Unbiased QCN for Scalable Server-Fabrics

Nikolaos Chrysos*, Fredy Neeser*, Rolf Clauberg*, Daniel Crisan*,
Kenneth Valk†, Claude Basso†, Cyriel Minkenberg*, and Mitch Gusat*
*IBM Research - Zurich, Switzerland, †IBM Systems & Technology Group, Rochester, USA

***Abstract*—Ethernet is the predominant Layer-2 networking technology in the datacenter, and evolving into an economical alternative for high-performance computing clusters. Ethernet traditionally drops packets in the event of congestion, but IEEE introduced lossless class services to enable the convergence of storage and IP networks. Losslessness is a simple, well-known concept, but its application in datacenters is hampered by the fear of ensuing saturation trees. In this work, we aim to accelerate the deployment of Quantized Congestion Notification (QCN). In particular, we first eliminate the intrinsic unfairness of QCN under typical fan-in scenarios by installing the congestion points at inputs, instead of at outputs as standard QCN does. We then demonstrate that QCN at input buffers cannot always discriminate between culprit and victim flows. To overcome this limitation, we propose a novel QCN-compatible marking scheme, namely occupancy sampling. We have implemented these methods in a server-rack fabric with 640, 100G Ethernet ports.**

***Index Terms*—Converged Datacenter Networks, Switching Fabrics, Congestion Control, 100G Ethernet.**

I. Introduction

Ethernet, which is the prevailing networking technology in datacenters, faces a challenge. If an Ethernet network prevents packet drops (i.e., if it is lossless), it can easily choke under bursty workloads, thus producing less and frequently delayed useful work. However, if it drops packets (lossy), it has to disengage from encompassing Fibre Channel over Ethernet (FCoE) and RDMA over Converged Ethernet (RoCE), which port storage, cluster and IP traffic in converged, *lossless* LAN networks.

For computer architects and the general parallel computing community, lossless networks are not that new. Examples are PCI, Infiniband, and many proprietary, on-chip and off-chip computer interconnects. However, most of today's datacenter networks are lossy. Lossy datacenter networks are known to have several serious performance issues with distributed applications. In TCP incast, for example, storage flows that fan-in into a server experience a throughput collapse in synchrony due to repetitive packet drops. Numerous proposals have been made to mitigate this effect, but the most effective and intuitive one is to enable link-level flow control in the network [1].

The benefits of lossless networks accrue when considering scale-out workloads, such as real-time, delay-sensitive applications that are typical for commercial datacenters or BigData analytics. As suggested in [2], a service that distributes work to 100 nodes may experience an unacceptable delay on 63% of the times it is deployed even if only 1% of the (network) flows is delayed; recovering from packet drops in software (e.g., using TCP retransmission timers) can increase flow completion times (FCTs) by more than an order of magnitude. Lossless networks were shown to reduce query completion times for Partition/Aggregate workloads [3]. Furthermore, lossless operation is equally important for virtual, software-based networks; enabling flow control in both the physical network and the virtual switch was shown to reduce FCTs by up to 7x for Partition/Aggregate queries [4].

A. Lossless Ethernet

To enable lossless services, IEEE first introduced (802.3x) PAUSE, a link-level flow control similar to *Stop&Go*. Recently, IEEE also standardized *Priority Flow Control (PFC)* for *Converged Enhanced Ethernet (CEE)* networks. The different priority levels are assigned private buffers in front of links. Within each priority, PFC acts as 802.3x PAUSE, but a PAUSEd priority does not affect the others, which is similar to virtual channel flow control in multiprocessor networks. Nevertheless, the industry is still reluctant to enable link-level flow control, mainly because it can induce *saturation trees*. These are formed when a number of congested flows fill up the link buffers in front of a link. Because of the flow control, the backlogs from such congested flows can backpropagate, thus forming

a saturation tree. The bad news is that such congestion spreading can block any packet, regardless of whether it belongs to a congestive flow (culprit) or to an innocent flow (victim).

To counteract saturation trees, IEEE (802.1Qau) has standardized a congestion control scheme for Ethernet networks, called *Quantized Congestion Notification (QCN)* [5]. QCN installs *Congestion Points (CPs)* at switch output queues. Each CP samples the arriving frames (i.e., Ethernet packets) according to a sampling interval, and characterizes the queue congestion by two state variables: position (offset), defined with respect to an equilibrium setpoint Q_{eq} as $Q_{\text{off}}(t) \triangleq Q(t) - Q_{\text{eq}}$, and velocity, $Q_\delta(t) \triangleq Q(t) - Q_{\text{old}}$. When the CP detects congestion, in the sense that the feedback value $F_b \triangleq Q_{\text{off}} + w \cdot Q_\delta$ is positive, it sends a *Congestion Notification Message (CNM)* to the source of the most recent frame (or flow), which is considered the culprit. *Converged Network Adapters (CNAs)* at the sources react to CNMs by instantiating set-aside queues and QCN *Rate Limiters (RLs)* for the congested flows. In response to CNMs, a RL multiplicatively decreases its injection rate as a function of the feedback value; in the absence of CNMs, it autonomously increases the injection rate.

QCN is an elegant congestion management scheme, but its deployment in real-world switching environments is challenging. In this paper, we address a number of QCN implementation issues, including fairness.

B. Contributions

Our main contributions are the following.

1) Using QCN, we achieve fair throughputs in fan-in scenarios by re-placing the CPs from the switch outputs to the switch inputs.
2) We propose a new flow marking scheme for QCN that is able to identify the culprit flows for non-FIFO frame departures,
3) We report the implementation of our mechanisms in a server-rack fat-tree fabric with 640 100G Ethernet ports.

II. QCN For Server-Rack Fabrics

Packet switches may comprise a single-stage crossbar or shared-memory chip, or a multi-stage interconnection network of multiple smaller switching elements. Standard QCN is based on an idealized and generic output-queued switch and allocates one CP at each output buffer [6][5, Sec. 30.2.1]. However, to avoid the excessive memory throughput of pure output queuing, scalable switching fabrics, in addition to output buffers, must employ input buffers as well. Our results apply to such *combined-input-output-queued (CIOQ)* switches and switching fabrics [7].

In CIOQ switches, the incoming data frames are stored in *Virtual Output Queues (VOQs)* in front of the internal interconnect, and a scheduler is responsible for transferring them to their targeted output queue. Although such VOQ-based architectures avoid head-of-line (HOL) blocking, they cannot prevent buffer hogging, where a congested flow monopolizes an input buffer. Ethernet congestion control, QCN, is designed to throttle the congested VOQ (flow) and protect the innocent ones. In this paper, we describe an advantageous QCN architecture with *CPs at switch inputs*, which we have implemented in a high-performance server-rack fabric.

Traditionally, datacenters used tree-like networks. These topologies have less capacity at layers closer to the root and cannot accommodate the aggregate bandwidth of all servers. To cope with the increasing volume of inter-server traffic, modern datacenters turn to *flattened* networks, based on the *fat-tree* topology, which can offer full-bisection bandwidth.

Our fabric, shown in Fig. 1, uses a two-level fat-tree *(spine-leaf)* topology—every leaf switch is connected to every spine switch (Fig. 1(b)). The leaf switches are integrated into the backplane of the racks, and each one constitutes the network edge for five (5) servers, providing a 100 Gb/s (bidirectional) link to every server. The fabric supports four racks, with 32 edge switches each, thus providing a total of 640 100G ports. The leaf switches have dedicated input and output buffers per port and per priority level. At its ingress interface, a leaf switch segments the incoming frames into variable-size fabric-internal packets (or cells), and stores the latter in 256B buffer units that are linked together to form VOQs. Each packet can use any of the available leaf-to-spine links, as enforced by a packet-level spraying mechanism that overcomes the limitations of flow-level hashing [8]. The original Ethernet frames are reordered and reassembled at their egress leaf switch (output buffers), and forwarded to their destination server.

The spines themselves are cell-based, CIOQ switching elements, agnostic of the higher-level protocols. They reside in separate chassis, and each one provides 136, 25 Gb/s ports, enabling high-density (indirect) connec-

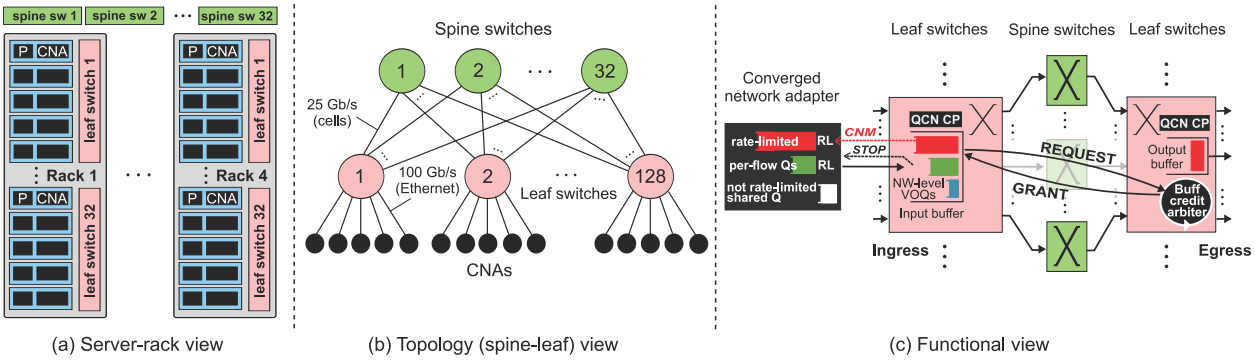


Figure 1. (a) A distributed server-rack fabric, comprising four racks and 640 servers, which are connected to a fat-tree network using 100G Ethernet links. (c) One ingress leaf switch is shown with its input buffer hosting the VOQs. An adapter (CNA) connecting to one port of the ingress leaf switch is also shown. When the input buffer occupancy reaches Q_{high}, the input port sends a STOP to the CNA, which does not forward new data until it receives a GO.

tions among the leaf switches. The spines feature small input and output buffers (16 cells per port), which are flow controlled using hop-by-hop backpressure. We use 32 spines in our four-rack system. With 32, 25 Gb/s links per leaf switch connecting to the spines, the fabric features an over-provisioning ratio of 8:5; this speedup accommodates any internal overhead, leaving some headroom to compensate for scheduling inefficiencies.

An edge-to-edge request-grant credit scheme is used to schedule the VOQ injections towards the egress buffers[9]. Before injecting a frame into the fabric, a VOQ must issue a request to the target output credit arbiter. The latter grants credits to the requesting VOQs using a round-robin-like discipline. Request and grants are 10B messages, routed through the spine switches.

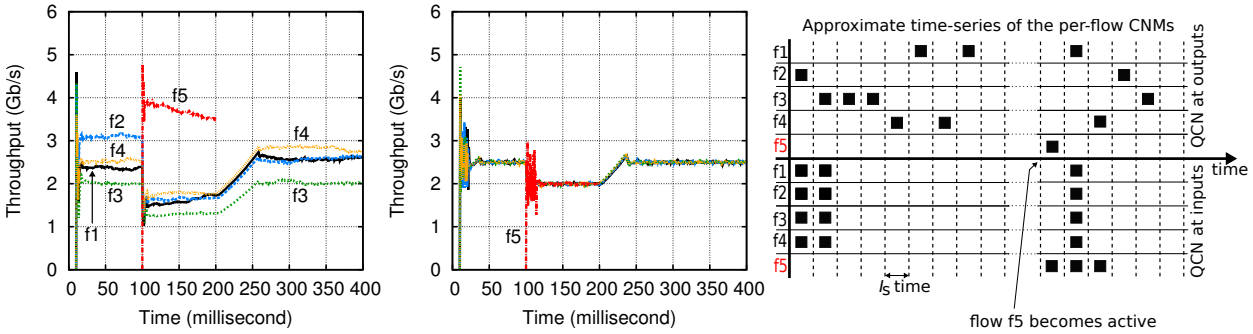
VOQs sharing input buffers: The buffer sizes in our fabric are approximately 150 KB per (input/output) port, and per Ethernet priority level. At each input, the VOQs of the same priority level share their buffer resources [10]. Effectively, as shown in Fig. 1(c), a VOQ that makes slow progress can fill up its input buffer, triggering a STOP (PAUSE) message to the upstream CNA. The PAUSE message will prevent buffer overflow by stopping all flows in the same priority level, irrespective of their destination. To counteract this needless blocking, we installed CPs at switch outputs, as recommended in [5]. Deviating from [5], we also installed CPs at switch inputs. The next section shows that it is preferable to activate the input CPs instead of the output ones.

III. Resolving QCN unfairness

Standard QCN employs CP at switch outputs. The trend towards switching fabrics with input buffers [7] suggests the possibility of installing QCN CPs at switch inputs instead of at switch outputs. QCN at an input buffer should detect overload, mark and throttle the culprit flow(s), and, using an appropriate Q_{eq}, keep the input buffer backlog below the Q_{high} threshold, thus avoiding the exertion of PAUSE. Note that the CNMs generated at inputs do not have to traverse the switch (from one port to another). Hence they neither consume fabric-internal bandwidth nor incur any additional delays. But, most importantly, as we demonstrate below, this alternative improves the fairness properties of standard QCN.

We evaluate various QCN alternatives on the fabric described in Sec. II, using a detailed C++ computer model. The Q_{high} (PAUSE-STOP threshold) is 110 KB, and the Q_{low} (PAUSE-GO threshold) 44 KB. QCN equilibrium point Q_{eq} = 60 KB. Finally, the QCN base sampling interval I_s = 150 KB, and the RL reaction time is 2.4 μs. In our experiments, we use 1522B frames for data traffic and 64B frames for PAUSE and QCN messages. We measure the raw throughputs of flows, which include the 20B per-frame overhead for the inter-frame gap, preamble and start frame delimiter.

To simplify the comparisons with recent literature and the IEEE 802 archives, we consider servers with 10G interfaces and standard QCN parameter settings [5]. We have also experimented with 100G links, and the results are qualitatively the same [8]. In our first experiment, we simulate a fan-in scenario on top of the



(a) QCN at switch outputs (b) QCN at switch inputs (c) Timing of CNMs

Figure 2. (a,b) Per-flow throughputs under a fan-in traffic scenario, where flows f1-f5 target the same destination. Flow f5 is active between 100 and 200ms. All configurations use PAUSE to sustain lossless operation. Experiments on the network of Fig. 1 with 10G servers. (c) The time-series of the per-flow CNMs. With QCN at outputs (industry-standard), the CNMs are serialized by the output CP; in contrast, with QCN at inputs, all inputs can issue CNMs in parallel.

fabric in Fig.1. Four full-bandwidth flows (f1-f4) that source from different input ports (and different spines) target a common destination. The system first stabilizes, and then, during 100-200ms, a new congestive flow (f5), from a separate input port, joins the fan-in.

Figures 2(a,b) depict the per-flow throughputs. With QCN at switch outputs, as in Fig. 2(a), the allocation is grossly unfair. Within 10-100ms, flow f2 gets 1.5× the rate of f3, while f4 and f1 lie in the middle. Just after it arrives at 100ms, flow f5 plummets from 10 to approximately 2.5 Gb/s. However, f5 ends up with about three times the rate of f3. Finally, when f5 stops, at 200ms, the rates of flows f1-f4 increase and stabilize at a new unfair allocation. The unfairness of QCN at outputs is to be attributed to the sampling errors, which are described below. In contrast, QCN at inputs (Fig. 2(b)) achieves strikingly precise fair rates. Interestingly, the system finds the new fair shares even when flow f5 becomes active at 100ms, despite the fact that f5 starts from a much higher rate than what other flows have at that time.

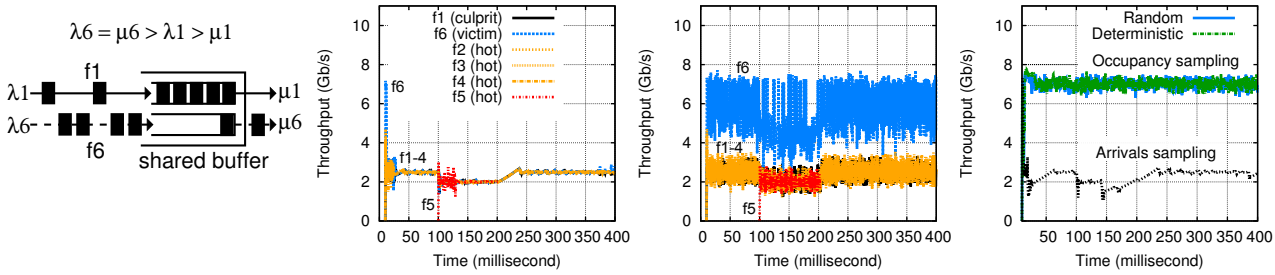
Figure 2(c) depicts the timing of the per-flow CNMs, separately for QCN CPs at the outputs and for QCN CPs at the inputs. Note that this figure does *not* present actual simulation data, but, nevertheless, it agrees with them quantitatively. With QCN at outputs, the CP at the bottleneck acts as a centralized serializer in a global feedback loop: It will stochastically sample an arbitrary interleaving of packets received from different flows/inputs. This process leads to transient unfairness episodes, whereby the same source may be notified not only earlier than its competitors at the bottleneck, but also repeatedly (two or more times in a row). Hence such sources may be rate-limited earlier and stronger. In contrast, with QCN at inputs, every congested flow is associated with a separate CP. These (N) CPs generate CNMs *in parallel*, each in proportion to the arrival rate of the corresponding flow. Thus, our system can sample the flows N times faster than the standard QCN. Working on the individual flows, before their arbitrary interleaving at the output, the proposed solution avoids the stochastic serialization of CNMs that is present with QCN at outputs, and yields fair flow throughputs.

IV. Selecting a Flow to Throttle

Beyond the statistical errors of a single congestion point, there is a fundamental limitation in the flow sampling method of QCN. As outlined in Fig. 3(a) and further described below, QCN may work correctly with FIFO-scheduled buffers, but it does not do so with an arbitrary service policy.

A. QCN arrival sampling: Policing flow speed

Consider the fan-in scenario of the previous section but now with one of the inputs hosting two flows instead of one: flow f1, which targets the congested output as before, at a rate of 3 Gb/s, and a new flow f6,



(a) Illustration of arrival sampling vs. occupancy sampling

(b) Per-flow throughputs: arrival-sampling QCN at inputs

(c) Per-flow throughputs: PAUSE-only

(d) Victim flow (f6) throughput with QCN at inputs; comparison of flow-marking schemes

Figure 3. (a) Two flows, f1 and f6, arrive at the same input buffer, but depart in non-FIFO order. Flow f6 has a greater arrival rate than f1, but does not build a backlog because its departure rate is just as big. However, flow f1 backlogs because its departure rate is small. QCN arrival sampling will direct most CNMs to f6, because its frames arrive more frequently. In contrast, occupancy sampling sends most CNMs to the flow with the higher backlog, namely, f1. (b-d) Same scenario as in Fig.2, but with one input hosting an additional flow, f6, which targets an uncongested output. Experiments on the network of Fig. 1 with 10G servers.

which targets an unrelated, uncongested output at 7 Gb/s. Thanks to the VOQs that are implemented at their input buffers, the two flows may depart in non-FIFO order. Ideally, flow f1 should achieve between 2 and 2.5 Gb/s, depending on whether flow f5 is also active (true for 100-200ms), and flow f6 should stay at 7 Gb/s throughout the experiment.

Surprisingly, as shown in Fig. 3(b), when we enabled QCN at the inputs, the two flows had equal rates. Deciding to send the congestion notification to the source of the most recently arrived frame, QCN penalizes a flow, f_n, based on its contribution $\lambda_n(t)$ to the overall arrival rate $\lambda(t) = \sum_{i=1}^{N} \lambda_i(t)$, ignoring its departure rate $\mu_n(t)$. Therefore, standard QCN sends congestion notifications to flows in proportion to their arrival rates. Applying this to our current example suggests that if flow f6 has a higher rate than f1 it will also have a higher probability to receive a congestion notification. But the input buffer occupancy cannot stabilize before the rate limiter of f1 has converged at f1's departure rate. As a result, f6 is also throttled towards f1's fair share. Note that in this scenario the PAUSE-only solution performs better than standard-QCN at the inputs, as shown in Fig. 3(c). Without rate limiters, the rates of the flows are indiscriminately modulated by PAUSE. Effectively, because flow f6 has an arrival rate $\frac{7}{3}\times$ higher than that of f1, also its departure rate is $\frac{7}{3}\times$ higher.

B. *QCN occupancy sampling: Policing flow backlog*

In this section, we modify QCN marking to make it compatible with our notion of congestion points at the inputs of switching fabrics. The key idea is to use the rate *mismatch* $\lambda_n(t) - \mu_n(t)$ as a discriminator when selecting which flow to throttle. In addition, our solution exploits the fact that a buffer acts as a rate mismatch integrator. In a lossless system, the contribution of flow f_n to the congestion point buffer occupancy with given initial condition is $q_n(t) = q_n(0) + \int_0^t (\lambda_n(\tau) - \mu_n(\tau))\, d\tau$.

The methods that we describe below use the input buffer occupancy of a flow as a cost function. As done in standard QCN, we generate a congestion notification message in response to the arrival of I_s bytes of payload [5], if the buffer is found congested, i.e. if the feedback value is positive ($F_b(t) > 0$). But instead of sending the congestion notification to the source of the frame that just arrived, we send it to the flow with the *highest occupancy* in the buffer monitored. Our first method, *deterministic occupancy sampling*, assuredly identifies the flow with the largest (average) rate mismatch. A graphical representation of how it resolves the problems of arrival sampling is provided in Fig. 3(a). In principle, there can be as many flows active in the congestion point as there are buffer slots. Therefore, maintaining a priority queue to sort flows may not scale well with increasing port speeds and buffer sizes.

Our second method, *random occupancy sampling*

eliminates the priority queue, and selects a culprit by *randomly picking* one occupied buffer slot and locating the corresponding frame header. The random selection will pick a particular flow with a probability given by the fraction of the overall congestion point buffer occupancy $q(t) = \sum_{i=1}^{N} q_i(t)$ taken by this flow. Hence, random QCN occupancy sampling has an (instantaneous) flow sampling probability $P_n^{(s)}(t) = q_n(t)/q(t)$. Observe that random occupancy is stateless in the sense that it does not keep track of the flow buffer occupancies.

Figure 3(d) shows the throughput of the innocent flow, f6, in the fan-in scenario, in which arrival sampling failed. As can be seen, with occupancy sampling, be it deterministic or random, flow f6 achieved its 7 Gb/s fair share. This is a huge improvement over standard QCN (arrival) sampling, which bounds f6 to the rate of f1.

Although the deterministic and random variants of occupancy sampling yield equal throughputs in this experiment, they do not perform exactly the same. Additional results in [8] show that random occupancy sampling issued some (but only few) CNMs to the victim flow f6, whereas the deterministic variant did not issue any. Nevertheless, this difference has no impact on the throughput behavior.

V. Implementation

We have implemented the leaf and spine switches using 32nm technology in 19.7×19.7 mm^2 and 18.4×18.4 mm^2, respectively. All switches operate at 454 MHz, and their power consumption is approximately 105 W (leaf) and 155 W (spine). The fall-through frame latency inside the fabric is approximately 1μs.

The QCN random occupancy sampling is implemented at the (100G) ingress ports of leaf switches. QCN has a fixed cost, whether one positions the CPs at the inputs or at the outputs of a switch. In our implementation, we have independent CP instances per server-side port and priority level. Every such port also has a CulpritArray with as many words as there are buffer units for the port, 12K in our fabric. These words keep the priority level of their corresponding (stored) frame and a pointer to the buffer unit that stores its header, where the Ethernet source address can be found. When I_s new bytes have been received on a priority level, and the corresponding feedback value is positive, the CP instance generates random addresses to sample the CulpritArray until it hits a valid word. This search is repeated until a frame of the targeted priority level is found. Having identified a proper culprit frame, the CP instance uses its source address to generate the CNM. Each CP instance generates CNMs at a peak rate of one CNM per 100 frames received.

The word size in our implementation is 20 bits (pointer to head buffer, priority level, valid-bit, ECC/parity), thus the overhead is less than 1% of other data stored per buffer (256B payload plus VOQ structures.) The width of the CulpritArray SRAM is 320 bits, thus a single read gives information on 16 buffer units that can be processed in parallel. Finally, if random addresses fail to find a culprit after a number of searches, then our engine sequentially searches the CulpritArray word by word.

VI. Related Work

AF-QCN, another proposal that improves on the standard QCN, is presented in [11]. AF-QCN augments QCN congestion points with per-flow rate measurements, enabling weighted fairness of link bandwidth. However, AF-QCN does not address the sampling limitations of QCN, which we identified and tackled in the present paper via a stateless solution. Related in concept with occupancy sampling, albeit in the lossy context, are derivatives of the push-out method [12].

In a technical report [8], we (i) demonstrate that QCN occupancy sampling works well on 100G links, (ii) study how to reduce the duration of PAUSE using a *keep-alive* mechanism, and (iii) present the evolution of input and output buffer occupancies. Also, [9] shows that our QCN architecture adapts well to multicast traffic, whereas standard QCN at switch outputs may needlessly penalize multicast flows.

Infiniband congestion control ignores the velocity of the queue buildup when determining congestion, but marks *every* packet once a queue has become congested; thus, it overcomes the unfairness introduced by the random sampling of QCN. Infiniband switches cannot generate packets, and they instead set a flag in congested packets, causing the endpoint receivers to send the congestion notifications back to the culprit sources. However, in doing so, the feedback control loop is stretched, making it difficult to keep it stable [13].

RECN is another interesting congestion control method for interconnection networks [14]. However, RECN dynamically allocates per-flow queues inside the switching nodes, which complicates switch design. RECN also assumes proprietary flow control messages

being exchanged among nodes; therefore, it is not clear how RECN can work in Ethernet networks that use PAUSE flow control.

VII. Conclusions & Future Work

Our work builds on the newly standardized QCN [5]. We have introduced an alternative congestion management architecture that positions the QCN congestion points at the inputs rather than outputs of modern switches and switching fabrics. The decisive advantage of our proposal is its deterministic fairness under fan-in traffic scenarios, which are typical in datacenters. In addition, we have described *(i)* a new QCN-compatible congestion marking scheme, suitable for the scheduled departures out of the switch input buffers, and *(ii)* a practical stateless implementation that randomly picks an occupied unit within the buffer to identify congestive flows.

We have presented an exemplary embodiment of our results in server-rack, fat-tree networks that offer full bisection bandwidth. Nevertheless, our results are applicable to other single- or multi-stage switch or network supporting converged Ethernet and QCN. Additional work is required to examine the performance of QCN at the inputs of internally blocking networks. Our preliminary results on Dragonfly-like topologies [15] show that QCN occupancy sampling at the inputs can throttle *internally bottlenecked* flows, as the VOQs of these flows will backlog, whereas QCN at the outputs remains oblivious of the internal backlogs. A difficulty arises because, due to backpressure, innocent flows may also backlog, albeit at a lower pace.

References

[1] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph, "Understanding TCP Incast Throughput Collapse in Datacenter Networks," in *Proc. ACM SIGCOMM Workshop for Research on Enterprise Networks (WREN)*, 2009.

[2] J. Dean and L. A. Barroso, "The Tail at Scale," *Communications of the ACM*, vol. 56, pp. 74–80, 2013.

[3] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz, "DeTail: Reducing the Flow Completion Time Tail in Datacenter Networks," in *Proc. ACM SIGCOMM*, Helsinky, Finland, August 2012.

[4] D. Crisan, R. Birke, N. Chrysos, C. Minkenberg, and M. Gusat, "zFabric: How to Virtualize Lossless Ethernet?" in *Proc. IEEE Cluster*, Madrid, Sept. 2014.

[5] *802.1Qau - Virtual Bridged Local Area Networks - Amendment: Congestion Notification*, IEEE Std., 2010. [Online]. Available: http://www.ieee802.org/1/pages/802.1au.html

[6] M. Alizadeh, B. Atikoglu, A. Kabbani, A. Lakshmikantha, R. Pan, B. Prabhakar, and M. Seaman, "Data Center Transport Mechanisms: Congestion Control Theory and IEEE Standardization," in *Proc. Allerton Conference on Communication, Control, and Computing*, Sept. 2008.

[7] S.-T. Chuang, A. Goel, N. McKeown, and B. Prabhakar, "Matching Output Queueing with a Combined Input/Output-Queued Switch," *IEEE Journal Selected Areas in Communications*, vol. 17, no. 6, pp. 1030–1039, Jun. 1999.

[8] N. Chrysos, N. Neeser, R. Clauberg, D. Crisan, K. Valk, C. Basso, C. Minkenberg, and M. Gusat, "Unbiased QCN for Scalable Server-Fabrics," *IBM Research Report, RZ3880*, October 2014.

[9] N. Chrysos, F. Neeser, B. Vanderpool, K. Valk, M. Rudquist, T. Greenfield, and C. Basso, "Integration and QoS of Multicast Traffic in a Server-Rack Fabric with 640 100G Ports," in *Proc. ACM/IEEE ANCS*, Marina del Rey, CA, Oct. 2014.

[10] Y. Tamir and G. L. Frazier, "Dynamically-Allocated Multi-Queue Buffers for VLSI Communication Switches," *IEEE Transactions on Computers*, vol. 41, no. 6, pp. 725–737, 1992.

[11] A. Kabbani, M. Alizadeh, M. Yasuda, R. Pan, and B. Prabhakar, "AF-QCN: Approximate Fairness with Quantized Congestion Notification for Multi-tenanted Data Centers," in *Proc. IEEE Hot Interconnects*, 2010.

[12] R. Pan, B. Prabhakar, and K. Psounis, "CHOKe - A Stateless Active Queue Management Scheme for Approximating Fair Bandwidth Allocation," in *Proc. IEEE INFOCOM*, March 2000, pp. 942–951.

[13] G. Pfister, M. Gusat, W. Denzel, D. Craddock, N. Ni, W. Rooney, T. Engbersen, R. Luijten, R. Krishnamurthy, and J. Duato, "Solving Hot Spot Contention Using Infiniband Architecture Congestion Control," in *Proc. HP-IPC*, 2005.

[14] J. Duato, I. Johnson, J. Flich, F. Naven, P. J. García, and T. N. Frinós, "A New Scalable and Cost-Effective Congestion Management Strategy for Lossless Multistage Interconnection Networks," in *HPCA*. IEEE Computer Society, 2005.

[15] J. Kim, W. J. Dally, S. Scott, and D. Abts, "Technology-Driven, Highly-Scalable Dragonfly Topology," *ACM SIGARCH Computer Architecture News*, vol. 36, no. 3, pp. 77–88, 2008.

Nikolaos Chrysos (nchrysos@ics.forth.gr) is with FORTH - Hellas, working on interconnection networks for high-density, Exascale-class datacenters and high-performance computers, and on virtualized heterogeneous systems. He holds MSc and PhD degrees from the University of Crete, Greece. From 2009 to 2014, he was with IBM Research - Zurich, where he contributed to the design and implementation of server-rack fabrics for 100G Ethernet.

Fredy Neeser (nfd@zurich.ibm.com) is a researcher at IBM Research - Zurich working on high-speed interconnects and datacenter networking. He contributed to Remote Direct Memory Access (RDMA) and Soft-RDMA/iWARP, a software implementation of IETF's iWARP stack, and to the design of a chipset for 100 Gb/s Converged Enhanced Ethernet (CEE), focusing on flow and congestion control.

Dr. Rolf Clauberg (cla@zurich.ibm.com) is a researcher with IBM Research - Zurich working on cloud and computing infrastructure. He holds a doctor of science degree and a diploma in physics from the University of Cologne/Germany. Both thesis works were performed at Forschungszentrum Julich in Germany.

Daniel Crisan (daniel.crisan@gmail.com) is a Site Reliability Engineer at Google Switzerland. Previously he worked at IBM Research Zurich in the System Fabrics group. His research interests include datacenter networking and virtualization. He holds a PhD from ETH Zurich and a M.Sc. in Computer Science from EPFL.

Kenneth M. Valk (kmvalk@us.ibm.com) received his B.S. and M.E. degrees, both in Computer & Systems Engineering, from Rensselaer Polytechnic Institute, Troy, NY, in 1991 and 1992 respectively. He currently works for International Business Machines where he has worked on network and cache-coherence hardware since 1992.

Claude Basso (basso2@fr.ibm.com) is an IBM Distinguished Engineer at IBM Research working on Cloud Data Center networking. Claude graduated from the "Ecole Nationale Superieure d'Informatique and Mathematiques Appliquees", Grenoble, France in 1981.

Cyriel Minkenberg (sil@zurich.ibm.com) is a Research Staff Member at IBM Research - Zurich. He holds MSc and PhD degrees from the Eindhoven University of Technology, The Netherlands. His current research focuses on interconnection networks for exascale high-performance computing systems and extreme-scale cloud data centers.

Mitch Gusat (mig@zurich.ibm.com) is a researcher and Master Inventor at IBM Research - Zurich. He holds Masters in CE, resp. EE, from the University of Toronto and Timisoara. He is member of ACM, IEEE, and holds a few dozen patents related to SDN, transports, HPC architectures, switching and scheduling.