* 51.1 (2008), pp. 107–113. DOI: 10.1145/1327452.1327492.

[46] C. J. Sher Decusatis, A. Carranza, and C. M. Decusatis. "Communication Within Clouds: Open Standards and Proprietary Protocols for Data Center Networking." In: *IEEE Communications Magazine* 50.9 (2012), pp. 26–33. DOI: 10.1109/MCOM.2012.6295708.

[47] Prajjwal Devkota and AL Narasimha Reddy. "Performance of Quantized Congestion Notification in TCP Incast Scenarios of Data Centers." In: *Proceedings of the 2010 IEEE International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE. 2010. DOI: 10.1109/MASCOTS.2010.32.

[48] Alberto Di Meglio, Melissa Gaillard, and Andrew Purcell. *CERN openlab Whitepaper on Future IT Challenges in Scientific Research*. 2014. DOI: 10.5281/zenodo.8765.

[49] Mihai Dobrescu et al. “RouteBricks: Exploiting Parallelism to Scale Software Routers.” In: *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*. SOSP ’09. ACM, 2009, pp. 15–28. DOI: 10.1145/1629575.1629578.

[50] Norbert Egi et al. “Towards High Performance Virtual Routers on Commodity Hardware.” In: *Proceedings of the 2008 ACM CoNEXT Conference*. CoNEXT ’08. ACM, 2008, 20:1–20:12. DOI: 10.1145/1544012.1544032.

[51] Paul Emmerich et al. “Performance Characteristics of Virtual Switching.” In: *Proceedings of the 2014 IEEE 3rd International Conference on Cloud Networking (CloudNet)*. 2014, pp. 120–125. DOI: 10.1109/CloudNet.2014.6968979.

[52] Paul Emmerich et al. “Assessing Soft-and Hardware Bottlenecks in PC-based Packet Forwarding Systems.” In: *Fourteenth International Conference on Networks (ICN 2015)*. 2015.

[53] Lyndon Evans and Philip Bryant. “LHC Machine.” In: *Journal of Instrumentation* 3.08 (2008), S08001. URL: http://stacks.iop.org/1748-0221/3/i=08/a=S08001.

[54] *Facebook*. Online; accessed 2016-04-12. URL: facebook.com.

[55] Kevin R Fall and W Richard Stevens. *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley, 2011.

[56] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. “A Scalable, Commodity Data Center Network Architecture.” In: *SIGCOMM Comput. Commun. Rev.* 38.4 (2008), pp. 63–74. DOI: 10.1145/1402946.1402967.

[57] Mohammad Al-Fares et al. “Hedera: Dynamic Flow Scheduling for Data Center Networks.” In: *NSDI*. Vol. 10. 2010, pp. 19–19.

[58] Nick Feamster, Jennifer Rexford, and Ellen Zegura. “The Road to SDN: An Intellectual History of Programmable Networks.” In: *SIGCOMM Comput. Commun. Rev.* 44.2 (2014), pp. 87–98. DOI: 10.1145/2602204.2602219.

[59] Oliver Feuser and Andre Wenzel. “On the Effects of the IEEE 802.3x Flow Control in Full-Duplex Ethernet LANs.” In: *Proceedings of the 1999 Conference on Local Computer Networks (LCN)*. IEEE. 1999, pp. 160–161.

[60] Roy Thomas Fielding. “Architectural Styles and the Design of Network-Based Software Architectures.” PhD thesis. University of California, Irvine, 2000.

[61] Sebastian Gallenmüller et al. “Comparison of Frameworks for High-Performance Packet IO.” In: *Proceedings of the Eleventh ACM/IEEE Symposium on Architectures for Networking and Communications Systems*. ANCS ’15. IEEE Computer Society, 2015.

[62] Eric Gamess and Neudith Morales. “Peak Performance of TCP and UDP in IPv4 and IPv6 over Ethernet Networks.” In: *International Journal of Digital Content Technology and its Applications* 7.9 (2013), p. 519.

[63] Jim Gettys and Kathleen Nichols. “Bufferbloat: Dark Buffers in the Internet.” In: *Queue* 9.11 (2011), 40:40–40:54. DOI: 10.1145/2063166.2071893.

[64] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. “The Google File System.” In: *SIGOPS Oper. Syst. Rev.* 37.5 (2003), pp. 29–43. DOI: 10.1145/1165389.945450.

[65] Alex Goldhammer and John Ayer Jr. “Understanding Performance of PCI Express Systems.” In: *Xilinx WP350, Sept* 4 (2008).

[66] Paul Goransson and Chuck Black. *Software Defined Networks: A Comprehensive Approach*. Elsevier, 2014.

[67] Ian Gorton and Deborah K Gracio. *Data-Intensive Computing: Architectures, Algorithms, and Applications*. Cambridge University Press, 2012.

[68] Ernst Gunnar Gran et al. “First Experiences with Congestion Control in InfiniBand Hardware.” In: *Proceedings of the 2010 IEEE International Parallel & Distributed Processing Symposium (IPDPS)*. IEEE. 2010, pp. 1–12.

[69] Mikkel Hagen and Ryan Zarick. “Performance Evaluation of DCB’s Priority-Based Flow Control.” In: *Proceedings of the 10th IEEE International Symposium on Network Computing and Applications (NCA 2011)*. IEEE. 2011, pp. 328–333.

[70] S. Han et al. “Building a Single-Box 100 Gbps Software Router.” In: *Proceedings of the 2010 17th IEEE Workshop on Local and Metropolitan Area Networks (LANMAN)*. 2010, pp. 1–4. DOI: 10.1109/LANMAN.2010.5507157.

[71] C. Hopps. *Analysis of an Equal-Cost Multi-Path Algorithm*. RFC 2992. RFC Editor, 2000.

[72] *HP 6600 Switch Series*. Online; accessed 2016-07-21. URL: http://www.hp.com/.

[73] Jaehyun Hwang, Joon Yoo, and Nakjung Choi. “Deadline and Incast Aware TCP for Cloud Data Center Networks.” In: *Computer Networks* 68 (2014). Communications and Networking in the Cloud, pp. 20–34. DOI: http://dx.doi.org/10.1016/j.comnet.2013.12.002.

[74] *IEEE P802.3bs 400 Gb/s Ethernet Task Force*. Online; accessed 2016-06-27. URL: http://www.ieee802.org/3/bs/index.html.

[75] “IEEE Standard for Bridging & Management.” In: *IEEE Std 802.1Q* (2014).

[76] "IEEE Standard for Ethernet." In: *IEEE Std 802.3* (2012).

[77] *Insight Direct UK Limited*. Online; accessed 2016-11-14. URL: http://www.uk.insight.com/.

[78] *Intel 64 and IA-32 Architectures Optimization Reference Manual*. Online; accessed 2016-06-16. URL: http://www.intel.com/content/dam/doc/manual/64-ia-32-architectures-optimization-manual.pdf.

[79] *Intel 82599 10 GbE Controller Datasheet*. Revision 3.1. Intel. 2015.

[80] *Intel DPDK: Data Plane Development Kit*. Online; accessed 2016-03-01. URL: http://dpdk.org/.

[81] *Intel Ethernet Converged Network Adapters XL710 10/40 GbE*. Online; accessed 2016-11-14. URL: intel.com/content/www/us/en/ethernet-products/converged-network-adapters/ethernet-xl710.html.

[82] *Intel Ethernet Multi-host Controller FM10000 Family*. Online; accessed 2016-12-17. URL: http://www.intel.com/content/www/us/en/embedded/products/networking/ethernet-multi-host-controller-fm10000-family-overview.html.

[83] *Intel Product Specifications*. Online; accessed 2016-08-31. URL: http://ark.intel.com/.

[84] *Intel Server Board S2600GZ/GL Technical Product Specification*. Revision 1.1. Intel. 2012.

[85] *Intel Xeon Processor E5-1600/2600/4600 Product Families: Datasheet Vol. 1*. Online; accessed 2016-08-30. URL: http://www.intel.com/content/www/us/en/processors/xeon/xeon-e5-1600-2600-vol-1-datasheet.html.

[86] *Intel Xeon Processor E5-1600/2600/4600 v2 Product Families: Datasheet Vol. 1*. Online; accessed 2016-10-13. URL: http://www.intel.com/content/www/us/en/processors/xeon/xeon-e5-1600-2600-vol-2-datasheet.html.

[87] Grzegorz Jereczek, Giovanna Lehmann Miotto, and David Malone. "Analogues Between Tuning TCP for Data Acquisition and Datacenter Networks." In: *Proceedings of the 2015 IEEE International Conference on Communications (ICC)*. 2015, pp. 6062–6067. DOI: 10.1109/ICC.2015.7249288.

[88] Grzegorz Jereczek et al. "A Lossless Switch for Data Acquisition Networks." In: *Proceedings of the 2015 IEEE 40th Conference on Local Computer Networks (LCN)*. 2015. DOI: 10.1109/LCN.2015.7366370.

[89] Grzegorz Jereczek et al. "A Lossless Network for Data Acquisition." In: *IEEE Transactions on Nuclear Science* 64.6 (2017). DOI: 10.1109/TNS.2017.2682182.

[90] Grzegorz Jereczek et al. “Approaching Incast Congestion with Multi-host Ethernet Controllers.” In: *IEEE Conference on Network Function Virtualization and Software Defined Networks (IEEE NFV-SDN)*. Under review. 2017.

[91] J. Jin et al. “Network Architecture and QoS Issues in the Internet of Things for a Smart City.” In: *Proceedings of the 2012 International Symposium on Communications and Information Technologies (ISCIT)*. 2012. DOI: 10.1109/ISCIT.2012.6381043.

[92] *Kernel Parameters*. Online; accessed 2016-11-28. URL: https://www.kernel.org/doc/Documentation/kernel-parameters.txt.

[93] Wonho Kim et al. “Automated and Scalable QoS Control for Network Convergence.” In: *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking*. Vol. 10. USENIX Association, 2010, pp. 1–1.

[94] Eddie Kohler et al. “The Click Modular Router.” In: *ACM Trans. Comput. Syst.* 18.3 (2000), pp. 263–297. DOI: 10.1145/354871.354874.

[95] Thorsten Kollegger. “The ALICE High Level Trigger: The 2011 Run Experience.” In: *Proceedings of the 18th IEEE-NPSS Real Time Conference (RT 2012)*. IEEE. 2012, pp. 1–4.

[96] D. Kreutz et al. “Software-Defined Networking: A Comprehensive Survey.” In: *Proceedings of the IEEE* 103.1 (2015), pp. 14–76. DOI: 10.1109/JPROC.2014.2371999.

[97] Elie Krevat et al. “On Application-level Approaches to Avoiding TCP Throughput Collapse in Cluster-based Storage Systems.” In: *Proceedings of the 2nd International Workshop on Petascale Data Storage: Held in Conjunction with Supercomputing ’07*. PDSW ’07. ACM, 2007, pp. 1–4. DOI: 10.1145/1374596.1374598.

[98] Santosh Kulkarni, Prathima Agrawal, et al. *Analysis of TCP Performance in Data Center Networks*. Springer, 2014.

[99] Anurag Kumar, D Manjunath, and Joy Kuri. *Communication Networking*. Elsevier, 2004.

[100] Gary Lee. *Cloud Networking: Understanding Cloud-based Data Center Networks*. Morgan Kaufmann, 2014.

[101] Guoming Liu. “Management, Optimization and Evolution of the LHCb Online Network.” PhD thesis. Università degli Studi di Ferrara, 2010.

[102] Guoming Liu and Niko Neufeld. “DAQ Architecture for the LHCb Upgrade.” In: *Journal of Physics: Conference Series* 513.1 (2014), p. 012027.

[103] Maziar Manesh et al. “Evaluating the Suitability of Server Network Cards for Software Routers.” In: *Proceedings of the Workshop on Programmable Routers for Extensible Services of Tomorrow*. PRESTO ’10. ACM, 2010, 7:1–7:6. DOI: 10.1145/1921151.1921161.

[104] John D McCalpin. “Sustainable Memory Bandwidth in Current High Performance Computers.” In: *Silicon Graphics Inc* (1995).

[105] Edward McConnell. “Data Acquisition Systems.” In: *Measurement, Instrumentation, and Sensors Handbook; 2nd ed.* Ed. by John G Webster and Halit Eren. CRC Press, 2014. ISBN: 978-1-4398-4891-3. DOI: 10.1201/b15664-102. URL: http://dx.doi.org/10.1201/b15664-102.

[106] *MCi Online*. Online; accessed 2016-11-14. URL: http://shop.mcidiventi.co.uk/.

[107] N. McKeown. “A Fast Switched Backplane for a Gigabit Switched Router.” In: *Business Communications Review* 27.12 (1997).

[108] Nick McKeown et al. “OpenFlow: Enabling Innovation in Campus Networks.” In: *ACM SIGCOMM Computer Communication Review* 38.2 (2008), pp. 69–74.

[109] P. M. Mohan, D. M. Divakaran, and M. Gurusamy. “Performance Study of TCP Flows with QoS-supported OpenFlow in Data Center Networks.” In: *2013 19th IEEE International Conference on Networks (ICON)*. 2013, pp. 1–6. DOI: 10.1109/ICON.2013.6781936.

[110] V. Moreno et al. “Commodity Packet Capture Engines: Tutorial, Cookbook and Applicability.” In: *IEEE Communications Surveys Tutorials* 17.3 (2015), pp. 1364–1390. DOI: 10.1109/COMST.2015.2424887.

[111] R. Morris. “TCP Behavior with Many Flows.” In: *Proceedings oo the 1997 International Conference on the Network Protocols*. 1997, pp. 205–211. DOI: 10.1109/ICNP.1997.643715.

[112] *Myricom*. Online; accessed 2016-04-05. URL: www.myricom.com.

[113] Thomas D Nadeau and Ken Gray. *SDN: Software Defined Networks*. "O’Reilly Media, Inc.", 2013.

[114] David Nagle, Denis Serenyi, and Abbie Matthews. “The Panasas ActiveScale Storage Cluster: Delivering Scalable High Bandwidth Storage.” In: *Proceedings of the 2004 ACM/IEEE Conference on Supercomputing*. SC ’04. IEEE Computer Society, 2004, pp. 53–. DOI: 10.1109/SC.2004.57.

[115] J. Nagle. “On Packet Switches with Infinite Storage.” In: *IEEE Transactions on Communications* 35.4 (1987). DOI: 10.1109/TCOM.1987.1096782.

[116] A. Al-Najjar, S. Layeghy, and M. Portmann. "Pushing SDN to the End-Host, Network Load Balancing Using OpenFlow." In: *2016 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*. 2016, pp. 1–6. DOI: 10.1109/PERCOMW.2016.7457129.

[117] Andrea Negri. "Evolution of the Trigger and Data Acquisition System for the ATLAS experiment." In: *Journal of Physics: Conference Series* 396.1 (2012), p. 012033.

[118] N. Neufeld. "LHC trigger & DAQ - An Introductory Overview." In: *Proceedings of the 18th IEEE-NPSS Real Time Conference (RT 2012)*. IEEE. 2012, pp. 1–4.

[119] Niko Neufeld et al. "The LHCb Eventbuilder: Design, Implementation and Operational Experience." In: *IEEE Transactions on Nuclear Science* 58.4 (2011), pp. 1877–1884.

[120] *Next Generation Intel Microarchitecture (Nehalem)*. Online; accessed 2016-06-14. URL: http://www.intel.com/pressroom/archive/reference/whitepaper_Nehalem.pdf.

[121] Rajesh Nishtala et al. "Scaling Memcache at Facebook." In: *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*. NSDI'13. USENIX Association, 2013, pp. 385–398.

[122] Linda Null, Julia Lobur, et al. *The Essentials of Computer Organization and Architecture*. Jones & Bartlett Publishers, 2014.

[123] B. A. A. Nunes et al. "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks." In: *IEEE Communications Surveys Tutorials* 16.3 (2014), pp. 1617–1634. DOI: 10.1109/SURV.2014.012214.00180.

[124] *Open Networking Foundation*. Online; accessed 2016-03-01. URL: http://www.opennetworking.org/.

[125] *Open vSwitch*. Online; accessed 2016-06-14. URL: openvswitch.org.

[126] *OpenFlow Switch Specification*. Online; accessed 2016-06-01. URL: opennetworking.org/sdn-resources/technical-library.

[127] *OpenStack Open Source Cloud Computing Software*. Online; accessed 2017-01-10. URL: https://www.openstack.org/.

[128] Priscilla Oppenheimer. *Top-Down Network Design*. Cisco Press, 2010.

[129] D. P. Palomar and J. R. Fonollosa. "Practical Algorithms for a Family of Waterfilling Solutions." In: *IEEE Transactions on Signal Processing* 53.2 (2005). DOI: 10.1109/TSP.2004.840816.

[130] J G Panduro Vazquez. "The ATLAS Data Acquisition System: from Run 1 to Run 2." In: *Nuclear Physics B - Proceedings Supplements* (2015).

[131] Krzysztof Pawlikowski. "Steady-state Simulation of Queueing Processes: Survey of Problems and Solutions." In: *ACM Comput. Surv.* 22.2 (1990), pp. 123–170. DOI: 10.1145/78919.78921.

[132] Jonathan Perry et al. "Fastpass: A Centralized Zero-Queue Datacenter Network." In: *ACM SIGCOMM Computer Communication Review* 44.4 (2015), pp. 307–318.

[133] Larry L Peterson and Bruce S Davie. *Computer Networks: a Systems Approach*. Morgan Kaufmann, 2011.

[134] Ben Pfaff and Bruce Davie. *The Open vSwitch Database Management Protocol*. RFC 7047. RFC Editor, 2013.

[135] Ben Pfaff et al. "Extending Networking into the Virtualization Layer." In: *8th ACM Workshop on Hot Topics in Networks (HotNets-VIII)*. 2009.

[136] B. Pfaff et al. "The Design and Implementation of Open vSwitch." In: *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*. NSDI'15. USENIX Association, 2015, pp. 117–130.

[137] G. Pfister et al. "Solving Hot Spot Contention Using InfiniBand Architecture Congestion Control." In: *Proceedings of the HP-IPC 2005 Conference*. 2005.

[138] *PF_RING*. Online; accessed 2016-06-14. URL: http://www.ntop.org/products/pf_ring/.

[139] Amar 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. USENIX Association, 2008, 12:1–12:14.

[140] *Python*. Online; accessed 2016-10-20. URL: https://www.python.org/.

[141] Sven-Arne Reinemo, Tor Skeie, and Manoj K. Wadekar. "Ethernet for High-Performance Data Centers: On the New IEEE Datacenter Bridging Standards." In: *IEEE Micro* 30.4 (2010), pp. 42–51. DOI: http://doi.ieeecomputersociety.org/10.1109/MM.2010.65.

[142] Yongmao Ren et al. "A Survey on TCP Incast in Data Center Networks." In: *International Journal of Communication Systems* (2012).

[143] Luigi Rizzo. "Netmap: a Novel Framework for Fast Packet I/O." In: *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*. USENIX ATC'12. USENIX Association, 2012.

[144] Roberto Rojas-Cessa, Yagiz Kaymak, and Ziqian Dong. "Schemes for Fast Transmission of Flows in Data Center Networks." In: *IEEE Communications Surveys & Tutorials* 17.3 (2015).

[145] Alexey Rybalchenko et al. “Efficient Time Frame Building for Online Data Reconstruction in ALICE Experiment.” In: *Journal of Physics: Conference Series*. Vol. 664. 8. IOP Publishing. 2015, p. 082048.

[146] Franklin Saka. “Ethernet for the ATLAS Second Level Trigger.” PhD thesis. University of London, 2001.

[147] Cheryl A Schmidt. *Complete CompTIA A+ Guide to IT Hardware and Software*. Pearson IT Certification, 2016.

[148] Rainer Schwemmer and Niko Neufeld. “A 32 Terabit/s Data Acquisition from Mostly COTS Components.” In: *IEEE Transactions on Nuclear Science* 62.4 (2015), pp. 1747–1751.

[149] *Scientific Linux*. Online; accessed 2016-07-19. URL: https://www.scientificlinux.org/.

[150] Rich Seifert and Jim Edwards. *The All-New Switch Book: The Complete Guide to LAN Switching Technology*. John Wiley & Sons, 2008.

[151] *Snabb Switch*. Online; accessed 2016-06-14. URL: http://www.snabb.co.

[152] Stefan-Nicolae Stancu. “Networks for the ATLAS LHC Detector: Requirements, Design and Validation.” PhD thesis. Bucharest, Polytechnic Inst., 2005.

[153] Stefan Stancu et al. “The Use of Ethernet in the Dataflow of the ATLAS Trigger and DAQ.” In: *Proceedings of 13th International Conference on Computing in High-Energy and Nuclear Physics (CHEP 2003)*. 2003.

[154] W Richard Stevens, Bill Fenner, and Andrew Rudoff. *UNIX Network Programming*. Vol. 1. Addison-Wesley Professional, 2004.

[155] *Supermicro*. Online; accessed 2016-11-14. URL: https://www.supermicro.com.

[156] Y. Tamir and G. L. Frazier. “High-Performance Multi-Queue Buffers for VLSI Communications Switches.” In: *Proceedings of the 15th Annual International Symposium on Computer Architecture*. ISCA ’88. IEEE Computer Society Press, 1988, pp. 343–354.

[157] Andrew S. Tanenbaum and David J. Wetherall. *Computer Networks*. 5th. Prentice Hall, 2011. ISBN: 978-0-13-212695-3.

[158] Yuki Tanisawa and Miki Yamamoto. “QCN with Delay-Based Congestion Detection for Limited Queue Fluctuation in Data Center Networks.” In: *Proceedings of the IEEE 2nd International Conference on Cloud Networking (CloudNet 2013)*. IEEE. 2013, pp. 42–49.

[159] V. Tanyingyong, M. Hidell, and P. Sjödin. "Using Hardware Classification to Improve PC-Based OpenFlow Switching." In: *Proceedings of the 2011 IEEE 12th International Conference on High Performance Switching and Routing*. 2011. DOI: 10.1109/HPSR.2011.5986029.

[160] V. Tanyingyong, M. Hidell, and P. Sjödin. "Improving Performance in a Combined Router/Server." In: *2012 IEEE 13th International Conference on High Performance Switching and Routing*. 2012. DOI: 10.1109/HPSR.2012.6260827.

[161] The ALICE Collaboration. "The ALICE Experiment at the CERN LHC." In: *Journal of Instrumentation* 3.08 (2008). URL: http://stacks.iop.org/1748-0221/3/i=08/a=S08002.

[162] The ATLAS Collaboration. *ATLAS High-Level Trigger, Data-Acquisition and Controls: Technical Design Report*. Tech. rep. Geneva, 2003. URL: https://cds.cern.ch/record/616089.

[163] The ATLAS Collaboration. "The ATLAS Experiment at the CERN Large Hadron Collider." In: *Journal of Instrumentation* 3.08 (2008), S08003. URL: http://stacks.iop.org/1748-0221/3/i=08/a=S08003.

[164] The CMS Collaboration. "The CMS Experiment at the CERN LHC." In: *Journal of Instrumentation* 3.08 (2008), p. 08004. URL: http://stacks.iop.org/1748-0221/3/i=08/a=S08004.

[165] *The Intel-CERN European Doctorate Industrial Program*. Online; accessed 2016-02-11. URL: http://openlab.web.cern.ch/ice-dip.

[166] The LHCb Collaboration. "The LHCb Detector at the LHC." In: *Journal of Instrumentation* 3.08 (2008), S08005. URL: http://stacks.iop.org/1748-0221/3/i=08/a=S08005.

[167] *The OpenDaylight Platform*. Online; accessed 2016-10-20. URL: https://www.opendaylight.org/.

[168] *TOP500 Supercomputing Sites*. Online; accessed 2016-03-14. URL: http://top500.org.

[169] J. Touch and R. Perlman. *Transparent Interconnection of Lots of Links (TRILL): Problem and Applicability Statement*. RFC 5556. RFC Editor, 2009.

[170] Subir Varma. *Internet Congestion Control*. Morgan Kaufmann, 2015.

[171] Vijay Vasudevan et al. "Safe and Effective Fine-grained TCP Retransmissions for Datacenter Communication." In: *SIGCOMM Comput. Commun. Rev.* 39.4 (2009), pp. 303–314. DOI: 10.1145/1594977.1592604.

[172] Arun Vishwanath, Vijay Sivaraman, and Marina Thottan. "Perspectives on Router Buffer Sizing: Recent Results and Open Problems." In: *SIGCOMM Comput. Commun. Rev.* 39.2 (2009), pp. 34–39. DOI: 10.1145/1517480.1517487.

[173] *Vyatta vRouter*. Online; accessed 2016-03-01. URL: http://www.brocade.com.

[174] Haitao Wu et al. "ICTCP: Incast Congestion Control for TCP in Data Center Networks." In: *Proceedings of the 6th International COnference*. Co-NEXT '10. ACM, 2010, 13:1–13:12. DOI: 10.1145/1921168.1921186.

[175] Yukai Yang et al. "Staggered Flows: An Application Layer's Way to Avoid Incast Problem." In: *Proceedings of the IEEE Asia Pacific Cloud Computing Congress (APCloudCC 2012)*. 2012, pp. 64–67. DOI: 10.1109/APCloudCC.2012.6486513.

[176] H. Yun et al. "MemGuard: Memory Bandwidth Reservation System for Efficient Performance Isolation in Multi-Core Platforms." In: *Proceedings of the 2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium*. 2013, pp. 55–64. DOI: 10.1109/RTAS.2013.6531079.

[177] Y. Zhang and N. Ansari. "On Architecture Design, Congestion Notification, TCP Incast and Power Consumption in Data Centers." In: *IEEE Communications Surveys Tutorials* 15.1 (2013), pp. 39–64. DOI: 10.1109/SURV.2011.122211.00017.

[178] Yan Zhang and Nirwan Ansari. "On Mitigating TCP Incast in Data Center Networks." In: *Proceedings of the 2011 IEEE INFOCOM Conference*. IEEE. 2011, pp. 51–55.

[179] Dong Zhou et al. "Scalable, High Performance Ethernet Forwarding with CuckooSwitch." In: *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*. CoNEXT '13. ACM, 2013, pp. 97–108. DOI: 10.1145/2535372.2535379.

[180] Yibo Zhu et al. "Congestion Control for Large-Scale RDMA Deployments." In: *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. ACM. 2015, pp. 523–536.

[181] T. Zinner et al. "Dynamic Application-Aware Resource Management Using Software-Defined Networking: Implementation Prospects and Challenges." In: *2014 IEEE Network Operations and Management Symposium (NOMS)*. 2014, pp. 1–6. DOI: 10.1109/NOMS.2014.6838404.

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

https://bitbucket.org/amiede/classicthesis/

Happy users of `classicthesis` usually send a real postcard to the author, a collection of postcards received so far is featured here:

http://postcards.miede.de/

Final Version as of June 27, 2017 (`classicthesis`).