Data Center Quantized Congestion Notification (QCN): Implementation and Evaluation on NetFPGA

Abdul Kabbani
Department of Electrical Engineering
Stanford University
Stanford, CA, 94305, USA
akabbani@stanford.edu

Masato Yasuda
System IP Core Research Laboratories
NEC Corporation
Kawasaki, Kanagawa, 211-8666, Japan
m-yasuda@ct.jp.nec.com

ABSTRACT
We recently proposed QCN, a Layer 2 congestion control mechanism, for high-speed data center networks (e.g., 10Gbps). QCN has been going through the IEEE standardization process and is officially finalized at this stage. As a Layer 2 protocol, QCN has the significant advantage of being easy to implement in hardware. By prototyping QCN on a NetFPGA board, we prove this claim in this paper as we implement the first QCN system. We furthermore demonstrate building a test-bed seed for conducting further QCN experiments with various network settings and topologies.

1. INTRODUCTION
The Data Center Bridging Task Group in the IEEE 802.1 Ethernet standards body has several active projects aimed at enabling enhancements to classical switched Ethernet so that it may provide more services. Details of the DCB Task Group’s projects are available at [1, 2]. Particularly relevant to the present paper is the Quantized Congestion Notification protocol (QCN), the essence of the Congestion Notification (CN, IEEE 802.1Qau) project. QCN is a Layer 2 congestion control mechanism, in which a congested switch can control the rates of Layer 2 sources whose packets are passing through the switch. Thus, QCN induces congestion control loops at Layer 2 similar to the well-known TCP/RED control loops at Layer 3.

In this paper, we present the design and implementation of the QCN system (NIC and Switch) on the NetFPGA platform [3, 4]. The NetFPGA platform is chosen because of its existing Ethernet ports with 1Gbps bandwidth per port.

We demonstrate the following through our QCN prototype:
1) The first complete operation of QCN realized at 1Gbps hardware with a 125MHz clock. The correctness of the hardware implementation is further verified by simulations.
2) The QCN on-chip IP core is small, portable (not platform-dependent and not limited to 1Gbps hardware), and simple.
3) Simple components can be added to the implementation allowing us to easily build a test-bed for conducting further experiments with:
   (a) Scalable data center topologies
   (b) Tunable network settings: per link round trip time, capacity, and buffer depth.
   (c) Tunable QCN parameters
   (d) The ability to trace and analyze all major network and QCN variables.

The rest of the paper is organized as follows. Section 2 describes the QCN mechanism. Section 3 discusses the QCN design on the NetFPGA platform. We evaluate the performance of this implementation in Section 4, and we conclude and outline further work in Section 5.

2. QCN MECHANISM
The QCN algorithm has been developed to provide congestion control at the Ethernet layer. QCN allows a primary Layer 2 bottleneck to alleviate congestion by directly signaling those Layer 2 sources whose packets pass through it to reduce their rates.

As opposed to the Internet, the Ethernet is more constrained and has challenging performance requirements. We summarize the two sets below and refer the reader to previous work [5] for further details.

Ethernet constraints:

No per-packet ACKs in Ethernet. This has several consequences for congestion control mechanisms: (i) Packet transmission is not self-clocked as in the Internet, (ii) path delays (RTT) are unknown, and (iii) congestion must be signaled by switches directly to sources.

Packets may not be dropped. Ethernet links may be paused and packets may not be dropped for assuring loss-less delivery.

No packet sequence numbers. L2 packets do not have sequence numbers from which RTTs, or the length of the “control loop” in terms of number of packets in flight, may be inferred.

Sources start at the line rate. Unlike the slow-start mechanism in TCP, L2 sources may start transmission at the full line rate of Ethernet Link.

Very shallow buffers. Ethernet switch buffers are typically 100s of KBytes deep, as opposed to Internet router buffers which are 100s of MBytes deep.

Small number-of-sources regime. In Ethernet (especially in Data Centers), it is the small number of sources that is typical.

Multipathing. There is more than one path for packets to go from an L2 source to an L2 destination. However, congestion levels on the different paths may be vastly different.
Performance requirements:

The congestion control algorithm should be

**Stable.** Buffer occupancy processes should not fluctuate. The algorithm needs to rapidly adapt source rates to these variations.

**Responsive.** Ethernet link bandwidth can vary with time due to traffic fluctuation in other priorities. The algorithm needs to rapidly adapt source rates to these variations.

**Fair.** When multiple flows share a link, they should obtain nearly the same share of the link’s bandwidth.

**Simple to implement.** The algorithm will be implemented entirely in hardware for fine-grained rate limit control. Therefore, it should be very simple.

The QCN Algorithm

We shall now describe the QCN algorithm [6]. The algorithm is composed of two parts:

1) **Switch or Congestion Point (CP) Dynamics:** this is the mechanism by which a switch buffer attached to an oversubscribed link samples incoming packets and generates a feedback message addressed to the source of the sampled packet. The feedback message contains information about the extent of congestion at the CP.

2) **Rate limiter or Reaction Point (RP) Dynamics:** this is the mechanism by which a rate limiter (RL) associated with a traffic fluctuation in other priorities. The algorithm needs to rapidly adapt source rates to these variations.

The CP Algorithm

Following the practice in IEEE standards, we think of the CP as an ideal output-queue switch even though actual implementations may differ. The CP buffer is shown in Fig.1. The goal of the CP is to maintain the buffer occupancy at a desired operating point, \( Q_{eq} \).

The CP computes a congestion measure \( F_b \) (defined below) and sends the \( F_b \) value to the source of the sampled packet (normally, the packet at the head of the queue at the time \( F_b \) was computed).

The higher \( F_b \) is, the more frequently incoming packets are sampled and the more feedback messages are sent to the data packets sources.

Let \( Q \) denote the instantaneous queue-size and \( Q_{old} \) denote the queue-size when the last feedback message was generated. Let \( Q_{off} = Q - Q_{eq} \) and \( Q_{delta} = Q - Q_{old} \), then \( F_b \) is given by the formula

\[
F_b = -(Q_{off} + WQ_{delta})
\]  

(1)

where \( W \) is a non-negative constant, taken to be 2 for the baseline implementation.

The interpretation is that \( F_b \) captures a combination of queue-size excess (\( Q_{off} \)) and rate excess (\( Q_{delta} \)). Indeed, \( Q_{delta} = Q - Q_{old} \) is the derivative of the queue-size and equals input rate minus output rate. Thus, when \( F_b \) is negative, either the buffers or the link or both are oversubscribed, and a feedback message containing the \( F_b \) value, quantized to 6 bits, is sent. When \( F_b \geq 0 \), there is no congestion and no feedback messages are sent. Table 1 shows the basic inter-sampling period at which a feedback message is recomputed and reflected back to the source as a function of the last computed \( F_b \) value. The actual inter-sampling period is the basic inter-sampling period uniformly jittered by +/- 15%. Jitter is introduced to increase the accuracy of sample.

![Figure 1. Congestion detection in QCN CP](image)

**The RP Algorithm**

Since the RP is not given positive rate-increase signals by the network, it needs a mechanism for increasing its sending rate on its own. Due to the absence of ACKs in Ethernet, the increases of rate need to be clocked internally at the RP. Before proceeding to explain the RP algorithm, we will need the following terminology:

- **Current Rate (CR):** The transmission rate of the RL at any time.
- **Target Rate (TR):** The sending rate of the RL just before the arrival of the last feedback message.
- **Byte Counter:** A counter at the RP for counting the number of bytes transmitted by the RL. It times rate increases by the RL. See below.
- **Timer:** A clock at the RP which is also used for timing rate increases at the RL. The main purpose of the Timer is to allow the RL to rapidly increase when its sending rate is very low and there is a lot of bandwidth becomes available. See below.

<table>
<thead>
<tr>
<th>Floor (F/8)</th>
<th>Basic inter-sampling period</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>150KB</td>
</tr>
<tr>
<td>1</td>
<td>75KB</td>
</tr>
<tr>
<td>2</td>
<td>50KB</td>
</tr>
<tr>
<td>3</td>
<td>37.5KB</td>
</tr>
<tr>
<td>4</td>
<td>30KB</td>
</tr>
<tr>
<td>5</td>
<td>25KB</td>
</tr>
<tr>
<td>6</td>
<td>21.5KB</td>
</tr>
<tr>
<td>7</td>
<td>18.5KB</td>
</tr>
</tbody>
</table>

\(^1\) It is helpful to think of \( Q_{eq} \) as roughly equal to 20% of the size of the physical buffer.
We now explain the RP algorithm assuming that only the Byte Counter is available. Later, we will explain how the Timer is integrated into the RP algorithm. Fig.2 shows the basic RP behavior.

![Figure 2. QCN RP operation](image)

**Rate decreases:** This occurs only when a feedback message is received, in which case CR and TR are updated as follows:

\[
CR = CR(1 - G_d |F_b|) \tag{2}
\]

\[
TR = CR \tag{3}
\]

where the constant \(G_d\) is chosen so that \(G_d |F_{b\max}| = 0.5\), i.e. the sending rate can be decreased by at most 50%.

**Rate increases:** This occurs in two phases: **Fast Recovery** and **Active Increase**.

**Fast Recovery (FR):** The Byte Counter is reset every time a rate decrease is applied and it enters the FR state. FR consists of 5 cycles; each cycle equal to 150KB of data transmission by the RL (jittered by +/- 15%). At the end of each cycle, TR remains unchanged while CR is updated as follows:

\[
CR = (CR + TR) / 2 \tag{4}
\]

Thus, the goal of the RP in FR is to rapidly recover the rate it lost at the last rate decrease episode.

**Active Increase (AI):** After 5 cycles of FR have completed, the RP enters the AI phase where it probes for extra bandwidth on the path. During AI, the byte counter counts out cycles of 75KB (jittered by +/- 15%). At the end of a feedback message, the RL updates TR and CR as follows:

\[
TR = TR + AI\ _{INC} \tag{5}
\]

\[
CR = (CR + TR) / 2 \tag{6}
\]

where \(AI\ _{INC}\) is a constant chosen to be 5Mbps in the base-line 10Gbps implementation. This completes the description of the basic RP algorithm using only the Byte Counter. Next we motivate and discuss the details of the Timer operation.

**Timer Motivation:** Since rate increases using the Byte Counter occur at times proportional to the current sending rate of the RL, when the CR is very small, the duration of Byte Counter cycles when measured in seconds can become unacceptably large. Since the speed of bandwidth recovery (or responsiveness) is a key performance metric, we have included the Timer in QCN.

![Figure 3. Byte Counter and Timer interaction with RL](image)

**Timer Operation:** The Timer functions similarly as the Byte Counter: it is reset when a feedback message arrives, enters FR and counts out 5 cycles of TIMER_PERIOD duration (TIMER_PERIOD is 15ms long in the 10Gbps baseline, jittered by +/- 15%), and then enter AI where each cycle period is reduced to half.

Even though the Byte Counter and Timer operate independently, they are used jointly to determine rate increases at the RL as shown in Fig.3. After a feedback message is received, they each operate independently and execute their respective cycles of FR and AI. Together, they determine the state of the RL and its rate changes as follows.

- The RL is in FR if both the Byte Counter and the Timer are in FR. In this case, when either the Byte Counter or the Timer completes a cycle, CR is updated according to (4).
- The RL is in AI if only one of the Byte Counter and the Timer is in AI. In this case, when either completes a cycle, TR and CR are updated according to (5) and (6).
- The RL is in HAI (for Hyper-Active Increase) if both the Byte Counter and the Timer are in AI. In this case, the ith time that a cycle for either the Byte Counter or the Timer is completed, TR and CR are updated as:

\[
TR = TR + iHAI\ _{INC} \tag{7}
\]

\[
CR = (CR + TR) / 2 \tag{8}
\]

where HAI INC is a constant set to 50Mbps in the 10Gbps baseline implementation. So the increments to TR (and hence of CR) in HAI occur in multiples of 50Mbps.

Thus, the Byte Counter and Timer should be viewed as providing “rate increase instances” to the RL. Their state determines the state and, hence, the amounts of rate increase at the RL.

Two other crucial features of the QCN algorithm which are useful for ensuring its reliable performance when the path bandwidth suddenly drops remain to be mentioned:

**Extra Fast Recovery:** Whenever a RP gets consecutive \(F_d\) messages (i.e. CR has not increased meanwhile), the RP does not decrease TR and does not reset the Byte Counter except for the first \(F_d\). Everything else (CR and Timer reset) functions in the same way as usual.

**Target Rate Reduction:** If TR is greater than CR by 10 times, decrease TR by a factor of 8.
3. IMPLEMENTATION

3.1 The NetFPGA Platform
NetFPGA [3, 4] is a programmable hardware platform for network teaching and research. The platform has a built-in Xilinx Virtex-II Pro FPGA (clocked at 125MHz) that performs all media access control (MAC) functions, and it attaches to the PCI bus of a Linux-based PC. The NetFPGA card has four Gigabit Ethernet ports, four ports internally connected to the host, and 4MB of SRAM.

3.2 QCN Design
Our implementation is within the User Data Path module provided by the NetFPGA common library. Next we show the QCN Switch and NIC implementation.

QCN Switch
Fig.4 shows the QCN Switch block diagram. Incoming packets are arbitrated in a round-robin fashion and forwarded to the appropriate output port/queue depending on the forwarding information set by the user. For the sake of our evaluation, we implemented the QCN Switch with one Congestion point only that receives all incoming packets regardless of their physical network port and forwards them to Output-Port0.

Before a packet is enqueued (Receive Block):
(i) Its source & destination MAC addresses are extracted. These fields are potentially needed in case a congestion message is to be generated.
(ii) The Byte Counter field is decremented by the packet size.
In the mean time, the \( F_b \) Generator module keeps track of the current and past Queue length, waiting for the Byte Counter to reset in order to:
(i) Set \( Q_{old} \) equal to the current queue length.
(ii) Compute the \( F_b \) value via the \( F_b \) Calculator module. If the computed value is negative, a congestion message is sent back to the source of the sampled packet. The structure of the congestion message is shown in Fig.5.
(iii) Re-initialize the Byte Counter value as defined in Table1 with +/- 15% jitter.
Congestion messages are then forwarded to the output MAC at the highest priority.

QCN NIC
Fig.6 shows the QCN NIC design with four Reaction Points (due to the NetFPGA logic space limitation). Packets are either forwarded from the host or internally generated at 1Gbps via the UDP Pkt Gen modules within each Reaction Point (each UDP Pkt Gen has a distinct source MAC address).

Forwarded and generated packets pass through a Token Bucket Rate Limiter. The depth of the bucket is arbitrarily set to 3KB.

Tokens of size proportional to CR (fed from the Rate Calculator module) are periodically added every 16usec.

Hence, with the smallest token size being 1byte, the minimum achievable rate of the token bucket is 0.5Mbps (i.e. 0.5Mbps rate granularity as well).

The Rate Calculator module, in its turn, keeps track of the CR, TR, Byte Counter stage, and Timer stage and is triggered by:
(i) Received congestions messages: The Receive module checks if the received packet matches the Type field of the congestion message (Fig.5), and extracts the \( F_b \) value and the destination MAC if so. The \( F_b \) value is forwarded to the appropriate RP via the \( F_b \) Demux module based on the extracted MAC address.

(ii) Byte Counter expires: The Byte Count module expires when either 150KB or 75KB (the value depending on the Byte Counting stage and uniformly jittered by +/- 15%) worth of packets are sent out from the RL.

(iii) Timer expires: The timer value is either 25msec or 12.5msec (also depending on the Timer Stage and uniformly jittered by +/- 15%).

Finally, and right before exiting the Reaction Point module, in the case of forwarded and generated packets, and before reaching the Receive module, in the case of arriving packets, we pass these packets through an SRAM-implemented circular FIFO. This module mimics links with larger delays and is denoted by the Delay Line module. It can artificially introduce per-link round-trip delay on each outgoing link independently and up to 64msec for the 4 outgoing links aggregated. The Delay Line module is by no means part of the QCN-related implementation, but is very critical for the flexibility of our experimental setup.
4. PERFORMANCE EVALUATION

4.1 Setup
We use two NetFPGAs as QCN NICs and a third one as a QCN Switch. The setup is shown in Fig. 7a with up to 8 RPs all sending their internally generated UDP traffic to the same Output Queue of size 150KB. A rate limiter is added to control the switch service rate as needed for our experiments, after which packets get dropped.

For our particular experiments, we vary the number of active RPs from 1 to 8, we vary the round-trip time (RTT) between 100us and 1000us, and we reduce the switch service rate from 0.95Gbps to 0.2Gbps for a period of 3.7sec and increase it back to 0.95Gbps afterwards as shown in Fig. 7b. Meanwhile, we report the CP queue size & the switch rate utilization results, and we compare our results with those obtained using OMNeT++ simulator [7].

Table 2 shows the QCN parameters in our evaluation. These parameters are based on IEEE standards.

<table>
<thead>
<tr>
<th>Table 2. QCN Evaluation Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>QCN NIC Parameters</td>
</tr>
<tr>
<td><strong>AI_INC</strong></td>
</tr>
<tr>
<td><strong>HAL_INC</strong></td>
</tr>
<tr>
<td><strong>G_d</strong></td>
</tr>
<tr>
<td>QCN Switch Parameters</td>
</tr>
<tr>
<td><strong>Q_cq</strong></td>
</tr>
<tr>
<td><strong>W</strong></td>
</tr>
</tbody>
</table>

4.2 Results
Figures 8 through 13 show that our implementation results closely match OMNeT++ results in all cases. Other experiments have been also run with different number of active RPs, different RTTs, and different switch service rates to further verify the implementation correctness, but the results are not reported to avoid redundancy.

5. CONCLUSION AND FUTURE WORK
We built a prototype of the QCN system on the NetFPGA platform, demonstrated the ease of implementation, and achieved the expected performance. A wide spectrum of data center topologies and network settings can now be evaluated under QCN. We are currently adding Priority-based Flow Control functions (PFC, IEEE 802.1Qbb) to our implementation.

We are also evaluating TCP’s performance with QCN and are interested in further studying the interaction with the PFC functions.

6. ACKNOWLEDGMENTS
We thank Prof. Prabhakar for his guidance and valuable feedback throughout the project. We also thank Dr. John Lockwood and Adam Covington for letting us share their NetFPGA machines earlier, Jianying Luo for helping provide the delay line and jitter components, and Berk Atikoglu for running OMNeT++ simulations.
Figure 8. Queue Size and Utilization in (a) Hardware and (b) OMNeT++, 1 source – 100u RTT

Figure 9. Queue Size and Utilization in (a) Hardware and (b) OMNeT++, 1 source – 500u RTT

Figure 10. Queue Size and Utilization in (a) Hardware and (b) OMNeT++, 1 source – 1000u RTT

Figure 11. Queue Size and Utilization in (a) Hardware and (b) OMNeT++, 8 sources – 100u RTT

Figure 12. Queue Size and Utilization in (a) Hardware and (b) OMNeT++, 8 sources – 500u RTT

Figure 13. Queue Size and Utilization in (a) Hardware and (b) OMNeT++, 8 sources – 1000u RTT

7. REFERENCES