RZ 3880 (# ZUR1410-006) 10/02/2014
Computer Science 10 pages

Research Report

Unbiased QCN for Scalable Server-Fabrics

Nikolaos Chrysos*, Fredy Neeser*, Rolf Clauberg*, Daniel Crisan*,
Kenneth Valk‡, Claude Basso‡, Cyriel Minkenberg*, Mitch Gusat*

*IBM Research – Zurich
8803 Rüschlikon
Switzerland

‡IBM Systems & Technology Group,
Rochester
USA

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 a economical alternative for high-performance computing clusters. Ethernet would traditionally drop packets in the event of congestion, but IEEE is striving to introduce lossless class services to enable the convergence of storage, cluster, and IP networks. Losslessness is a simple, well-known concept that may offer substantial benefits, 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), which IEEE has standardized, by making it compatible with emerging server-rack fabrics. 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, and propose a novel QCN-compatible marking scheme, namely occupancy sampling. Finally, we also study the interactions between QCN and PAUSE. We have implemented our methods in a next-generation, server-rack fabric with 640 100G ports. Our experiments on both 10G and 100G links show that the combined approach rectifies QCN's fairness and reduces the PAUSE period. Effectively, the proposed enhancements are a significant step forward in scaling converged datacenter networks.**

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 and cluster 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 unacceptable delay on 63% of the times it is deployed even if only one percent of the (network) flows is delayed; recovering from packets 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], [4]. 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 [5].

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, similarly to virtual channel flow control in multiprocessor networks [6].

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—similar to the flits of two colliding packets in wormhole routing— 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)* [7]. 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 rate limiters for the congested flows. In response to CNMs, a QCN rate limiter 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 solution for eliminating saturation trees, but its implementation 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) We achieve fair throughputs with QCN in fan-in scenarios by re-placing the congestion points from the switch outputs to the switch inputs.
2) We propose a new flow marking scheme 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 spine-leaf fabric that comprises 640 100G Ethernet ports.

II. Fair QCN for Large Switching Fabrics

A packet switch may correspond to a single-stage crossbar or shared memory chip, or it may be constructed from smaller switching elements forming a multi-stage interconnection network, referred to as a switching fabric below.

In practice, scalable switching fabrics must apply some form of per-input buffer allocation. In such combined-input-output-queued (CIOQ) architectures [8], 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 outputs. Although VOQ-based architectures avoid head-of-line (HOL) blocking, they cannot prevent buffer hogging. In practice, the many VOQs at each input share a buffer memory (DAMQ) [9]. Therefore, a VOQ that makes slow progress can monopolize its input buffer. An input backlogged in this way will assert PAUSE to its upstream CNA, stopping all flows in the same priority level, irrespective of their destination. Ethernet congestion control, QCN, is designed to throttle the congested VOQ (flow) and protect the innocent ones, thus is complementary to PAUSE flow control and a prerequisite for converged networks.

A. Emerging distributed server-rack (CIOQ) fabrics

In this paper, we describe the QCN architecture that we have implemented in a server-rack fabric.

The fabric, shown in Fig. 1, has dedicated input and output buffers per port and per priority level, and extends to the backplane of a cluster of server racks. It is thus a distributed, multi-stage realization of a CIOQ switch, built around a *flattened*, spine-leaf (fat-tree) topology. The leaf switches are integrated into the backplane of the racks, and each one constitutes the (Ethernet) network-edge switch for five (5) servers, providing 100 Gb/s of bandwidth to each server on a single link. Our fabric currently supports four racks, with 32 edge switches each, thus a total of 640 100G ports.

The leaf switches collectively act as a distributed Ethernet bridge, and are responsible for MAC learning and frame forwarding. At its ingress interface, a leaf switch stores the incoming frames into VOQs (input buffers), segments them into variable-size fabric-internal packets (or cells), which are then injected into the interconnect. Each packet can use any of the available leaf-to-spine links, as enforced by a packet-level spraying mechanism

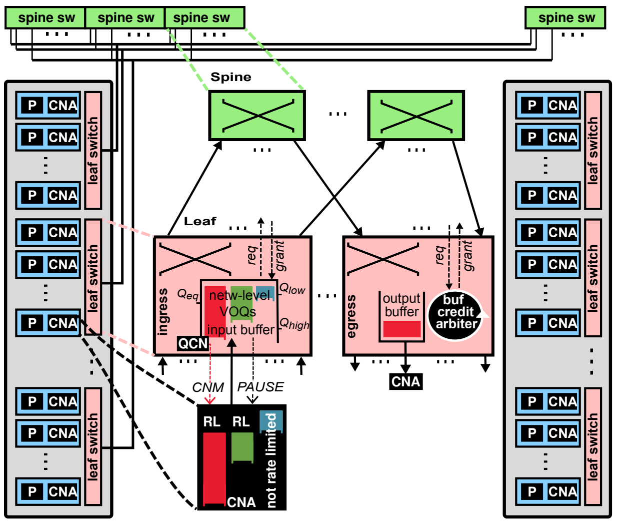


Figure 1. A distributed server-rack fabric with an exploded view of two leaf switches. The ingress leaf switch is shown with its input buffer hosting the VOQs. An adapter (CNA) connecting to one port of the ingress leaf switch is also shown. When the input buffer occupancy reaches Q_{high}, the input port sends a STOP to the CNA, which does not forward new data until it receives a GO.

that overcomes the limitations of ECMP-style routing alternatives [10], [11]. The original Ethernet frames are 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 connections among the leaf switches. We use 32 spines in our four-rack system. They feature small input and output buffers, which are flow controlled using hop-by-hop backpressure. There are 32 25 Gb/s links per leaf switch connecting to the spines. Thus, the fabric features an over-provisioning ratio of 8:5, which accommodates any internal overhead and also leaves 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 at the destination leaf switches: 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 [12].

Although this switching fabric looks far more complicated than a standalone switch module, its edge-to-edge, scheduled flow control, together with the multi-path routing and the internal over-provisioning make it a large, fair CIOQ switch. The details of this fabric are the subject of a future publication; here we mainly focus on its QCN architecture.

Table I
SIMULATION PARAMETERS

	Parameter	*Value*
Switch	Input buffer	150 KB
Switch	Output buffer	150 KB
PAUSE	Q_{high} (STOP threshold)	110 KB
PAUSE	Q_{low} (GO threshold)	44 KB
QCN	Q_{eq} (equilibrium)	60
QCN	w (weight for Q_δ)	2
QCN	RL reaction time	2.4 μs
QCN	I_s (base sampling interval)	150 KB

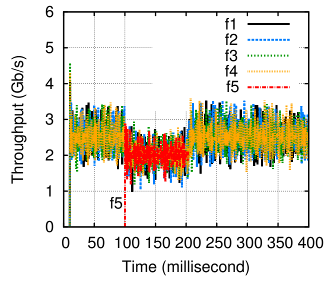
Implementation and tests that follow: We have implemented the leaf and spine nodes using 32nm technology in 19.7×19.7 mm^2 and 18.4×18.4 mm^2 respectively. They both operate at 454 MHz, and their power consumptions are approximately 105 W and 155 W. For spine switches, most of the power (130W) is consumed by moving data across the chip I/O interfaces, despite using state-of-the-art low-power High-Speed Serial (HSS) technology. Preliminary tests on the real hardware have verified the proper functioning of the QCN units. In this paper, we report performance results from simulations using a detailed C++ computer model.

Table I reports the main QCN parameters. 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. The fall-through frame latency is approx. 1μs.

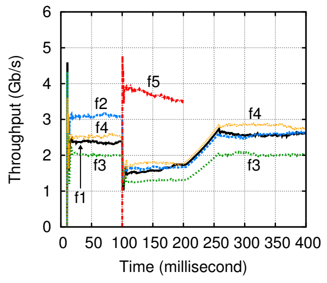
To simplify the comparisons with recent literature and the IEEE 802 archives, we first consider servers with 10G interfaces and standard QCN parameter settings [7]. Later, in Sec. V, we will experiment on 100G links.

B. *Resolving QCN unfairness: Charge them at the door*

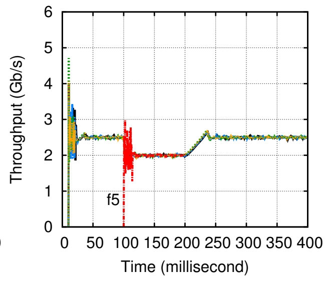
Standard QCN is based on an idealized output-queued switch and allocates one CP at each output [13][7, Sec. 30.2.1]. This switch architecture was selected by IEEE as the most generic one—almost all switch architectures have output queues.



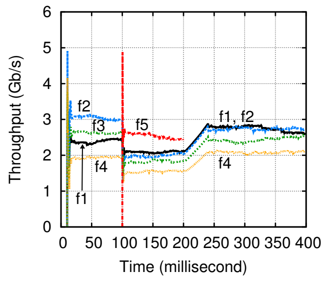
(a) PAUSE only



(b) QCN at switch outputs



(c) QCN at switch inputs



(d) QCN at switch inputs & outputs

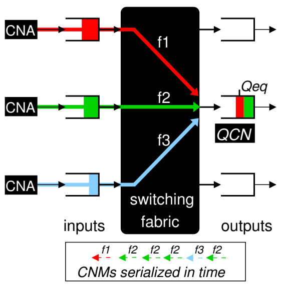
Figure 2. Per-flow throughputs under a fan-in traffic scenario, where flows f1-f5 target the same destination. Flow f5 is active between 100 and 200ms. All configurations use PAUSE to sustain lossless operation. Experiments on the network of Fig. 1 with 10G servers.

The trend towards switching fabrics with input buffers suggests the possibility of installing QCN CPs at switch inputs, as shown in Fig. 1, instead of at switch outputs. As we demonstrate below, this has a number of advantages. Most importantly, it corrects QCN unfairness under typical fan-in scenarios.

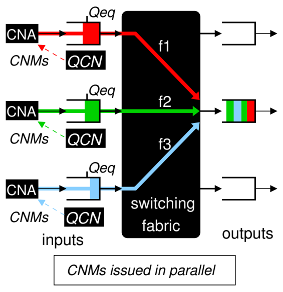
QCN at an input buffer should detect overload, mark and throttle the culprit flow(s), and, using an appropriate Q_{eq}, keep the input buffer backlog below the Q_{high} threshold, thus avoiding the exertion of PAUSE. Note that the CNMs generated at inputs do not have to traverse the switch (from one port to another). Hence they neither consume fabric-internal bandwidth nor incur any additional delays.

The two alternative placements of congestion points are depicted in Fig. 3. Assuming that flows in the figure start from the same rates, they should receive a comparable amount of CNMs within every time window to achieve equal bandwidths. The standard QCN 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 row). Hence such sources may be rate-limited earlier and stronger. As our results show, these flows do not practically recover, leading to massive disruptions of fairness, longer delay tails and unbalance.

In our proposal, shown in Fig. 3(b), 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: 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, our solution avoids the stochastic serialization of the standard one.



(a) QCN at outputs (industry-standard)



(b) QCN at inputs

Figure 3. A fan-in scenario comprising three flows: (a) With QCN at outputs, the output CP serializes the CNMs based on the stochastic frame arrivals at the output; effectively, an unlucky flow may receive a burst of CNMs. (b) With QCN at inputs, there is a separate CP for each flow; thus, the flows receive CNMs in parallel, each in proportion to its arrival rate.

In our first experiment, we simulate a fan-in scenario on top of the rack 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.

We evaluate three different configurations. The first configuration uses PAUSE flow control; the second configuration additionally places QCN congestion points at switch outputs, and the third configuration places the congestion points at switch inputs.

Figure 2 depicts the per-flow throughputs (arrivals at switch input buffers). With QCN disabled, see Fig. 2(a), flows fluctuate wildly around their (dynamic) fair shares, as PAUSE flow control modulates their arrivals into a series of variable-width on/off pulses. With QCN at switch outputs, in Fig. 2(b), the allocation is grossly unfair: within 10-100ms, flow f2 gets $1.5\times$ 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 is to be attributed to the aforementioned statistical errors: the flows that are sampled first are treated badly. Typically, such over-throttled flows will try to consume any unused bandwidth. However, during the additive increase phase of QCN's autonomous recovery, rate limiters that start at a higher value typically make both faster and greater steps up.

In contrast, QCN at inputs, shown in Fig. 2(c), 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. As noted above, moving the CPs to the inputs eliminates the stochastic unfairness introduced by sampling an aggregation of flows at an output. This, in combination with the fair service of flow VOQs, enforced by the output credit arbiter, results in nearly perfect bandwidths.

Lastly, in Fig. 2(d), we examine the possibility of installing QCN congestion points at both switch inputs and outputs. As can be seen, the results are not qualitatively different from those in Fig. 2(b), where congestion points are installed only at switch outputs.

III. Selecting a Flow to Throttle

We showed that QCN at switch inputs improves fairness by canceling the statistical errors of a single congestion point. But beyond these statistical errors, there is a fundamental limitation in the flow sampling method of QCN. As outlined in Fig. 4 and further described below, QCN may work correctly with FIFO-scheduled buffers, but it does not do so with arbitrarily scheduled buffers.

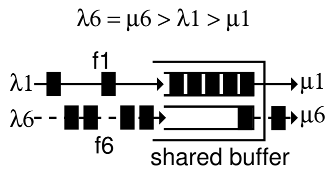


Figure 4. Two flows, f1 and f2, arrive at the same input buffer, but depart in non-FIFO order. Flow f2 has greater arrival rate than f1, but does not build a backlog because its departure rate is just as big. However, flow f1 backlogs because its departure rate is small. QCN arrival sampling will direct most CNMs to f2, because its frames arrive more frequently. In contrast, occupancy sampling sends most CNMs to the flow with the higher backlog, namely, f1.

A. *QCN arrival sampling: Policing flow speed*

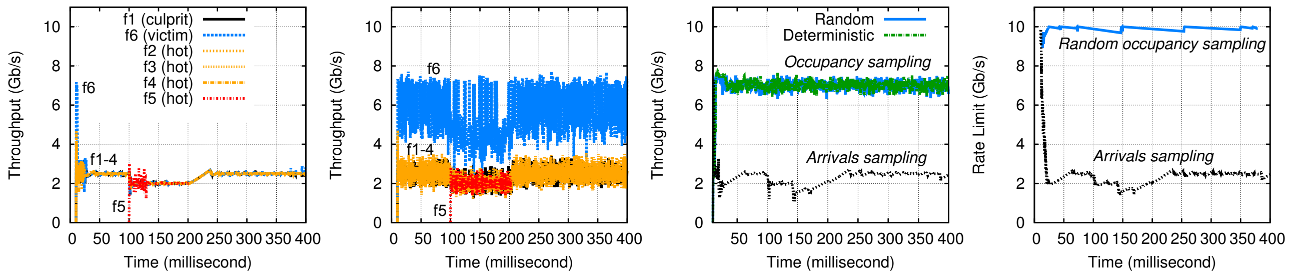
Consider the fan-in scenario of the previous section but now with one of the inputs hosting two flows instead of one: flow f1, which targets the congested output as before, at a rate of 3 Gb/s, and a new flow f6, which targets an unrelated, uncongested output at 7 Gb/s.

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. 5(a), 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 $\mu_n(t)$. (For a complete analysis, please refer to [14, Sec. 2].) 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 f1 has converged at its output fair rate. As a result, f6 is also pulled 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. 5(b). Without rate limiters, the rates of the flows are indiscriminately modulated by PAUSE.



(a) Per-flow throughputs: arrival-sampling QCN at inputs (b) Per-flow throughputs: PAUSE-only (c) Victim flow (f6) throughput with QCN at inputs; comparison of flow-marking schemes (d) Rate limiter for victim flow (f6) with QCN at inputs; comparison of flow-marking schemes

Figure 5. Same scenario as in Fig.2, but with one input hosting one additional flow, which targets a uncongested output. Experiments on the network of Fig. 1 with 10G servers.

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.

Multicast traffic: An additional advantage of the proposed architecture is seen under multicast traffic. Using the standard QCN architecture (Fig. 3(a)), a *multicast* frame heading to two destinations can be sampled twice, and thus generate two (or more depending on to its fan-out degree) CNMs. As shown in [15], this multiplication of CNMs can result in unfair treatment of multicast flows. In contrast, with QCN at inputs (Fig. 3(b)), every multicast frame is sampled on par with unicast frames.

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 $q_n(t) = q_n(0) + \int_0^t (\lambda_n(\tau) - \mu_n(\tau))\, d\tau$.

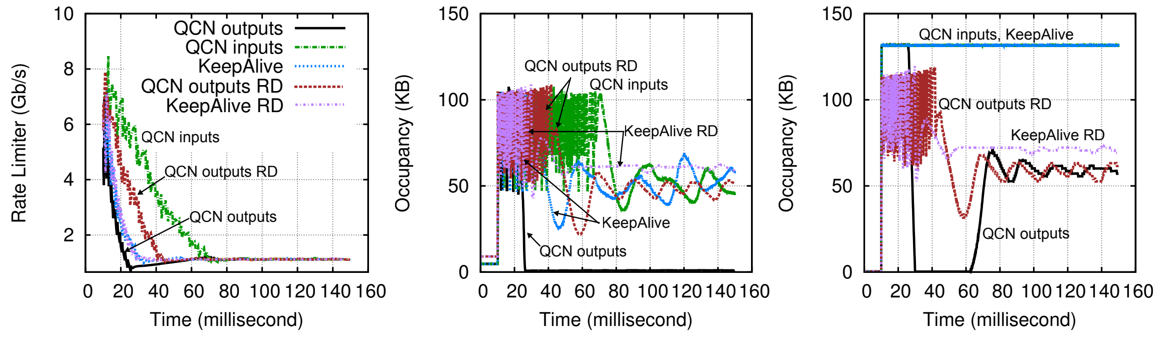
The methods that we describe below use the input buffer occupancy of a flow as a cost function. As done in standard QCN, we generate a congestion notification message in response to the arrival of I_s bytes of payload [7], if the buffer is found congested, i.e. if the feedback value is positive ($F_b(t) > 0$). 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.

This 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. 4. 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 5(c) 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 yielded equal throughputs in this experiments, they did not perform identically. Figure 5(d), depicting the rate limits of the two flows, shows that the random variant issued some (but only few) congestion notifications to the victim flow f6, whereas the deterministic one did not issue any. (The absence of points for flow f6 indicates that the corresponding rate limiter was not allocated.) Deterministic occupancy sampling spots the most congested flow and keeps throttling it until



(a) Rate limiters' values (b) Input buffer (c) Output buffer

Figure 6. Single flow experiment. The capacity of the targeted output suddenly drops to 1 Gb/s. Comparison of the convergence time of QCN at inputs, with and without keep-alive, with that of QCN at outputs. Experiments on the network of Fig. 1 with 10G servers.

either some other flow takes over in buffer occupancy or congestion ceases. In contrast, with random occupancy sampling, the CNM rates are governed by the occupancies ratios $q_n(t)/q(t)$.

IV. Interactions between QCN and PAUSE

In practice, at the onset of a congestive episode, QCN coexists with PAUSE. As PAUSE modulates the arrivals at switch inputs, it can modify the backlog of the buffer that QCN monitors.

To expose the complex interactions between QCN and PAUSE, we launched a single uncongested flow and suddenly reduced the capacity at its destination to 1 Gb/s. Note that, in this example, QCN arrival sampling performs identically with occupancy sampling because only one flow is present.

Figure 6(a) plots the flow rate limit. As can be seen, QCN at the switch outputs throttles the flow within 15ms, whereas, QCN at the switch inputs needs 4× as long. This delayed reaction is undesirable as it prolongs the PAUSE activity.

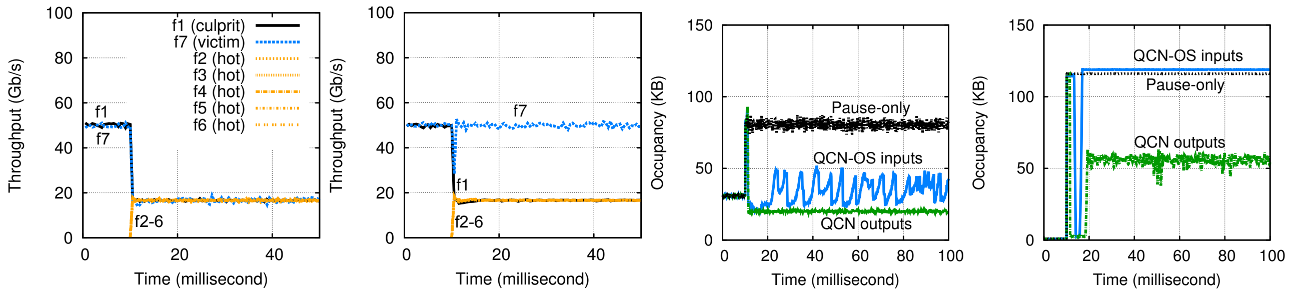
In this scenario, the input and output congestion points deal with a single flow, therefore its natural to ask why QCN at the inputs delays so much longer. The answer lies in the PAUSE activity, which is present at the input but not at the output buffer. Looking at Fig. 6(b), we see that initially the input backlog swings rapidly between the Q_{high} and Q_{low} PAUSE thresholds. Effectively, the input congestion point may sometimes sample a negative Q_δ, which can refrain it from issuing a congestion notification. Oblivious of the real situation, the reaction point interprets the absence of a congestion notification (after having sent I_s bytes) as an opportunity to increase its rate limit, therefore delaying the convergence.

However, as shown in Figure 6(c), while the flow has not yet been throttled adequately, the output buffer, which is flow-controlled using switch-internal credits, stays perpetually full. Such a consistent state allows the output to exert a stream of congestion notifications that stabilize the rate limiter in a much shorter time.

To reduce the PAUSE activity in systems with QCN at inputs, we can enable *QCN keep-alive*. Normally, while the input has exerted PAUSE, the congestion point does not receive any new frames and therefore does not generate any CNM. But as occupancy sampling does not rely on frame arrivals, a clock source can be used during PAUSE to keep generating congestion notifications as long as the buffer stays congested.

In our baseline configuration, during PAUSE, the keep-alive clock triggers a sampling of the buffer with a period of I_s bytes. As can be seen in Fig 6(a), this shortens the convergence delay, bringing it in par with that of QCN at switch outputs. It is interesting to note here that the keep-alive mechanism also averts the unwanted timer-induced recovery of rate-limiters while a CNA is PAUSEd.

We repeated these experiments enabling edge-to-edge reliable delivery inside the fabric. In this configuration, each input keeps a copy of the frames that it injects until acknowledgments signal that the frames have been properly received by their output and forwarded on the outgoing link. As shown in Fig 6(a), with reliable delivery (RD) enabled, standard QCN at the outputs delayed much longer to converge. The reason is that, as shown in Fig 6(c), the occupancy of the output buffer



(a) PAUSE-only (QCN arrival sampling at inputs performs the same) (b) QCN-OS at inputs with keep-alive (c) Input buffer of flows f1 and f7 (d) Buffer of congested output

Figure 7. Per-flow throughputs and buffer occupancies in input-generated hotspot test. Flows f1 and f6 start from the same input port at 5 Gb/s each. Then at 10ms, flows f2-f5, each coming from a separate input, target the destination of flow f1. Experiments on the network of Fig. 1 with 100G servers.

fluctuates wildly, mirroring the fluctuations at the input buffer, which are caused by PAUSE: the number of packets at the input buffer upper bounds the number of packets at the output because all packets present at the output must also be present at the input.

V. Evaluation of QCN at 100G Links

In this section we further evaluate our methods on the server-rack CIOQ fabric of Fig. 1 with 100G links. For the transition from 10G to 100G links, we have changed the following parameters of QCN: T (timer-reset cycle) from 10 to 2ms, R_{AI} from 5 to 15 Mb/s, and R_{HAI} from 50 to 250 Mb/s [13].

We consider an input-generated hotspot scenario with flows f1 and f7 sharing a switch input, and arriving at 50 Gb/s each. Then, at 10ms, flows f2-f6 are also activated, and target the destination of flow f1. As shown in Fig. 7, the PAUSE-only alternative performs really bad, assigning a rate of 16.6 ($\approx$ 100/6) Gb/s to victim flow f7. QCN-OS (random) at switch inputs results in much higher throughput (50 Gb/s) than both the PAUSE-only solution and arrival-sampling QCN at inputs.

Figures 7(c,d) plots the total buffer occupancies at the congested output and at the input of flows f1 and f7: with QCN at the outputs, the input buffer is almost empty, whereas with QCN-OS at the inputs, it stays around Q_{eq}. In contrast, with QCN-OS at switch inputs, the output buffer is almost full, whereas it is at around Q_{eq} for QCN at switch outputs.

VI. Complexity of Occupancy Sampling

QCN always has a fixed cost, whether one positions the CPs at the inputs or at the outputs of a switch. In our implementation of random occupancy sampling, 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 that priority level, the CP instance computes the corresponding feedback value: If it 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.

The word size in our implementation is 20 bits (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 the engine sequentially searches the CulpritArray word by word. As the congested priority levels are also backlogged, this procedure finds a culprit in reasonable time—our rate limiters have a response latency of 2.4 μs anyhow, hence generating the CNM promptly was not very critical.

VII. Related Work

AF-QCN, another proposal that improves on the standard QCN, is presented in [16]. 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 [17].

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 [18]. Both QCN and Infiniband use automatic rate recovery; in QCN this is implemented in a fashion similar to TCP CUBIC [19].

RECN is another interesting congestion control method for interconnection networks [20]. 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.

VIII. Discussions & Conclusions

Our work builds on the newly standardized QCN [7]. We have introduced an alternative congestion management architecture that positions the QCN congestion points at switch inputs rather than switch outputs. The decisive advantage of our proposal is its deterministic fairness under fan-in traffic scenarios, which are typical in datacenters. Furthermore, our proposal adapts well to multicast traffic, whereas standard QCN may needlessly penalize multicast flows.

In addition, we have *(i)* described 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 a flow as a congestive culprit. Finally, we have investigated the interactions between QCN and PAUSE activity, proposing the QCN keep-alive function to reduce the PAUSE activity.

We presented an exemplary embodiment of our results in server-rack, flattened, fat-tree networks, which are free of internal blocking. Nevertheless, our results are applicable to any single- or multi-stage switch or network supporting converged Ethernet and QCN. QCN occupancy sampling at the inputs can identify and throttle bottlenecked internal flows, as the VOQs of these flows will backlog because of flow control. In contrast, QCN at the outputs remains oblivious to such internal backlogs. Preliminary results in blocking, Dragonfly-like networks show that better discrimination is achieved when we enable edge-to-edge reliable delivery. A focus of our research will be to examine how the benefits of our architecture can be propagated to alternative networks.

IX. Acknowledgments

We would like to thank our colleagues in IBM Research–Zurich and in the hardware division of IBM (STG), as well as Charlotte Bolliger and Anne-Marie Cromack for proofreading the manuscript.

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 (WREN Workshop)*, 2009.

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

[3] D. Crisan, A. S. Anghel, R. Birke, C. Minkenberg, and M. Gusat, "Short and Fat: TCP Performance in CEE Datacenter Networks," in *Proc. IEEE Hot Interconnects*, 2011.

[4] 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*, 2012.

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

[6] W. J. Dally, "Virtual-Channel Flow Control," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 3, no. 2, pp. 194–205, 1992.

[7] *802.1Qau - Virtual Bridged Local Area Networks - Amendment: Congestion Notification*, IEEE Std., 2010. [Online]. Available: http://www.ieee802.org/1/pages/802.1au.html

[8] 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, 1999.

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

[10] A. Dixit, P. Prakash, Y. C. Hu, and R. R. Kompella, "On the Impact of Packet Spraying in Data Center Networks," in *Proc. IEEE INFOCOM*, 2013, pp. 2130–2138.

[11] N. Chrysos, F. Neeser, M. Gusat, C. Minkenberg, W. Denzel, and C. Basso, "All Routes to Efficient Datacenter Fabrics," in *Proc. INA-OCMC*, Berlin, Germany, January 2014.

[12] N. Chrysos, F. Neeser, M. Gusat, R. Clauberg, C. Minkenberg, C. Basso, and K. Valk, "Tandem Queue Weighted Fair Smooth Scheduling," *Design Automation for Embedded Systems, Springer*, March 2014.

[13] 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 CCC*, 2008.

[14] F. Neeser, N. Chrysos, R. Clauberg, D. Crisan, M. Gusat, C. Minkenberg, K. Valk, and C. Basso, "Occupancy Sampling for Terabit CEE Switches," in *Proc. IEEE Hot Interconnects*, 2012.

[15] 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*, Oct. 2014.

[16] 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.

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

[18] 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.

[19] S. Ha, I. Rhee, and L. Xu, "CUBIC: A New TCP-Friendly High-Speed TCP Variant," *ACM SIGOPS Operating System Review*, vol. 42, no. 5, pp. 64–74, July 2008.

[20] 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 *IEEE HPCA*, 2005.