Analogues between tuning TCP for Data Acquisition and Datacenter Networks

Grzegorz Jereczek*†, Giovanna Lehmann Miotto*, David Malone†
*European Organization for Nuclear Research (CERN), Geneva, Switzerland
†Maynooth University, Maynooth, Irleand

***Abstract*—A many-to-one communication pattern is present both in Data Acquisition (DAQ) and datacenter networks. 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 paper provides two contributions. First, we confirm that there are strong analogies between the TCP behavior in DAQ and datacenter networks. Second, we evaluate different proposals from the datacenter environment for application in DAQ to improve performance and reduce buffer requirements.**

I. Introduction

ATLAS [1] is a general-purpose particle detector designed to study particle collisions at the Large Hadron Collider (LHC) at CERN. ATLAS, like other existing and emerging large-scale experiments, produces increasingly high volumes of data. Their source can be a wide variety of instruments, including sensors, detectors, antennas or telescopes. Although characteristics and goals of these experiments vary, they share a common challenge for real-time filtering, storage and analysis of the acquired data [2]. One of the key components of this chain is a network, called the data acquisition network. It collects together the outputs from all the instruments to reconstruct a physical process. The ATLAS DAQ network is used throughout this paper as a case study (see Fig. 1a).

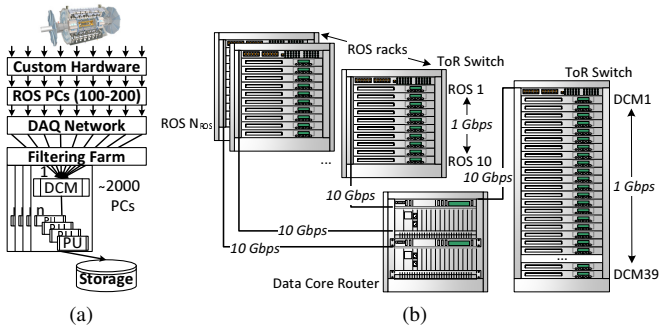
There is a desire to use commodity hardware in these systems whenever possible. It simplifies the configuration and maintenance, but also reduces the costs. The ATLAS DAQ network is based on TCP/IP and Ethernet technologies [3]. This poses, however, several challenges due to the bursty nature and many-to-one communication pattern of the data flow in DAQ. It resembles one of the patterns identified to be very common in typical datacenter networks: incast [4].

This paper discusses the analogues between tuning TCP for DAQ and datacenters to avoid incast. We evaluate if solutions proven to be good for datacenter could be also employed in DAQ. In the first part of the article we describe a typical DAQ network and reveal its relation to the TCP incast in datacenters. This part ends with related work in this area. In the second half of the article we validate proposals for solving datacenter-incast in the real DAQ system of the ATLAS experiment.

II. Background and motivation

A. Data Acquisition Networks

DAQ networks collect data from their sources and transport them to processing units for further analysis. The high level



(a) (b)

Fig. 1. The ATLAS DAQ system (a) and its network topology (b) as of summer 2014.

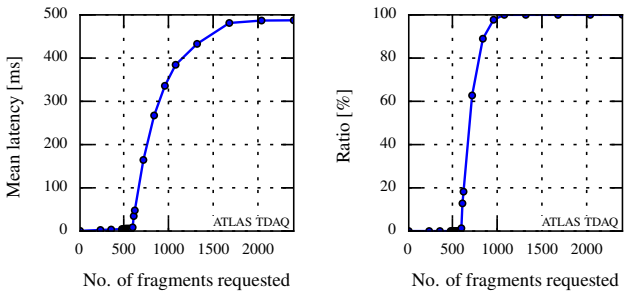
diagram of the ATLAS DAQ is presented in Fig. 1a.

Events in the LHC nomenclature are particle collisions inside of the LHC and their physical "fingerprints" are recorded by the detector, which is the data source for the DAQ system. The Read-Out Subsystem (ROS) is an interface between the ATLAS-specific data acquisition components and commodity equipment. This is the entry point to the DAQ network transporting event data to the filtering farm (HLT, High Level Trigger). The ROS sends data of a particular event to a single Processing Unit (PU), which is a worker process on a commodity server performing event reconstruction and analysis. There are multiple PUs on each server of the 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 ATLAS DAQ/HLT system is based on 10 Gbps Ethernet network with TCP as transport layer protocol and is capable of filtering 2 Tbps of event data, which is required by the experiment. The requirement for the average data bandwidth can be satisfied with commodity networking hardware nowadays, but the specifics of data acquisition place an additional burden on the network subsystem. The burstiness of the traffic makes it difficult to handle with shallow-buffered network switches.

B. Relation to TCP incast in datacenters

Depending on the exact configuration of the ROS, between 100-200 PCs are used to read out the event data from the detector. If a DCM requested full event data from all the ROS PCs simultaneously, it would cause a large traffic burst in the network. Switches without sufficient buffer capacity would drop some portion of the packets, leading to an increased event data collection time, which is one of the important parameters



(a) Data collection latency (b) Events with TCP timeouts

Fig. 2. Collapse of event data collection due to abrupt increase of the latency (a), if too many event fragments are requested at the same time. It coincidences with the fraction of events experiencing collection latency above 200 ms (b), which is an indication of at least one TCP timeout. Fragments of size 1 kB are spread across 200 ROS PCs.

that need to be kept under control in a DAQ system. This kind of behavior is presented in Fig. 2a (see caption for details).

Synchronous requests for event data fragments from a single DCM to a large number of ROS servers closely resembles the many-to-one communication patterns in datacenters, which are affected by the TCP incast pathology [5]. TCP incast is defined as "a catastrophic throughput collapse that occurs as the number of storage servers sending data to a client increases past the ability of an Ethernet switch to buffer packets" [6]. Incast is usually analyzed in terms of throughput, which is significantly degraded by the TCP recovery mechanisms after packet loss. TCP timeouts, in particular, introduce delays of hundreds of milliseconds leading to throughput degradation [6]. The typical value for the minimum TCP retransmission timeout (RTO) in TCP stack implementations of common Linux distributions is 200 ms [7] (including Scientific Linux 6, SLC6 [8], used at CERN). These delays are the direct cause of the increased data collection latency as can be seen in Fig. 2b. A single TCP timeout increases the collection time by at least 200 ms, compared to three orders of magnitude lower 200 µs round trip time (RTT) of the DAQ network. It confirms a close relation between TCP incast in datacenters and traffic burstiness in DAQ. Therefore, solutions from datacenters could be tested in DAQ systems and, if successful, would help in increasing their level of commoditization. Based on this premise we explore possible adoption of techniques proposed for datacenters to solve or reduce the impact of TCP incast in DAQ systems.

C. Related work

A number of proposals for incast mitigation in datacenter, ranging from the link layer through the transport layer up to the application layer, can be found in the literature. For a review of recent work in this area, see [4], [9], [10] and references therein. Analytic models of incast behavior can be found in [5], [11], [12], but first indications of poor TCP performance with many flows were already presented in [13].

Details on the architecture of the ATLAS data acquisition system are discussed in [14], [15]. Characteristics of the DAQ traffic patterns on the example of the LHCb experiment at CERN can be found in [16]. The CMS experiment approaches the same problem with an InfiniBand network [17].

III. Current system

The following section begins with a description of the network configuration of the ATLAS DAQ, used throughout this paper to analyze and evaluate TCP performance. Then we describe the current technique to mitigate the incast problem, which is an application layer solution: traffic shaping.

A. Network configuration

The LHC and all its experiments are currently undergoing a major upgrade for its second run period starting in 2015. Since no data collection is ongoing, it is possible, to some extent, to test and evaluate modified TCP variants on a real large-scale DAQ system.

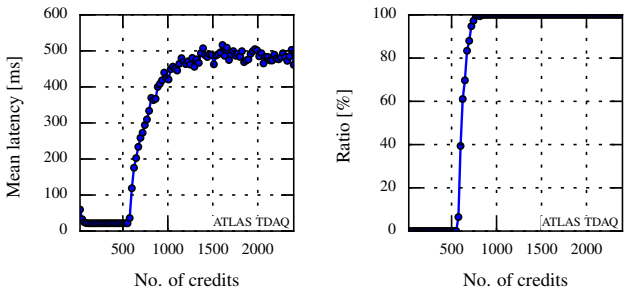
The diagram of the ATLAS data acquisition network configured for the purposes of this paper is presented in Fig. 1b. The ROS subsystem is used in emulation mode, generating dummy data. On the DCM side, no actual event processing is done either. The DCMs request fake events from the ROS and push it to fake PUs with preconfigured processing times. In this way, the network subsystem can be analyzed in isolation from other factors while the topology corresponds closely to the real DAQ configuration.

Fake event fragments are spread across ROS PCs, 12 fragments each. All fragments make an entire event (depending on the number of enabled ROS PCs). These PCs are physically located in different racks (9-12 ROS PCs each) so as not to limit their bandwidth by the top of rack (ToR) to data core (DC) link. For the filtering farm, only a single rack with 39 DCM machines is used in this configuration. HP6600 switches [18] are used to aggregate machines within the same rack. The core of the network is made with the Brocade MLXe32 router [19] using the Virtual Output Queuing (VOQ) architecture [20]. ToR switches and the DC router are connected via 10 Gbps uplinks. All connections within the rack are 1 Gbps.

As already mentioned, the average RTT of the network is 200 µs, so its bandwidth-delay product (BDP) is:

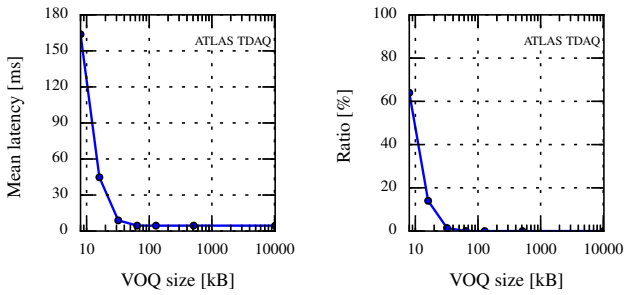
$$BDP = Bandwidth \cdot RTT = 25\,\text{kB} \approx 17\,\text{Packets} \qquad (1)$$

with a single packet's Maximum Transmission Unit (MTU) of 1500 B. This means that 100 ROS applications sending a single TCP segment to one DCM would exceed the network's BDP by a factor of 6. This excess must be buffered in the switches, otherwise packet drops and increased latency are observed. The initial congestion window of SLC6 [8] is 10 TCP segments, so upon receiving a request from DCM all ROS endpoints are allowed to inject all event data into the network. This explains the abrupt change of the data collection latency in Fig. 2a and 2b above 500 fragment requests. The excess does not fit into the DCM ToR switch buffers, so packets are dropped leading to TCP retransmissions and timeouts. From this it can be estimated that the HP6600 [18] ToR switch has about 500 kB buffer per 1 Gbps output port.



(a) Data collection latency (b) Events with timeouts

Fig. 3. TCP incast mitigation with traffic shaping when requesting full event data (2400 fragments of 1 kB across 200 ROS PCs) from a single DCM.



(a) Data collection latency (b) Events with timeouts

Fig. 4. Solving TCP incast with increased buffer sizes. Performance of a single DCM (2400 event fragments of 1 kB across 200 ROS PCs).

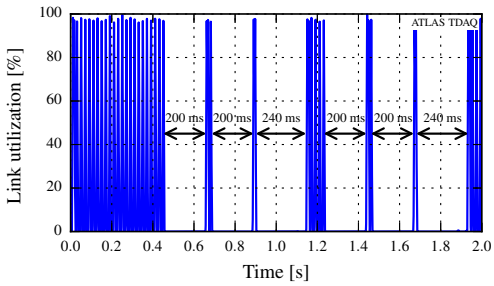


Fig. 5. Link utilization of a DCM over longer period of time. It remains idle for at least 200 ms after a TCP timeout, which suspends data collection.

B. Traffic shaping

Traffic shaping is a simple credit-based system allowing the reduction of the impact of the DAQ traffic burstiness [15]. It is employed on the aggregator side, the DCMs. Each of them is assigned a fixed amount of credits. One credit permits a request for one event fragment (independent of its size, a fixed fragment size among the entire ROS is used in this paper). A DCM shall 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.

The capability of traffic shaping to mitigate TCP incast is demonstrated in Fig. 3. 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 packetization delay of the entire event size on a 1 Gbps link of 19.2 ms, thus proving the efficiency of the algorithm. Below 100 credits the network is not fully utilized (insufficient traffic in the network). On the other hand, the switch buffers are overstressed and start to drop packets above 550 credits quota (Fig. 3b, compare to 2a).

IV. Approaches to avoid TCP-incast

In this section we introduce some approaches to improve the TCP performance when facing incast, which we evaluate in reference to possible adoption in data acquisition systems. We start with a straightforward solution, which is increasing the buffer sizes in the network. Then we will confirm the need to revise the parameters for TCP retransmission mechanisms. A simplistic approach of statically configuring the TCP congestion parameters is analyzed afterward. We have also chosen one of the proposals for modified TCP congestion control, Deadline and Incast Aware TCP (DIATCP [21]), to evaluate as an alternative to traffic shaping.

A. Increasing the buffer sizes

The root cause of incast is packet drops due to buffer overflows in the network switches. A simple solution is to increase the buffer space [6]. Although the most effective, it is also the least scalable approach as networking hardware with larger memory cost more. The Brocade MLXe32 router supports VOQ with queues of size up to 256 MB [19]. Together with the traffic shaping algorithm it can handle the entire DAQ farm consisting of ~2000 servers filtering the LHC data.

Fig. 4 shows the effectiveness of increasing buffer sizes in case when only one DCM is active in the system. It was simulated by changing the maximum allowed VOQ size at the DC router. With more than 64 kB VOQ the packet losses are eliminated in the given configuration. Because of the high costs of switch memory this way of solving incast can only be used if memory cost is reduced.

B. Fine-grained TCP retransmissions

With our results we confirm the desirability for the revision of the TCP retransmission timeouts as already indicated in [5] and [7]. Both publications show that microsecond, instead of millisecond, timeouts significantly reduce the consequences of incast. We do not consider it as solution to incast for DAQ systems, where retransmissions because of congestion can be eliminated in first place, but the implications of non-congestion related drops should be diminished as well. What can be seen in Fig. 5 is the effect of TCP retransmission timeouts during event data collection, which leads to locking of the PUs while waiting for a TCP retransmission after a timeout of at least 200 ms and increasing CPU idle time.

C. Static TCP configuration

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

$$W = \min(cwnd, awnd) \tag{2}$$

The optimal window is about the size of the BDP of the network. It ensures that enough bytes are in-flight to keep its links busy [3]. On the other hand, too large a value of W leads to queuing in the switches and eventually packet drops. The network is saturated without causing losses when:

$$BDP \leq W < BDP + B \tag{3}$$

where B is the available buffer at the switch [22]. In case of multiple flows in the network:

$$BDP \leq \sum_{i=1}^{N} \min(cwnd_i, awnd_i) < BDP + B \tag{4}$$

where $\min(cwnd_i, awnd_i)$ is the send window of the i-th flow. The total number of flows (N) is given by the number of ROS PCs (N_{ROS}) and DCMs (N_{DCM}):

$$N = N_{ROS} \cdot N_{DCM} \tag{5}$$

A simple static configuration of the send window in ROS $cwnd_i = cwnd$ can be a potential solution to TCP incast. The $cwnd$ can be easily configured with custom Linux loadable congestion control modules, so kernel recompilation is not even required. Assuming a large receiver's window $awnd$, the condition (4) for $cwnd$ of a single flow becomes:

$$\frac{BDP}{N} \leq cwnd < \frac{BDP + B}{N} \tag{6}$$

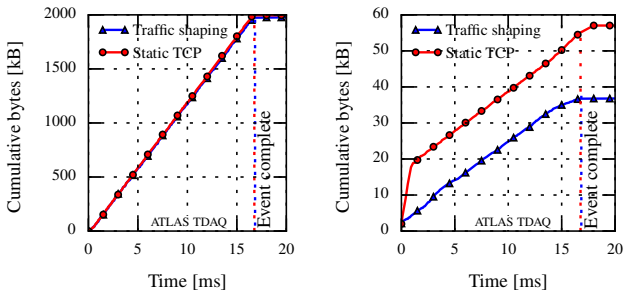
We pointed out in Section III-A that with $N = 147$ we already exceed the BDP, but with Equation (6) we can approximate the required amount of buffering. With fixed $cwnd$ we guarantee that there will be no more than:

$$N \cdot cwnd = N_{ROS} \cdot N_{DCM} \cdot cwnd \tag{7}$$

of packets or bytes in-flight in the DAQ network. We evaluated this approach with $cwnd$ of 2 packets. In configuration with a single DCM and MTU of 1500 B we will need no more than 416 kB of buffering, which does not exceed the estimated per port memory at the ToR switch. We can also calculate an upper bound for the 39 DCMs setup, which is about 17 MB and gives an estimate of the required VOQ size for a complete rack, with all its DCM applications.

2) Single event performance: Fig. 6 shows the IO behavior of traffic shaping and static TCP configuration when collecting a single event. This data was generated from TCP 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 Fig. 6b), whereas with traffic



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

Fig. 6. IO graphs of a single event. The last ROS response (a) arrives after 16.766 ms for static $cwnd$ of 2 packets and after 16.84 ms for traffic shaping (396 credits quota). 147 ROS PCs with 1.1 kB fragments.

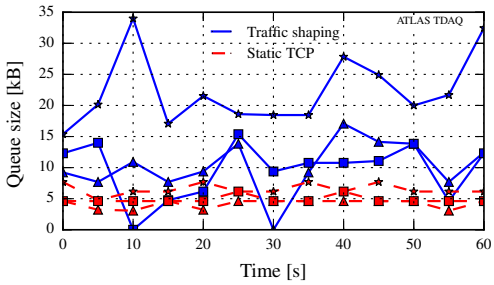


Fig. 7. Maximum VOQ size of three different blades during event data collection. One blade experiences higher load (more connected ROS racks).

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 (Fig. 1b) the ROS racks are connected to different blades of the DC router (Fig. 1b) with independent VOQs. The instantaneous size of a single VOQ is stable over the time with lower maximum in case of static configuration (Fig. 7). We can explain that by the fact that particular ROS PC responds with a burst of event data for a single request, if traffic shaping is used. Static TCP 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 (Fig. 6b) is 20 kB higher for the static configuration due to the increased number of TCP acknowledgments (TCP ACKs). With $cwnd$ of 2 packets an ACK is sent for every 2 packets coming from ROS. With the traffic shaping approach their rate is lower, especially if TCP segmentation offloads are enabled on the DCM. 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. It is true only for the static TCP. In section IV-C3 we will see that this will be the cause for slightly increased mean latency of data collection seen by the application layer.

3) Incast avoidance with 1 DCM: The cumulative distribution function (CDF) of event collection latency is presented in

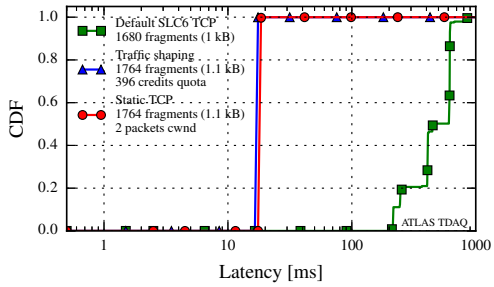


Fig. 8. Data collection latency with 1 DCM. Static TCP and traffic shaping eliminate TCP timeouts. All events suffer from at least one TCP timeout with the default SLC6 TCP (TCP-Cubic without incast-avoidance algorithms).

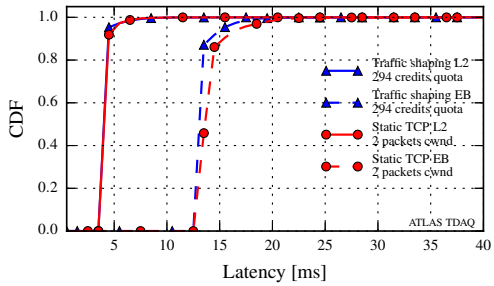


Fig. 9. Data collection latency with 39 DCMs. It is divided into L2 and EB stages to fit into the available bandwidth. No TCP timeouts can be observed for both solutions. 147 ROS PCs provide data fragments of 1.1 kB size.

Fig. 8. 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 (Fig. 6) as seen from a TCP dump. The explanation lies in the fact that the mean latency in Fig. 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 Fig. 6b for the static TCP after having received the last data response from ROS. The difference in mean latency is minor and does not significantly affect the entire system's performance.

4) Incast avoidance with 39 DCMs: Incast is avoided also in a setup with 39 DCMs (Fig. 9). Since the bandwidth of the DC-ToR 10 Gbps link is not enough to handle the required event rate (2 kHz), filtering is split into two phases [14]:

1) Partial event reconstruction ("L2");
2) Full event building ("EB").

In the first one a subset of event fragments is requested from ROS and prompt decision is made by a PU whether these data look promising. DCM pulls the remaining fragments only for events accepted for the EB phase, so the bandwidth is reduced. It is a simplification of the real ATLAS event filter, but sufficient to study the network behavior.

The CDF (Fig. 9) is analogous to the one presented in the previous section. There are no TCP timeouts both for the static and traffic shaping algorithms and the second one shows slightly lower mean latency of the EB stage.

D. DIATCP

Deadline and Incast Aware TCP (DIATCP) is a TCP variant proposed for datacenter [21]. 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.

1) Algorithm: DIATCP is deployed only at the aggregator (the DCM in our case). In DIATCP the peers' (ROS) sending rate is controlled to avoid incast and application deadlines. For a description refer to [21]. 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 analyze 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 the network, DIATCP utilizes the advertisement field in the TCP ACKs to allocate a specific window size to each ROS peer. It is the $awnd$ (see Section IV-C1) that is controlled by DIATCP and the condition defined with (4) takes the form:

$$BDP \leq \sum_{i=1}^{N} awnd_i < BDP + B \tag{8}$$

where $awnd_i$ is the advertised window to the ROS on i-th DCM-ROS 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 a single DCM is the global window as defined in [21]:

$$gwnd = \sum_{i=1}^{N_{ROS}} awnd_i \tag{9}$$

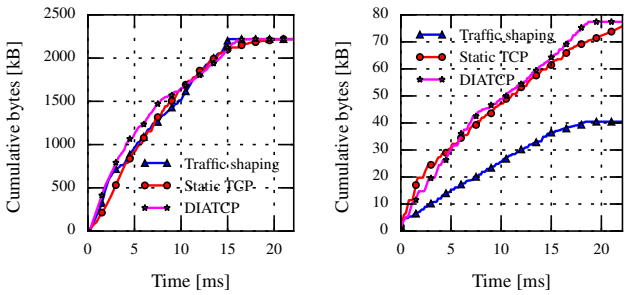
With the same $gwnd$ at each DCM, the condition (8) becomes:

$$BDP \leq N_{DCM} \cdot gwnd < BDP + B \tag{10}$$

[21] gives guidance on tuning $gwnd$ depending on the network's RTT. We limit ourselves for $gwnd$ fulfilling the requirement (8) 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 in event fragment units. We can therefore treat DIATCP as traffic shaping implemented at the transport layer.

2) Testbed: DIATCP cannot be implemented in 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. Since the ATLAS DAQ system is about to start with the next run of LHC next year, we decided to evaluate DIATCP with an experimental testbed running the same ATLAS DAQ software. It consists of 160 ROS applications emulated on 4 server class PCs equipped with double-port Intel 82599 10 Gbps Ethernet controllers and 1 DCM with 1 Gbps controller. ROS and DCM are connected



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

Fig. 10. IO graphs of a single event. The last ROS response arrives at DCM after 18.94 ms in case of DIATCP with $gwnd$ of 160 packets, 22.19 ms for static $cwnd$ of 2 packets and 18.48 ms for traffic shaping (421 credits quota). Experimental testbed used in all cases.

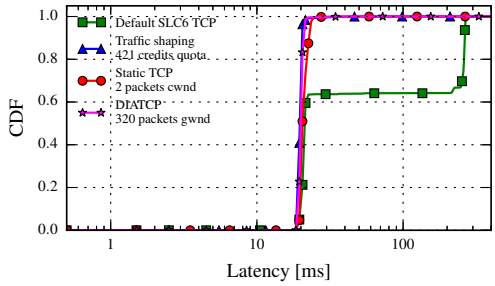


Fig. 11. Data collection latency (1 DCM). The three variants perform without TCP timeouts opposed to the default TCP. Experimental testbed in all cases. The software router drops less packets than the ToR switch of the ATLAS DAQ resulting in less timeouts and better performance of SLC6 than in Fig. 8.

via a server with the same Intel adapters. It is configured as a software router using the Linux IP forwarding capabilities. With help of different Linux queuing parameters, it is possible to emulate the behavior of a real DAQ network. Each ROS application is configured with 12 fragments of 1.1 kB.

3) Single event performance: The IO graphs in Fig. 10 indicate comparable latencies for DIATCP and traffic shaping. Static TCP performs measurably worse with the last ROS response arriving 3 ms later. We believe this is because the use of a server instead of a real switch. The traffic is spread among different Ethernet cards and CPU sockets on the server-based router. The BDP of this network (0.5 ms RTT) is already exceeded with a 2 packets $cwnd$, but the complexity of this configuration has more implications on the results when serving all TCP flows in parallel compared to a standard router. Higher TCP traffic in the direction to ROS with DIATCP (Fig. 10b) is caused by frequent TCP window updates.

4) Incast avoidance with 1 DCM: Fig. 11 confirms that DIATCP can be a valid candidate for DAQ. TCP timeouts are gone from the system in case of all the three solutions. The mean latencies of DIATCP and traffic shaping are comparable.

V. Conclusion

In this paper we showed that there are analogues between the use of TCP in DAQ networks and the incast problem in datacenters. We evaluated a number of mechanisms to alleviate these problems, allowing improved performance and reduced buffer requirements. Even a simple static configuration of TCP congestion control, which does not require any additional programming overhead, can already effectively solve the problem. Advanced TCP incast avoidance algorithms proposed for datacenter can be 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.

Acknowledgment

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).

The authors thank Dr. Jaehyun Hwang for sharing and help in integrating the DIATCP implementation.

References

[1] ATLAS TDAQ Collaboration, "The ATLAS Trigger/DAQ Authorlist," CERN, Geneva, Tech. Rep. ATL-DAQ-PUB-2013-001, Jun 2013.
[2] A. Di Meglio *et al.*, "CERN openlab whitepaper on future IT challenges in scientific research," 2014.
[3] K. R. Fall *et al.*, *TCP/IP illustrated, volume 1: The protocols*. Addison-Wesley, 2011.
[4] Y. Ren *et al.*, "A survey on TCP incast in data center networks," *International Journal of Communication Systems*, 2012.
[5] Y. Chen *et al.*, "Understanding TCP incast and its implications for big data workloads," DTIC Document, Tech. Rep., 2012.
[6] A. Phanishayee *et al.*, "Measurement and analysis of TCP throughput collapse in cluster-based storage systems." in *FAST*, vol. 8, 2008.
[7] V. Vasudevan *et al.*, "Safe and effective fine-grained TCP retransmissions for datacenter communication," in *Proc. ACM SIGCOMM*, 2009.
[8] Scientific Linux. [Online]. Available: https://www.scientificlinux.org/
[9] I. R. Shohab *et al.*, "Transport layers approaches for TCP incast problem at data center networks," *IJSER*, vol. 5, 2014.
[10] R. P. Tahiliani *et al.*, "TCP variants for data center networks: A comparative study," in *Proc. IEEE ISCOS*, 2012.
[11] Z. Feng *et al.*, "An analytic goodput model for TCP incast," in *Proc. IEEE IMIS*, 2012.
[12] M. Alizadeh *et al.*, "Data center TCP (DCTCP)," in *Proc. ACM SIGCOMM*, 2010.
[13] R. Morris, "TCP behavior with many flows," in *Proc. IEEE ICNP*, 1997.
[14] J. G. Panduro Vazquez, "The ATLAS data acquisition system: from run 1 to run 2," ATL-DAQ-PROC-2014-035, Tech. Rep., 2014.
[15] T. Colombo, "Data-flow performance optimisation on unreliable networks: the ATLAS data-acquisition case," ATL-DAQ-PROC-2014-029, Tech. Rep., 2014.
[16] G. Antichi *et al.*, "Time structure analysis of the LHCb DAQ network," in *Computing in High Energy and Nuclear Physics (CHEP)*, 2013.
[17] T. A. Bawej *et al.*, "Boosting event building performance using Infiniband FDR for CMS upgrade," Tech. Rep., 2014.
[18] HP 6600 switch series. [Online]. Available: http://www.hp.com/
[19] Brocade MLX series. [Online]. Available: http://www.brocade.com/
[20] A. Kumar *et al.*, *Communication networking*. Elsevier, 2004.
[21] J. Hwang *et al.*, "Deadline and incast aware TCP for cloud data center networks," *Computer Networks*, 2014.
[22] H. Zheng *et al.*, "An effective approach to preventing TCP incast throughput collapse for data center networks," in *Proc. IEEE GLOBECOM*, 2011.