A Lossless Network for Data Acquisition

Grzegorz Jereczek, Giovanna Lehmann Miotto, David Malone, and Miroslaw Walukiewicz
on behalf of ATLAS Trigger and Data Acquisition

***Abstract*—The bursty many-to-one communication pattern, typical for data acquisition systems, is particularly demanding for commodity TCP/IP and Ethernet technologies. We expand the study of lossless switching in software running on commercial-off-the-shelf servers, using the ATLAS experiment as a case study. In this paper we extend the popular software switch, Open vSwitch, with a dedicated, throughput-oriented buffering mechanism for data acquisition. We compare the performance under heavy congestion on typical Ethernet switches to a commodity server acting as a switch. Our results indicate that software switches with large buffers perform significantly better. Next, we evaluate the scalability of the system when building a larger topology of interconnected software switches, exploiting the integration with software-defined networking technologies. We build an IP-only leaf-spine network consisting of eight software switches running on distinct physical servers as a demonstrator.**

***Index Terms*—Ethernet networks, computer network management, software defined networking, packet switching, packet loss, TCPIP, data acquisition.**

I. Introduction

ATLAS [1] is a general-purpose particle detector designed to study particle collisions at the Large Hadron Collider (LHC) at CERN. One of the key components of ATLAS, and other large experiments, is the data acquisition (DAQ) network. It collects the outputs from all the instruments to facilitate reconstruction of a physical process. 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. Will commodity TCP/IP and Ethernet technologies in their current form still be able to effectively adapt to the bursty traffic without losing packets knowing the scarcity of buffers in the traditional networking hardware? This concern can be alleviated. There are reasons to evaluate software switching on commercial-off-the-shelf (COTS) servers to provide reliable transport in congested conditions. The flexibility of design in software, performance of modern computer platforms, and buffering capabilities constrained solely by the amount of DRAM memory are a strong basis on which to build a DAQ network. Furthermore, this is possible with commodity technologies. Hence we have started our work on lossless software switching in [2] and now we extend the study.

In this paper we show that building a high-bandwidth lossless network for DAQ based on software switches is feasible and they are a viable solution for future small- and large-scale systems based on commodity TCP/IP and Ethernet. Our results indicate that software switches with large buffers perform significantly better than some typical switches under heavy congestion. In our evaluation in Section IV-C, a server with twelve 10 Gbps Ethernet interfaces acting as a prototype switch reaches its maximum bandwidth of 120 Gbps. It completely avoids throughput degradation, while hardware switches reach no more than 85 % of the requested load under similar conditions. We also show how a number of software switches can be interconnected to build terabit networks. We highlight aspects such as bandwidth scaling, management, and load balancing. In this context, we discuss the usability of Software-Defined Networking (SDN) technologies to centrally manage and optimize a data acquisition network.

We continue to use ATLAS as a case study. Its DAQ network, having demanding traffic characteristics, is a good environment to evaluate candidate technologies. The conclusions are, however, not limited to ATLAS. They apply to other networks susceptible to the problems arising from the many-to-one pattern as well as other data acquisition systems.

This paper is structured as follows. Section II gives an overview of DAQ networks and the incast congestion problem, followed by a brief introduction to software switching and SDN. Section II concludes with related work. We discuss our extensions to Open vSwitch in Section III and analyse single switch performance in an all-to-all incast scenario in Section IV. Scaling to larger topologies with centralized control is presented in Section V. We conclude our work in Section VI.

II. Background and motivation

A. DAQ networks and many-to-one communication

In LHC nomenclature, particle collisions inside a detector are referred to as *events*. Their physical “fingerprints” are recorded by the detector, which is the data source for the DAQ system. Different event fragments corresponding to an event are usually buffered on multiple readout units (commodity PCs in ATLAS), which constitute the Readout System (ROS). The DAQ network connects the data sources, being the ROS, to the data sinks, being a computer farm which builds[1], processes and filters the events (HLT — High Level Trigger). Every HLT node requests fragments of a particular event from the ROS (*pull architecture*[2]). A number of independent worker

Manuscript received June 22, 2016; revised January 17, 2017.

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).

G. Jereczek is with CERN, Geneva, Switzerland and with Maynooth University, Maynooth, Ireland (e-mail: grzegorz.jereczek@cern.ch).

G. Lehmann Miotto is with CERN, Geneva, Switzerland.

D. Malone is with Maynooth University, Maynooth, Ireland.

M. Walukiewicz is with Intel Corporation, Gdansk, Poland.

[1] Event building is the physical transfer of event data to one location.

[2] Pull architecture means that the destinations request data from the readout units. In contrast, in the push architecture the sources send the data whenever they are ready [3].

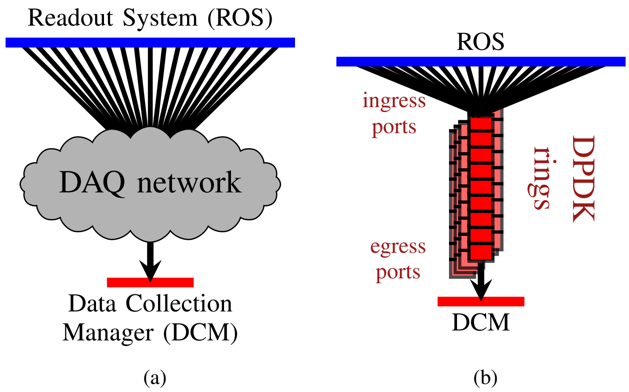


Fig. 1. Many-to-one communication in a data acquisition network (a). Data originating from the experiment's instruments are sent over a network to data collectors. Only one collector is drawn for clarity. Depending on the size of an experiment, hundreds and thousands of independent collectors are used in a large filtering farm (HLT). The proposed buffering mechanism providing a dedicated queue for each data collector is shown in (b).

processes run on a single filtering node working on different events depending on the number of available CPU cores. However, there is only one process (DCM — Data Collection Manager) per node that handles all the communication on behalf of the processing units.

In DAQ, 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 [3]. These conditions lead to *incast* congestion — a problem that is present also in the IT industry, especially in datacenter networks [4]. It occurs when multiple nodes respond with data synchronously to a single requester, as pictured in Fig. 1a. It has been observed that these responses, although not very large on average, suffer from a high packet loss rate [5]. The scale of the problem increases with the number of responders, i.e. the size of the readout system in case of DAQ networks.

This incast pattern is particularly demanding for Ethernet switches and TCP/IP-based communication. The switch ports connected to data collectors are oversubscribed by the data sent from many readout units. Switches with insufficient buffers must drop some or most of the packets. The problem is systematic, so the built-in TCP retransmission mechanisms are not a solution [3]. They ensure reliability, but with a cost in performance. In an ideal situation, the network should adapt to the traffic and avoid congestion before it happens. One way of achieving this is to provide enough buffering space within the network, to accommodate traffic spikes and bursts without the need for discarding data [6]. Large experiments, like those of the LHC, apply traffic shaping in order to cope with the inherent instantaneous oversubscription, but they still require particularly large packet buffers. For example, there are ~100 ROS nodes with 4x10GbE links and ~2000 HLT nodes with 1xGbE link to the network. Only expensive, high-end switches or routers can be taken into consideration for use [3], but they cannot be regarded as commodity. Furthermore, the current trend in the industry is to provide devices with smaller buffers [7] and not with deeper ones as the scaling needs of the experiments would require. The lack of buffering is the main obstacle in the use of a simple push architecture in data acquisition [3], [8].

B. Software switching and software-defined networking

With recent advances in commodity networking a potential solution to our problem is to design a data acquisition network mixing switches with servers that act themselves as networking devices — *software switches*. A modern server with multiple Ethernet ports connected over the PCI Express (PCIe) bus gives the opportunity to provide large 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. In [2] we showed that a specialized switch can indeed operate without packet loss while maintaining high throughput and avoiding incast congestion. This was without any controls on the injected traffic, beyond that arising naturally in a DAQ application. This prototype software switch provided 120 Gbps bandwidth. It was based on a dedicated application using the DPDK framework [9] for fast packet processing in software. In this work we extend Open vSwitch (OvS) [10] with a similar DAQ-oriented buffering mechanism and compare its performance to traditional ToR (top of rack) switches. OvS is a production quality virtual switch with built-in support for DPDK, which provides features and protocol support typical for conventional switches. The OpenFlow (OF) protocol [11] and Open vSwitch Database (OVSDB) protocol [12] are also supported, which we use to enable the centralized network control and optimization.

Depending on the size of a DAQ system, multiple interconnected software switches are required to provide full connectivity. Today, the classical architecture with two deep-buffered large routers at the core of the network is used to interconnect approximately 2000 nodes in the ATLAS experiment [8]. In this paper we study a topology based on specialized software switches, possibly combined with traditional ToR switches with small buffers as an alternative for the current architecture.

No less important than throughput are aspects such as administration, configuration, fault tolerance and load balancing. The SDN paradigm [13] opens a completely new perspective for building, managing, and optimizing networks. In SDN the networking control and forwarding planes are physically decoupled and the network intelligence is centralized 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 [14]. OpenFlow (originally proposed in [15]) and OVSDB are used for communication between the controller and the devices. This concept contrasts with traditional networks using distributed control planes, in which network devices are comprised of both: a data plane, being a switch fabric connecting ports, and a control plane that is the brain of a device, implementing various protocols [14]. This SDN controller could be integrated with an already existing DAQ control framework of an experiment. We demonstrate in this paper how the entire solution based on SDN, TCP/IP protocol stack, and packet processing in software can be optimized as a whole to provide the maximum performance of a DAQ network.

C. Related work

The majority of approaches to incast congestion focus on control of the packet injection rate so as not to overwhelm buffers in the network [6], [8], [16]–[18]. All of them have means to improve, to some extent at least, the performance of DAQ and other incast-sensitive networks. However, these mechanisms, flow control in particular, do not work well in DAQ systems characterized by *constant instantaneous overload*, unless the network is significantly overprovisioned, because they are designed to absorb fluctuations only [3]. Our approach differs in that there is no rate control required on the sender side because there is enough buffering, which leads to both optimal operation and simplification of the entire system.

In this work we argue that this alternative approach for many-to-one communication networks, large network buffers in commodity servers, can be considered as a cost-effective, flexible, and scalable solution. The main building blocks of our proposal are software packet processing and SDN. We 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. Review of the literature related to fast packet processing on commodity servers as well as the preliminary discussion of the applicability of this approach in DAQ is available in our previous paper [2].

Details of the architecture of the ATLAS data acquisition system are discussed in [8], [19]. Characteristics of the DAQ traffic patterns on, for example, the LHCb experiment at CERN can be found in [20]. An example of an InfiniBand-based DAQ network is the CMS experiment [21].

III. An OvS-based lossless switch for DAQ

In [2] we proved 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. In this section we explore whether the same can be achieved with OvS and the OF and OVSDB protocols.

A. Design

The core of the idea behind the lossless software switch, which we proposed in [2], can be summarized 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 Fig. 1b).
- A mechanism that maps the packets destined to a particular data collector into its queue.

In our original design in [2] all of these items were implemented in a dedicated switching application using the DPDK framework.

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 to [2]. 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 the parallel spine-leaf topology with multipath in Section V.

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).

3) Alternative approach: OpenFlow version 1.0.0 [11] 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.

B. Implementation

We added 725 and removed 11 lines of code in five files of the OvS release 2.4.0 (with DPDK 2.0.0) to implement the daqring device and optimize the switch for high throughput. For the SDN controller we used the OpenDaylight project release Lithium [22]. We used the REST API [23] and Python [24] to automate the installation of the ROS-DCM flows on the switch.

Some optimizations to the OvS code were required to reach the full performance in our DAQ scenario (for details on OvS's design see [25]). The critical modifications included:

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

In order to achieve the highest performance of our switch, we follow the DPDK recommendations, especially with regard to NUMA-aware (Non-Uniform Memory Access) object allocation in memory and the use of huge page tables. The send and receive operations are performed by multiple CPU cores, which are configured, where possible, to match the NUMA-node of the corresponding network interface.

IV. Evaluation

We evaluate our implementation of the lossless OvS-based software switch with a small-scale DAQ setup using the ATLAS DAQ software in emulation mode. We use the same configuration as in [2] to allow comparison with the performance of our first prototype.

A. Evaluation setup

The DuT (Device under Test) is a server running a single instance of the lossless OvS-based software switch on 64-bit Fedora 20, kernel version 3.18.7-100. We use one of the server boards based on the Intel C602 chipset, S2600GZ [26], with two Intel Xeon EP-2680 eight-core CPUs. A total of 128 GiB DDR3 memory is available (64 GiB per CPU socket). We equipped the platform with six dual-port 10GbE cards (Intel 82599 Ethernet controller [27]) providing a total bandwidth of 120 Gbps (Fig. 2).

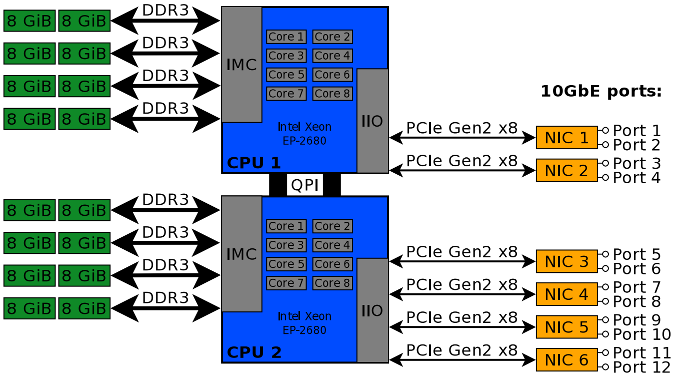


Fig. 2. Hardware topology of a server being used as a network switch. IMC stands for the Integrated Memory Controller, whereas IIO is the Integrated I/O Module. Six dual-port Network Interface Cards (NIC) provide a total bandwidth of 120 Gbps.

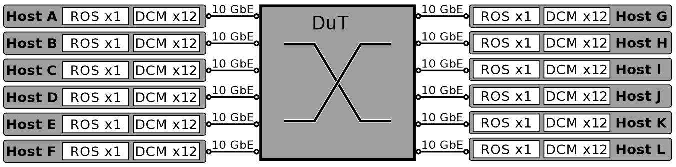


Fig. 3. The emulated data taking configuration for the performance evaluation of the software switch in the all-to-all incast scenario.

For traffic generation we use the ATLAS DAQ/HLT software in emulation mode running on twelve hosts connected with unshared 10GbE link directly to the DuT (Fig. 3). The ROS subsystem generates dummy data, while the DCMs request the fake events from the ROS. No event processing is performed on the nodes and so the network subsystem, which in this case is solely the DuT, can be analysed in isolation from other factors, such as processing latency. We emulate 12 DCMs and a single ROS application on every host to create a demanding all-to-all incast scenario. Each ROS provides a single event fragment of size 128 KiB. Total event size is 1.5 MiB. A single DCM does not request another event before it receives all fragments of the previous event from all available ROSes. All tests are performed with an MTU of 1500 B (for details of MTU choice please refer to [2]).

Data collection latency of a single event is understood as the timespan between sending the first request to the ROS and receiving the last fragment from the ROS. Goodput is commonly referred to as the application-level throughput — the raw event data bandwidth excluding all protocol overheads (Ethernet, TCP/IP, ATLAS protocol).

In this evaluation we disable dynamic TCP congestion control in all ROS hosts and instead use a static sender congestion window [4]. The window is set to a very large value so that each ROS is not rate limited by TCP, further increasing the incast effect. This allows us to evaluate the performance of our prototype without the influence of congestion control and test in the best DAQ scenario, pushing simply all the available data on to the wire. In other words, there is no traffic injection control and specialist optimizations discussed in our earlier study [4] are not in place for the experiments in this paper.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNS.2017.2682182, IEEE Transactions on Nuclear Science

Values for latency, throughput, and bandwidth are averages over approximately 120 s in all cases.

B. Achievable goodput and latency

Fig. 4 gives the comparison of the performance (goodput and latency) between the DPDK DAQ-dedicated application from [2] and the OvS with our optimizations:

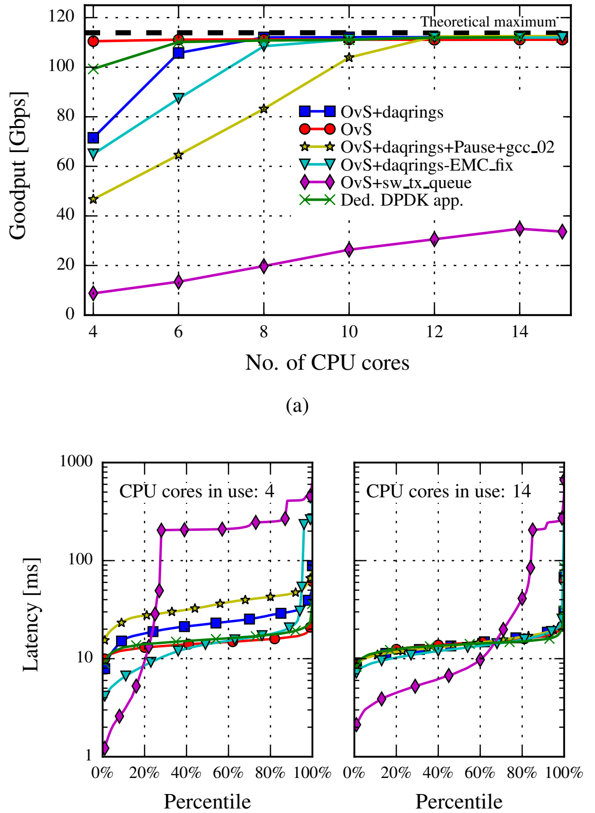
- *OvS+sw_tx_queue.* OvS without daqrings, and with the default OvS software tx-queue. Reaches only 30.6 %.
- *OvS+daqrings-EMC_fix.* OvS with daqrings and all optimizations, except the one for the EMC. Reaches 98.3 % with at least ten CPU cores.
- *OvS+daqrings+Pause+gcc_02.* OvS with daqrings and all optimizations, but also with IEEE 802.3x pause frame [28] and the default GCC optimization level. Reaches 98.8 % but requires twelve CPU cores. Pause frames eliminate packet drops for lower core counts, but increase the mean latency (plot on the left-hand side in Fig. 4b).
- *OvS.* OvS with all optimizations, 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 optimizations. Reaches 98.6 % of the theoretical goodput. There is significant performance degradation for less than eight CPU cores.
- *Ded. DPDK app.* Reaches 98.4 % 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 except *OvS+sw_tx_queue*. When using only four CPU cores the median increases for *OvS+daqrings*, and even more so for *OvS+daqrings+Pause+gcc_02*.

The results show that the optimized OvS delivers near maximum performance because of the large buffering capabilities. Using daqrings, performance increases further slightly. We predict that this difference will increase for larger topologies, which we will discuss in Section V. Daqrings also 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 buffers (e.g. see Section V-C). *OvS+daqrings* requires, however, more CPU cores than pure optimized 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.

C. Comparison with ToR switches

We continue our evaluation by comparing the DAQ-optimized OvS with two 10GbE ToR switches (shallow buffers) from different vendors in the same all-to-all incast scenario (the DuT in Fig. 3 acts as one of the ToRs). The results in Fig. 5a show that 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. TCP timeouts are clearly visible in the latency distribution in Fig. 5b with substantial numbers of event collection latencies exceeding 200 ms. Enabling Ethernet pause frame (and disabling TCP congestion control again) significantly improves performance, but they still reach only 64 % and 85 % compared to nearly 100 % achieved by OvS. Timeouts are avoided, but the latencies are still significantly higher than those of the 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.



(a)

(b)

Fig. 4. Goodput (a) and latency distribution (b) in all-to-all incast scenario with modified Open vSwitch.

D. Energy consumption

Fig. 6 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 (same switches as in the previous evaluation) were taken from their data sheets, whereas the ones for OvS are plotted for different CPU frequencies and CPU core counts and correspond to full achievable load (at least 95 % in all cases).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNS.2017.2682182, IEEE Transactions on Nuclear Science

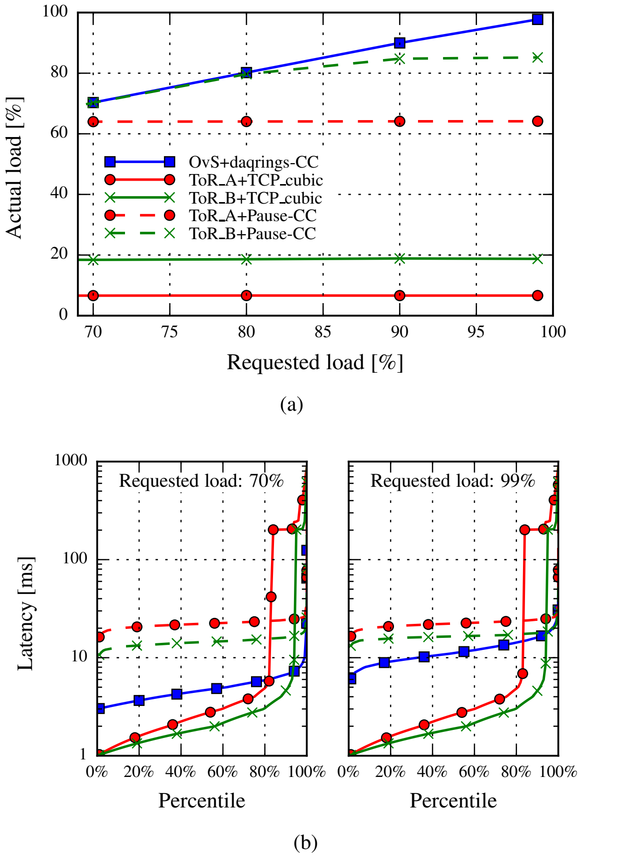


Fig. 5. Performance comparison of DAQ-optimized Open vSwitch with two ToR switches in all-to-all incast scenario. Achieved load in (a) and latency distribution in (b). *-CC* means that TCP congestion control is not used.

Although the server running OvS provides better performance, the power consumption is higher, especially when compared with ToR A. The situation can be improved in two 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. Also, low power CPUs could be considered, if providing enough performance. Second, remaining unused CPU cores could be used to perform different tasks, e.g. event filtering or experiment readout, optimizing the utilization of the entire system. We provide an initial evaluation in the following section.

E. 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 STREAM benchmark [29] 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 of our evaluation setup. The results are presented in Fig. 7. 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. For a real application, careful analysis and tuning would be required to guarantee the desired level of performance.

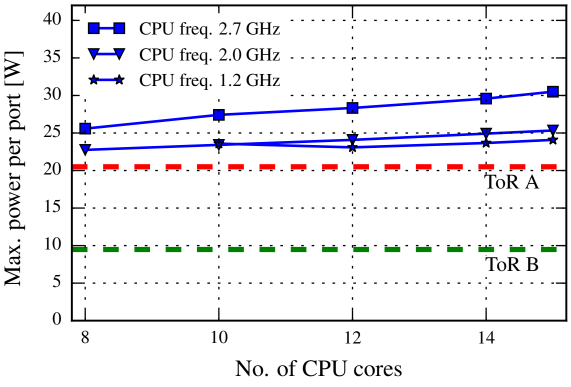


Fig. 6. Average per port power consumption in all-to-all incast scenario with DAQ-optimized Open vSwitch for various CPU frequencies. In all cases goodput is at least 95 % of the theoretical maximum.

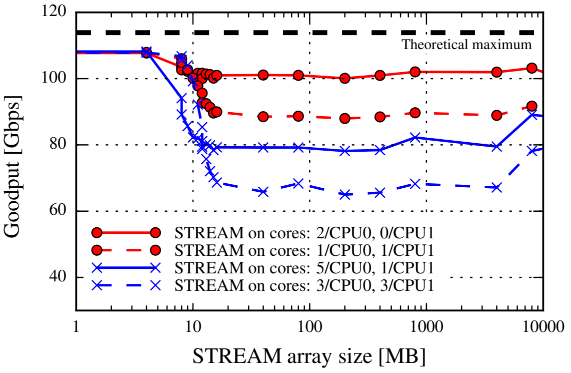


Fig. 7. Goodput in all-to-all incast scenario, if some of the unused CPU cores are used to run the STREAM benchmark.

V. A LOSSLESS NETWORK FOR DAQ

The results presented in the previous section are based on a small-scale prototype with twelve 10GbE ports. Bandwidth-wise the offered performance is already comparable with the requirements of ATLAS in the first data acquisition period, five years ago. Nevertheless, a DAQ network requires large port density for full connectivity between the readout system and filtering farm. In ATLAS it is currently provided by the classical architecture with two large routers at the core of the network [8]. In this section we investigate whether this classical topology could be replaced by a number of interconnected software switches. This architecture will have to sustain throughput that will be two or more orders of magnitude larger for the future upgrades of the experiments of the LHC. We also study other important aspects such as management and load balancing.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNS.2017.2682182, IEEE Transactions on Nuclear Science

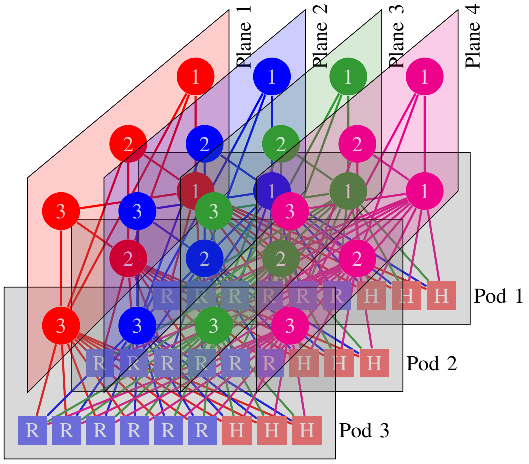


Fig. 8. A DAQ network in the parallel spine-leaf topology.

A. Parallel spine-leaf topology

Instead of combining hundreds of ports in a single core device, we evaluate the concept of the spine-leaf topology, popular in datacenters [30]. It is built with a large number of devices, but of a lower port density. In the context of DAQ, spine-leaf has been evaluated by the ALICE experiment [31]. The authors emphasized good scalability and lower vulnerability of the network. 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 proposed lossless software switches or a combination of software and ToR switches to overcome the incast congestion problem. The former are used to built a lossless spine-leaf network and the latter connect, with slower links, the leaf layer to the end-nodes. Lossless operation and high performance is achieved by using traffic shaping at daqrings. Additionally, pause frames can be used to absorb temporal fluctuations, other than incast congestion. In general, with appropriate traffic shaping at the software switches in the leaf layer, the spine layer could be built with ToR switches in order to improve port density.

When designing the topology we need to take into account the fact that in general the readout nodes can have multiple links to the network[3]. We follow a similar approach as in the datacenter fabric designed by Facebook [32], [33]. The basic network unit, a *pod*, is in our case a subset of ROS nodes (R) and filtering racks (H). A single filtering rack is a ToR switch that connects multiple servers. Each server runs a data collector and the multiple event filtering processes like in the normal DAQ system. The pod's nodes are connected to multiple switches – leaf switches. Each pod's switch connects to an independent *plane* — spine-leaf plane. The pods are interconnected in every plane by an independent spine-leaf plane. We refer to this topology as *parallel spine-leaf topology* because the planes provide independent connectivity. The ROS nodes and HLT ToRs can connect to a number of planes depending on the number of available uplinks and bandwidth requirements. An example of this topology with three pods, six ROSes and three filtering racks in a pod, and four planes is depicted in Fig. 8.

B. Bandwidth scaling

From the viewpoint of a DAQ system it is important to estimate the offered bandwidth of the proposed topology. Normally, the bisection bandwidth 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. Traditional approaches include link aggregation groups (LAG, IEEE802.3ad [28]) and equal-cost multi-path (ECMP, IEEE802.1Qbp [34]) that use hashing to distribute packets across available links. In our proposal, however, we can profit from the fact that we possess the knowledge about the entire network and the specific traffic pattern. We can use the SDN controller and the OF protocol to monitor and optimally distribute the traffic across available paths. In addition, constant monitoring of the path status can be implemented and used to provide fault tolerance. In case of failure, ROS-DCM flows can be redistributed. Because the number of paths between readout and collectors is large, a single link failure or even switch failure should not introduce any significant performance degradation.

The most obvious approach to optimize the network is to assign all flows of a particular DCM to use a single plane and a single spine switch in that plane. An example is given in Fig. 9. A DCM in the second HLT rack 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. All DCMs can be then spread across all available planes and spine switches, so 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.

In the following we will derive the theoretical bandwidth of the DAQ network under these assumptions. This approach can be considered as an example of the waterfilling algorithm, well-known in communication [35]. The DAQ bandwidth can be optimized by assigning flows to particular paths depending on the available capacity.

The total DAQ bandwidth can be calculated as follows:

$$B_{\mathrm{DAQ}} = n_{\mathrm{p}} \cdot n_{\mathrm{H}} \cdot n_{\mathrm{R}} \cdot B_{\mathrm{RH}} \tag{1}$$

with n_{p} as the number of planes, n_{H} as the total number of HLT racks, n_{R} as the total number of readout nodes, and B_{RH} as 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:

$$B_{\mathrm{RD}} = \min\left(\frac{b}{n_{\mathrm{R}}}, \frac{b}{n_{\mathrm{H}}}, B_{\mathrm{RH_{interpod}}}\right) \tag{2}$$

The base bandwidth of a single link is denoted with b. For the flows traversing the spine layer, the bandwidth is limited by

[3] In the present ATLAS DAQ network ROS PC's are directly connected to the core with four 10GbE links [8].

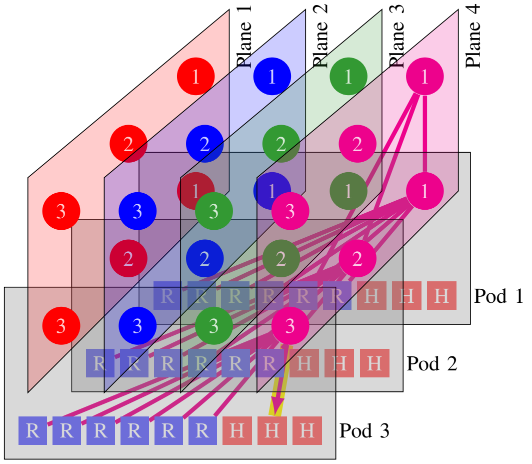


Fig. 9. The method for load balancing by assigning a particular plane and spine switch for all flows belonging to a DCM.

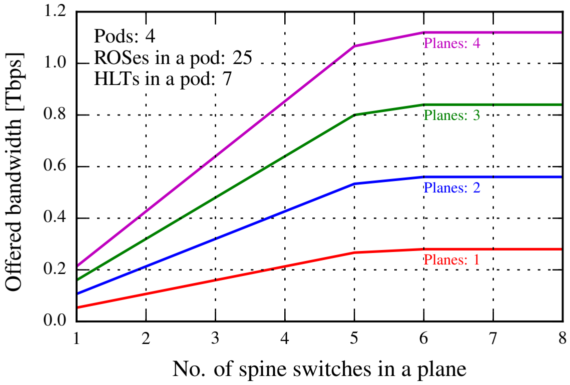


Fig. 10. An example for bandwidth scaling for a DAQ network in the parallel leaf-spine topology.

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:

$$B_{\mathrm{RH}_{\mathrm{interpod}}} = \frac{b \cdot n_{\mathrm{S}}}{n_{\mathrm{R}_{\mathrm{pod}}} \cdot n_{\mathrm{H}_{\mathrm{pod}}} \cdot (n_{\mathrm{pod}} - 1)} \qquad (3)$$

with n_{S} as the number of spine switches in a plane, $n_{\mathrm{R}_{\mathrm{pod}}}$ as the number of readout nodes in a pod, $n_{\mathrm{H}_{\mathrm{pod}}}$ 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 can be written as:

$$n_{\mathrm{R}} = n_{\mathrm{R}_{\mathrm{pod}}} \cdot n_{\mathrm{pod}}, \; n_{\mathrm{H}} = n_{\mathrm{H}_{\mathrm{pod}}} \cdot n_{\mathrm{pod}} \qquad (4)$$

These simple equations can give a quick estimate of the offered DAQ bandwidth for various configurations. An example for a DAQ system with 100 readout nodes, 28 HLT racks, and 10 Gbps of base link bandwidth is given in Fig. 10. 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.

C. Evaluation

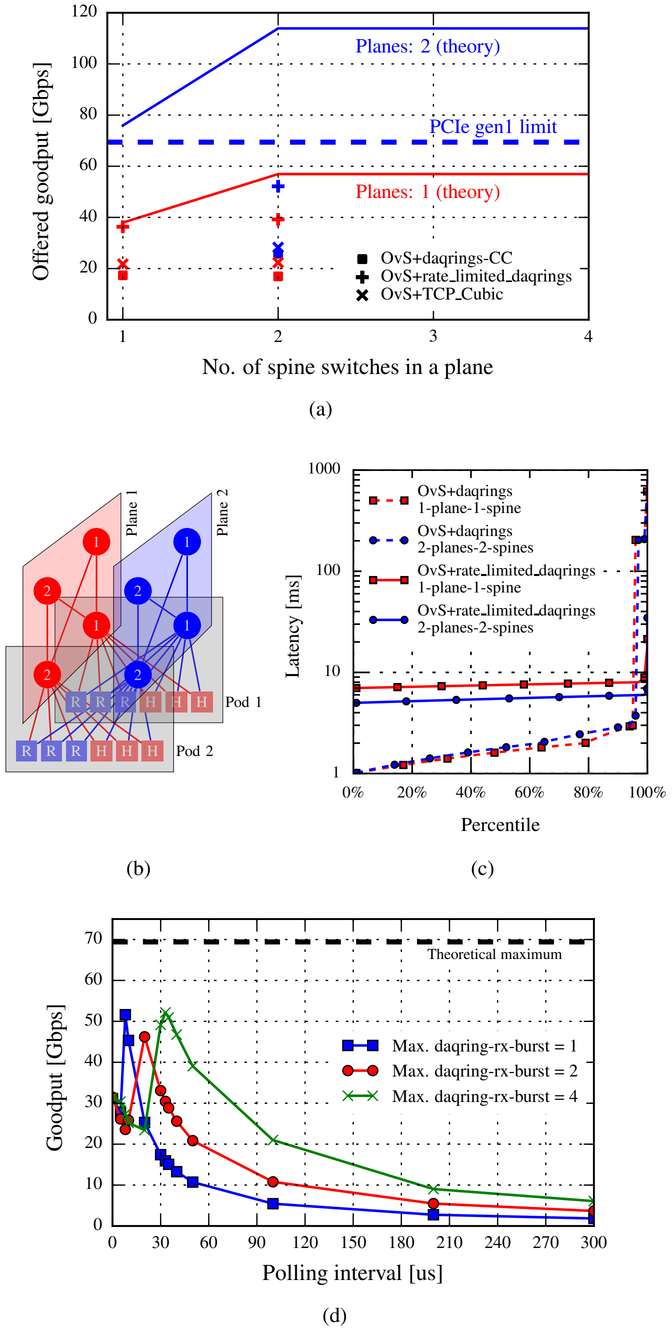
In order to evaluate the potential usage of the proposed design in building a centrally managed DAQ network with DAQ-optimized Open vSwitches we prepared an IP-only parallel leaf-spine network consisting of eight software switches running on separate physical servers (Fig. 11b). The servers acting as switches follow the same specification as in Section IV. 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.

We continue to use OvS optimizations described in Section III-B. The OpenDaylight's REST API is used to distribute the flows across available paths with the algorithm presented in Section V-B. All flows are based solely on IP addresses and TCP port numbers, so the switches are not using layer-2 MAC addresses to forward packets. ROSes and DCMs run on separate physical hosts (12 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. 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 [32], using traditional network protocols with ECMP routing and flow-based hashing.

Fig. 11a shows that the achieved goodput is far from the theoretical values calculated using the equations derived in V-B. 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.

This allows us to evaluate the use of per-daqring rate limitation. 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 III. Both the burst size and the polling cycle can be tuned as illustrated in Fig. 11d. Careful fine-tuning gives us the best combination of those values, which eliminates packet drops (lossless operation) and maximizes the goodput by adapting to the particular limitations of the PCIe/NIC combination. We can now compare the performance of various configurations (Fig. 11a, 11c):

1) *1-plane-1-spine.* OvS with rate-limited daqrings reaches 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) *2-planes-1-spine.* The goodput of the OvS with rate-limited daqrings is slightly higher than above, but lower than the theoretical value. 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.



(a)

(b)

(c)

(d)

Fig. 11. Performance of the prototype parallel leaf-spine topology (b) in many-to-many incast scenario. Achieved goodput in (a), latency distribution in (c), and tuning of the rate-limited daqrings in (d). Theoretical maximum is limited by the PCIe gen1 bandwidth and the on-chip memory of the network cards at the end hosts.

3) *2-planes-2-spines.* Goodput of all configurations improves. All possible paths through the network are used. OvS with rate limitation gives the best goodput (about 75 % of the theoretical PCIe gen1 limit), and 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.

While limitations of our edge hosts prevent us reaching theoretical throughput limits, the results still confirm that our design can optimize the performance of the entire network and provide lossless operation, even when all event data are simply pushed from the ROS. We also see that relying on the default TCP Cubic end-to-end congestion control results in significantly lower performance.

Our main goal in this experiment was to demonstrate how to build a centrally managed DAQ network using the OF and OVSDB protocols to control and optimize the traffic. This initial evaluation confirms that this approach is feasible.

VI. Conclusion

In this paper we proposed a novel approach to build and manage data acquisition networks based on commodity TCP/IP and Ethernet technologies with the main objective of providing lossless operation under incast congestion for a known data collection configuration. The nearly limitless memory of a software switch coupled with the flexibility of a software network controller allows us to design a dedicated server-based software switch with enormous packet buffers. Taking advantage of the DAQ-optimized parallel spine-leaf topology this software switch could be considered as a potential replacement for the expensive feature-rich core routers in the future upgrades of the experiments at the LHC and other DAQ systems.

First, we optimized the production quality Open vSwitch for DAQ-specific traffic characteristics and implemented our own queueing mechanism. We 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.

Second, we demonstrated that a potential topology of interconnected software switches can scale to higher bandwidths. For this topology, we proposed and evaluated on real hardware a method to manage and optimize the network using software-defined technologies, OpenFlow and OVSDB. We showed that the network can be centrally programmed using solely IP and TCP addressing.

Thus, we have shown that this design can be a viable solution for data acquisition networks. Nevertheless, there are still some remaining questions. One relates to the achievable port density. There are physical limits on the number of network interfaces that can be installed in a single server and also the physical space available at an experiment's site is limited. Second, fault tolerance, including controller, switch, and link failures, also requires detailed evaluation. A further line of research could also include the study on the fairness across data collectors, how service disciplines might impact latencies, and a generalized algorithm for load balancing when link bandwidths across the farm are not identical.

References

[1] The ATLAS Collaboration, "The ATLAS Experiment at the CERN Large Hadron Collider," *Journal of Instrumentation*, vol. 3, no. 08, p. S08003, 2008.

[2] G. Jereczek *et al.*, "A lossless switch for data acquisition networks," in *Proc. IEEE LCN*, 2015, pp. 552–560.

[3] N. Neufeld, "LHC trigger & DAQ — an introductory overview," in *Proc. IEEE-NPSS Real Time Conference*. IEEE, 2012, pp. 1–4.

[4] G. Jereczek *et al.*, "Analogues between tuning TCP for data acquisition and datacenter networks," in *Proc. IEEE ICC*, 2015, pp. 6062–6067.

[5] S. Varma, *Internet Congestion Control*. Morgan Kaufmann, 2015.

[6] A. Phanishayee *et al.*, "Measurement and analysis of TCP throughput collapse in cluster-based storage systems." in *FAST*, vol. 8, 2008, pp. 1–14.

[7] A. Vishwanath *et al.*, "Perspectives on router buffer sizing: Recent results and open problems," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 2, pp. 34–39, 2009.

[8] T. Colombo *et al.*, "Data-flow performance optimisation on unreliable networks: the ATLAS data-acquisition case," *Journal of Physics: Conference Series*, vol. 608, no. 1, p. 012005, 2015.

[9] DPDK. [Online]. Available: http://dpdk.org/

[10] Open vSwitch. [Online]. Available: http://openvswitch.org/

[11] OpenFlow switch specifications. [Online]. Available: https://www.opennetworking.org/sdn-resources/technical-library#tech-spec

[12] B. Pfaff *et al.*, "The Open vSwitch Database Management Protocol," Internet Requests for Comments, RFC Editor, RFC 7047, December 2013. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7047.txt

[13] Open Networking Foundation. [Online]. Available: http://www.opennetworking.org/

[14] T. D. Nadeau *et al.*, *SDN: Software Defined Networks*. O'Reilly Media, Inc., 2013.

[15] N. McKeown *et al.*, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[16] R. Rojas-Cessa *et al.*, "Schemes for fast transmission of flows in data center networks," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 3, pp. 1391–1422, 2015.

[17] Y. Zhang *et al.*, "On architecture design, congestion notification, TCP incast and power consumption in data centers," *IEEE Communications Surveys Tutorials*, vol. 15, no. 1, pp. 39–64, 2013.

[18] Y. Ren *et al.*, "A survey on TCP incast in data center networks," *International Journal of Communication Systems*, 2012.

[19] J. G. Panduro Vazquez, "The ATLAS data acquisition system: from run 1 to run 2," *Nuclear Physics B - Proceedings Supplements*, 2015.

[20] G. Antichi *et al.*, "Time structure analysis of the LHCb DAQ network," *Journal of Physics: Conference Series*, vol. 513, no. 6, p. 062009, 2014.

[21] T. Bawej *et al.*, "Boosting event building performance using Infiniband FDR for the CMS upgrade," in *Proc. TIPP*, 2014, p. 190.

[22] The OpenDaylight platform. [Online]. Available: https://www.opendaylight.org/

[23] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[24] Python. [Online]. Available: https://www.python.org/

[25] B. Pfaff *et al.*, "The design and implementation of Open vSwitch," in *Proc. USENIX NSDI*, 2015.

[26] *Intel Server Board S2600GZ/GL Technical Product Specification*, Intel, 2012, revision 1.1.

[27] *Intel 82599 10 GbE Controller Datasheet*, Intel, February 2015.

[28] "IEEE standard for Ethernet," *IEEE Std 802.3*, 2012.

[29] J. D. McCalpin, "Sustainable memory bandwidth in current high performance computers," *Silicon Graphics Inc*, 1995.

[30] M. Alizadeh *et al.*, "On the data path performance of leaf-spine datacenter fabrics," in *Proc. IEEE HOTI*. IEEE, 2013, pp. 71–74.

[31] F. Carena *et al.*, "Preparing the ALICE DAQ upgrade," vol. 396, no. 1, p. 012050, 2012.

[32] A. Andreyev. Introducing data center fabric, the next-generation Facebook data center network. [Online]. Available: https://code.facebook.com/posts/360346274145943/introducing-data-center-fabric-the-next-generation-facebook-data-center-network/

[33] G. Berger. Facebook fabric networking deconstructed. [Online]. Available: http://firstclassfunc.com/facebook-fabric-networking

[34] "IEEE standard for bridging & management," *IEEE Std 802.1Q*, 2014.

[35] D. P. Palomar *et al.*, "Practical algorithms for a family of waterfilling solutions," *IEEE Transactions on Signal Processing*, vol. 53, no. 2, pp. 686–695, 2005.