2018 IEEE 26th International Conference on Network Protocols

DCQCN+: Taming Large-scale Incast Congestion in RDMA over Ethernet Networks

Yixiao Gao[†], Yuchen Yang[†‡], Chen Tian[†], Jiaqi Zheng[†], Bing Mao[†], Guihai Chen[†]
[†]*State Key Laboratory for Novel Software Technology, Nanjing University, China*
[‡]*Computer Science Department, Brown University, USA*

***Abstract*—Remote Direct Memory Access (RDMA) gains growing popularity in datacenter networks. The state-of-the-art congestion control scheme is DCQCN. However, DCQCN has performance problems when large-scale incast communication happens. DCQCN uses fixed period and steps for rate increase when probing for available bandwidth and this scheme is not scalable. The solution is *adaptive parameter control*: using a long period and a small increase step for large-scale incast, and a short period and a large increase step for small-scale incast. Although the idea is intuitive, there are many practical challenges. The scale of congestion is not easy to estimate while control scheme should be cautiously designed. In this paper, we propose DCQCN+ to improve performance for large-scale incast congestion in RDMA networks. DCQCN+ adapts the rate control mechanisms to different scenarios. DCQCN+ can deal with incast congestion of at least 2,000 flows both in simulation and testbed. The scale is 10 times larger than that of DCQCN in simulation and 4 times larger in testbed. DCQCN+ also has 10 times smaller latency.**

I. Introduction

Remote Direct Memory Access (RDMA) [1] gains growing popularity in datacenter networks. Residing in operating systems' kernel space, the traditional networking stack (*i.e.*, TCP/IP) has high CPU overhead and adds additional kernel processing latency. Originating in supercomputing, RDMA protocol stack is implemented in NIC hardware and can directly access a remote host's memory. RDMA thus provides high throughput, low latency and low CPU consumption simultaneously. RDMA over Converged Ethernet v2 (RoCEv2) [2] builds RDMA on lossless Ethernet, which uses Priority-based Flow Control (PFC) [3] to prevent packet loss at the link level. RoCEv2 is the dominant deployment form in datacenter networks [4].

The state-of-the-art congestion control scheme in RoCEv2 networks is DCQCN [5]. DCQCN is a rate-based congestion control. Similar to DCTCP, a switch marks the Explicit Congestion Notification (ECN) bit of passing packets if congestion happens. A receiver periodically notifies its sender via a signal packet called Congestion Notification Packet (CNP). The sender increases/decreases its sending rate accordingly.

However, DCQCN has performance problems when large-scale incast communication happens. When many servers synchronously send data to the same receiver, incast occurs [6]. This many-to-one traffic pattern is common for many data center applications such as MapReduce shuffle [7] and distributed storage (*e.g.*, Ceph). DCQCN cannot suppress an incast congestion when its number of flows exceeds hundreds (Section III). This problem leads to an extremely large queue length at the congestion point, which leads to a large packet latency. Also, it can cause congestion spreading via long-lasting PFC storms [4], which in turn can cause problems such as victim flows and traffic collapse.

DCQCN uses fixed period and step for rate increase when probing for available bandwidth and this scheme is not scalable. Fixed rate recovery period and fixed increase steps (*i.e.*, $55\mu s$ and 40Mbps by default in DCQCN [5]) perform well for small-scale incast congestions. As the number of incast flow grows, the rate of a single flow becomes too small to send even one packet in a single rate recovery period. Thus, a flow cannot get a rate cut CNP signal before it increases rate. As a result, the summed-up rate at the congested point is always larger than the expected convergence rate. Congestion cannot be alleviated in this case. Note that simply tuning parameters for large-scale incast congestion would hurt performance of small-scale incast scenarios. When using a large enough period and a small increase step, rate recovery is accordingly much slower. This causes throughput loss which is also unfavorable.

Our key insight is that: senders should be aware of the scale of each incast, so that they can adjust their aggressiveness accordingly. There already exist end-2-end signal packets (*e.g.*, CNPs in DCQCN). An incast receiver can exploit them to explicitly relay the scale information to senders. Concretely, we can use a reserved field of CNP packets. Thus, senders can estimate the scale with the information, then update their congestion control parameters properly.

The challenges come from different aspects. Firstly, estimate for congestion scale is not that simple. Generally, flows that traverse the same congested switch port can have different senders and receivers. For deployment in datacenters, we should only use commodity switches with limited functions. This means that we must infer the scale of congestion on endpoints. Meanwhile, for minimal modification to the original protocol, the measure for congestion scale should be simple. Secondly, there should be additional modification to the protocol so that it can keep working well when parameters are adaptively changed. Since the scale is large, congestion is hard to drain off. On the other hand, a design with excessive suppression on flows can easily cause throughput loss. Adaptive parameters may not work well without proper compensation on the design.

In this paper, we propose DCQCN+ to improve performance for large-scale incast congestion in RDMA networks.

978-1-5386-6043-0/18/$31.00 ©2018 IEEE
DOI 10.1109/ICNP.2018.00021

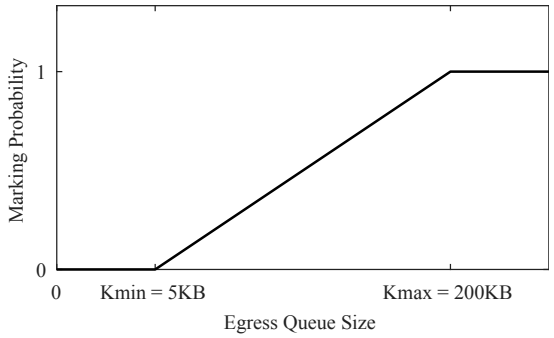


Fig. 1. ECN Marking Probability

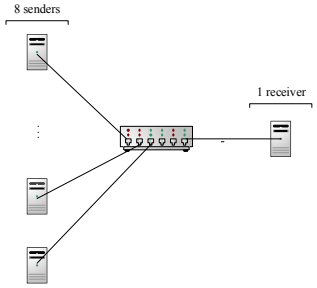


Fig. 2. Incast Topology

DCQCN+ adapts the rate control mechanisms to different scenarios. For example, it adaptively allocates CNP resource under the constraint of a NIC's hardware capability. We build a simulator for DCQCN+ on NS-3 [8] for performance evaluation. Besides, we conduct approximation experiments by adjusting parameters of Mellanox ConnectX-4 NICs to verify our design points. DCQCN+ can deal with incast congestion of at least 2,000 flows both in simulation and testbed. The scale is 10 times larger than that of DCQCN in simulation and 4 times larger in testbed. DCQCN+ also has 10 times smaller latency.

II. Background

A. RDMA

RDMA is a network feature that allows user-space applications to directly read or write remote memory without kernel interference or memory copying. The kernel bypass characteristic of RDMA provides high bandwidth and low latency for datacenter applications. So it is preferred in datacenter networks.

RDMA is designed to be lossless, which means there must be no packet loss due to buffer overflow at switches. In RoCEv2, this is ensured by PFC in L2. PFC is a simple traffic control scheme that pauses and rewakes the upstream port according to buffer occupation status. PFC can prevent packet loss, but incurs many other problems such as unfairness, victim flows[5], and deadlock[4]. Besides, congestion cannot be drained off only by PFC. We do need flow-level rate control at the sources on endpoints. Thus, DCQCN is designed and widely deployed in RDMA networks.

B. DCQCN

DCQCN requires both NICs on endpoints and switches to participate in. ECN is configured on switches to find out congestion. Notification and reaction parts are designed on NICs.

1) ECN Configuration: At the egress queues of switches, packets are marked on ECN bits randomly according to queue length as Fig. 1 shows. RED is used for randomly marking. When ingress rate is larger than egress rate, the buffer accumulates. As the queue length exceeds the minimal threshold, packets are marked on ECN bits to indicate congestion.

To obtain low queueing latency, DCQCN maintains a low buffer occupation on switches. Each packet of 1KB size in the buffer adds 0.8us on queuing time at 10Gbps links. The maximal ECN threshold (200KB) means $160\mu s$ ($40\mu s$) delay for 10Gbps (40Gbps) links, which is acceptable as a maximal value. The average queue length is smaller[5].

2) Flow rate reduction: The receiver generates and sends a CNP to the sender only if (1) the received packet is marked on ECN bits and (2) the flow has not been notified for a fixed period. This period, called interval between CNPs N, is a static parameter that needed to be configured ahead of time. [5] chooses the value $50\mu s$ considering the CNP generating power of NICs and the number of packets with common MTU (1,000B) can be received during this time. Actually, another consideration of this design is to *wait and observe*. Even though the rate of flows has already been cut, the queue length needs time to decrease. Besides, the interval also includes time for CNP transport.

Senders cut the current rate R_C and the target rate R_T as follows:

$$R_T = R_C, \tag{1}$$

$$R_C = R_C(1 - \frac{\alpha}{2}), \tag{2}$$

$$R_C = max\left\{R_C, R_{min}\right\}, \tag{3}$$

$$\alpha = (1 - g)\alpha + g, \tag{4}$$

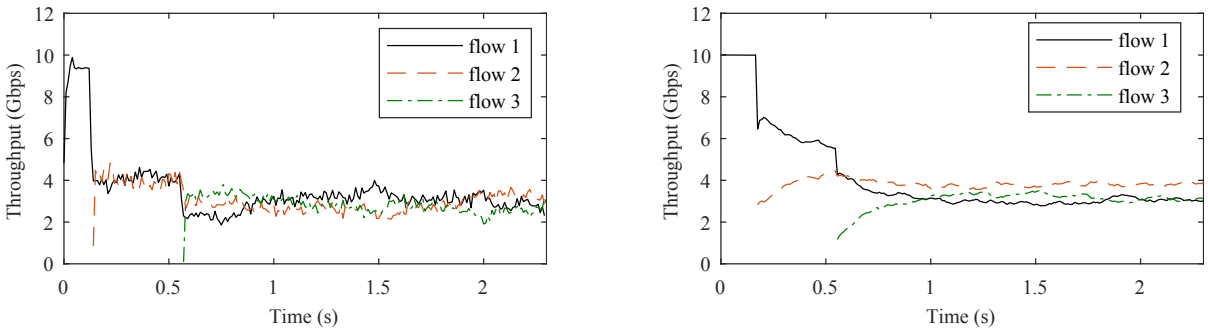
here α denotes the reduction factor, g is a pre-configured constant, and R_{min} means the minimum rate of a flow. Rate cut is trivial. If CNPs are received for 2 or more periods, α will increase and rate cut ratio will be larger at the next time.

3) Flow rate recovery: Receivers maintain a time counter and a byte counter and corresponding state bits for each flow. There is also a state counter to sign the state of increase. The time state and byte state are set to 0 after rate cut. When a flow has not received CNPs for time $K = 55us$, its time state increases. When a flow has already sent B bytes without receiving a CNP, its byte state increases. When one of the two states is not 0, the flow starts recovery in a bisecting way for $F = 5$ rounds called *Fast Recovery* (*FR*):

$$R_C = \frac{R_T + R_C}{2}. \tag{5}$$

TABLE I
PARAMETERS CONTRAST IN DCQCN AND CX4

CX4 Parameter	CX4 value	DCQCN Parameters	DCQCN value
rpg_time_reset	300us	Timer (K)	55us
rpg_byte_reset	2MB	Byte Counter (B)	10MB
rpg_ai_rate	5Mbps	Additive Increase Step (R_{AI})	40Mbps
rpg_hai_rate	40Mbps	Hyper Increase Step (R_{HAI})	100Mbps
$min\_time\_between\_cnps$	0us	CNP interval (N)	50us
$rate\_reduce\_monitor\_period$	4us	-	-



(a) DCQCN CX4 implementation

(b) DCQCN NS-3 simulation with CX4 parameters

Fig. 3. The difference between DCQCN and CX implement

The start of a new round is the increase of the states. After that, *Additive Increase* (*AI*) and *Hyper Increase* (*HI*) can be triggered. In *Additive Increase*, rate increases by fixed step R_{AI}:

$$R_T = R_T + R_{AI}, \tag{6}$$

$$R_C = \frac{R_T + R_C}{2}. \tag{7}$$

Hyper Increase is triggered when both the states exceed F. In *Hyper Increase*, the rate shows an exponential growth with parameter R_{HAI}:

$$R_T = R_T + iR_{HAI}, \tag{8}$$

$$R_C = \frac{R_T + R_C}{2} \tag{9}$$

where $i = \min\{time\_state, byte\_state\} - F$.

At each recovery, α will also decrease:

$$\alpha = (1 - g)\alpha. \tag{10}$$

4) Line-rate strategy: In DCQCN, flows start at line rate to get full utilization of links. With line-rate strategy, small flows obtain very high throughput without additional pre-communication.

However, line-rate is so aggressive that, as the number of congested flow grows, the trivial-way rate cut needs more periods to take effect. During this time, the queue length is large enough to trigger PFC. If the flow rate can be finally pressed down, then the buffer will drain. But if the flow rate is not suppressed appropriately, pause caused by PFC will last until the end of the congested flows, which is a big hurt to performance.

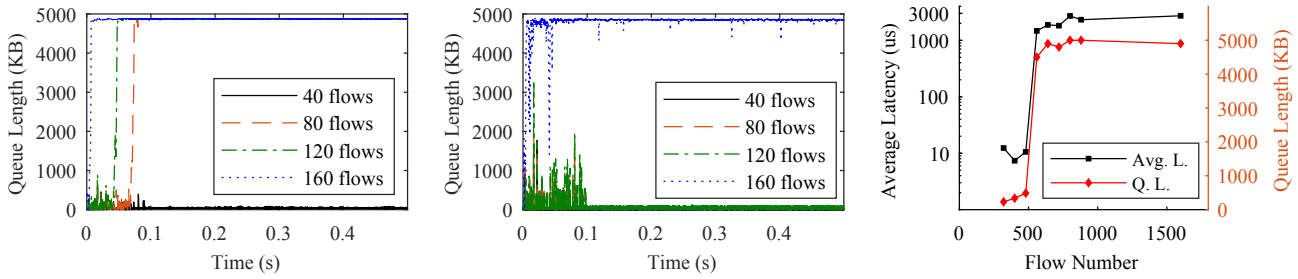
III. PROBLEMS OF DCQCN

In this section, we illustrate the problems of DCQCN that we discovered in our simulation and testbed experiments. Firstly, we show the differences between the configurations of DCQCN and CX4 implementation. The designs are also different. To deal with this, we have to conduct our experiments in both simulation and testbed. Secondly, we demonstrate and analyze the incast problem of DCQCN. With these phenomena and the analysis of their reasons, we can see the inspiration for the new congestion control scheme.

A. Experiment Setup

1) Simulation: The DCQCN simulation[5][9] on NS-3 has been released by Yibo Zhu, the designer of DCQCN. We check the design and configuration according to [5].

2) Testbed: We use 9 hosts with Mellanox ConnectX-4 NICs (CX4 in short) and 1 Mellanox SN-2700 switch in our testbed. A topology of 8:1 incast as is used, as shown in Fig. 2. For large-scale congestion, same number of flows start on each sender. In testbed experiments, we use libpcap[10] to capture packets for throughput statistics. To trigger traps in libpcap, we must turn on the sniffer on NICs. However, this functionality is a heavy load on NICs. With sniffer on, we can only capture throughput of about 18Gbps. So libpcap is only used in the experiments in Fig. 3 where links are restricted to 10Gbps. This is necessary since we have to clearly observe how DCQCN and CX4 work to know the differences. In other testbed experiments, we turn off the sniffer to prevent performance loss. We use the tool ib_send_lat in *Perftest package*[11] to test latency of flows. The maximal buffer in usage can be known from the counters on the switches.



(a) NS3, 10Gbps links. (b) NS3, 40Gbps links. (c) Incast tests on CX4, 10Gbps links.

Fig. 4. DCQCN fails to drain off congestion both in NS3 simulation and CX4 testbed.

These measurements cost very little thus are not harmful to performance.

B. Difference Between DCQCN and CX4 Implementation

We have observed that there are some differences between DCQCN in [5] and Mellanox CX4 implementation. However, the proprietary algorithm in the firmware of CX4 keeps updating. Besides, the recommended parameters are adjusted based on their real deployment environments. Thus, only by comparison of parameters and results of experiments can we learn something about these differences.

The setting of CX4 parameters is different at many points from DCQCN in [5]. Table I shows some parameters in contrast. Note that $min\_time\_between\_cnps$ (CNP interval in DCQCN) is set to 0 by default in CX4 implementation. This seems to be too small. However, by measuring the interval between CNPs,we have verified that the CNP interval of each flow is exactly controlled by the parameter $min\_time\_between\_cnps$. It means that NIC will send CNP for each ECN-marked packet that it receives. Using such low CNP interval, when congestion happens, consecutive packets are marked and this cause excessive rate cut. To prevent this, CX4 uses a new parameter $rate\_reduce\_monitor\_period$ ($4\mu s$ by default), which represents the minimum time interval between two rate reductions.

We noticed that with such parameters, rate should increase much less and slower, which means longer convergence time after rate cut. However, the result is not that clear since the difference between results of simulation and testbed is not negligible. We suspect that there are still other adjustments between them. To show this, we configured our simulation with CX4 parameters and conducted a simple experiment: 4 hosts are connected to one switch with 10Gbps links, 3 senders starts 1 flow each to the only receiver. Fig 3 shows how the rates of flows alter with respect to time in CX4 and simulation. In CX4 implementation, flows converge much faster than in simulation with the same parameters.

It is hard to say how the firmware works in the NIC, so it is necessary to discuss in simulation and CX4 implementation separately. Though with different incast scales, the failure for congestion occurs both in the simulation and testbed for CX4. Besides, we verify our explanation and design both in simulation and CX4 implementation. Thus, our result is both suitable for the initial DCQCN and CX4 implementation.

C. The Failure of DCQCN for Large-scale Incast

We have observed that both in simulation and testbed experiments, switch buffer queue length at congestion point maintains a very large value when large-scale incast happens. This shows a failure for convergence of DCQCN. For experiments, we use an incast topology of 9 hosts and 1 switch as Fig. 2 shows both in simulation and testbed. 8 senders start flows of the same number and keep sending until the end of experiments. In simulation, all flows are started at a uniform random value of $(0, 0.1)$.

Fig. 4 shows the relation between buffer queue length and number of flows in NS-3[8] simulation. In simulation default parameters of DCQCN in [5] is adopted. For 10Gbps links, we try another experiment with the parameters of rate increase (*e.g.* R_{AI}) $1/4$ of that on 40Gbps links. As the figure shows, under incast of about 80 (160) flows for 10Gbps (40Gbps) links, DCQCN works poorly and congestion cannot be alleviated.

Fig. 4(c) shows the relation of average latency, maximal buffer size and number of flows in CX4 implementation. Here average latency and maximal buffer are counted from 5 seconds after flows start. 5 seconds is a long enough time for flows to reduce rate and eliminate congestion as it is thousands of a single period. As we can see, under an incast of about 480-560 flows, CX4 can no longer deal with the congestion. The maximal buffer usage is more than 4MB, and large buffer queue causes extremely long average latency. Average latency of about 2,000μs means 2.5MB buffer on average. PFC has to work to prevent packet loss all the time. The scale is larger than the result of simulation since the parameter difference in DCQCN and CX4 implementation that we have discussed.

We have verified that when using the parameters of CX4 in simulation, the largest scale is also 480-560 flows. However, in the following discussion, we will see that simply using different parameters is not a suitable solution for large-scale incast.

D. Why DCQCN Fails for Large-scale Incast

We can summarize the failure for large-scale congestion into 3 possible reasons: insufficient CNP supply, fixed-step increase, and fixed recovery timer.

Insufficient CNP supply can only happen in the implementation. We test the maximal CNP generation rate of the NIC by keeping $min\_time\_between\_cnps$ $0\mu s$, and configure wrong VLAN tags such that the CNP can be captured by libpcap rather than used and absorbed by the NICs. The CNP generation rate of CX4 is about 1 packet per microsecond. This is sufficient for 10Gbps links as packets of 1KB size arrive every 1.25 μs. But for 40Gbps and 100Gbps links, this is far from enough. If some congested flows cannot receive CNPs in time, rate recovery will continue. It means that flows are unaware of congestion sometimes, which may break convergence. We can regard this a performance requirement for NICs rather than a design problem, but we should see the price to generate and transport so much CNPs at the same time.

Fixed-step increase is a simple design that is easy to implement. However, it is not robust for large-scale congestion. Total increase step of throughput is proportional to the number of recovering flows. For an extreme example, if there are 1,000 flows in *AI* with $R_{AI} = 40Mbps$, the total rate increase step can be as large as 20Gbps, i.e. half of the line rate on 40Gbps links. This is too large a step to converge.

The last but most important reason is that the fixed recovery timer is too short. In DCQCN, the recovery timer is set to $55\mu s$. For example, it takes $100\mu s$ to send one packet of 1KB if flow rate is 80Mbps. This time is almost twice of the timer. Then the timer is certain to exceed and flow rate increases. In other words, with this timer flow rate is almost always larger than 80Mbps. Besides, the minimum rate should be much smaller than the convergence rate to drain off congestion. 80Mbps is the convergence rate of only 500 flows for 40Gbps links (125 for 10Gbps). This is a simple upper bound for the scale of congestion when using DCQCN. Although recovery timer has been enlarged in the CX4 implementation, it is still not enough for thousands of flows.

One may think using larger recovery time or smaller increase step to deal with congestion. However, this is harmful to throughput for small incast. Fig. 5 shows the total throughput of 8 flows when using different parameters. We can see clearly that the recovery speed is dominated by the timer and increase step. Unfortunately, total throughput recovery is also related to number of flows since increase step is fixed for a single flow. Thus, parameters that work for large incasts cause an unbearable throughput loss for small incasts. The parameters are fixed, but the number of active flows is varying as flows start and end.

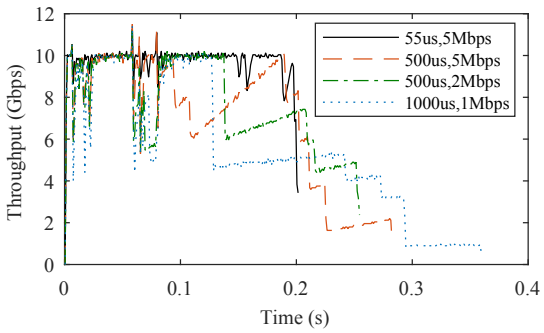


Fig. 5. Parameters (Recovery Timer (K), Additive Increase Step (R_{AI})) for large flows cause throughput loss at small incast in DCQCN.

IV. Design

A. Design Intuition and Overview

To deal with the incast problem, DCQCN+ uses *dynamic parameters scheme*. For adapting parameters, senders must know about the incast scale. In DCQCN+, we use two kinds of information. The first is the CNP period carried in CNPs. In our design, all congested flows with the same receiver share the CNPs equally by time multiplexing. Thus, CNP period can reflect the number of congested flows on the receiver. We utilize an available field in CNPs to carry the CNP period without using additional signal packets. The second thing to learn about congestion is the flow rate itself. Generally, flows of larger rate have more packets in the congested queue, thus with a larger possibility packets it will receive a CNP and reduce rate. So when converged, all congested flows have almost same rate, which is related to the flow number.

How to choose parameters according to the CNP period and the flow rate is another question. There are two design points. (1) The recovery timer should always be larger than the CNP period to receive a CNP. Besides, it should be large enough for the rate to send at least one packet. Basically we can set the maximum of them as the recovery timer. In addition, we use a relaxation ratio λ ($\lambda > 1$). (2) The total throughput increase should not increase as the number of congested flows grows. Thus, increase step R_{AI} should be proportional to flow rate.

However, with such parameters flow recovery will be pretty slow after congestion. So we use an additional exponential growth phase *HI*. Though has the same name with an increase phase in DCQCN, this is a little different from the design DCQCN as this exponential increase is related to the flow rate rather than to a constant. Besides, it is easier for small flows to trigger *HI* in DCQCN+ than in DCQCN. In *HI*, a small flow can increase 1,000 times only in 10 iterations.

Generally speaking, DCQCN+ is not very sensitive to its parameters based on such designs. *Dynamic parameters scheme* makes DCQCN+ less sensitive compared to the *static* DCQCN. We will discuss into more details in section VI.

Just like DCQCN, DCQCN+ can be divided into 3 logical parts: Congestion Point (CP), Notification Point (NP) and Reaction Point (RP). We discuss each part separately.

B. Design Details

1) CP Algorithm: DCQCN+ uses the same CP algorithm as DCQCN does since a friendly-deployed solution always requires little from the switches. Switches are configured RED for ECN marking like Fig. 1, but the minimal and maximal marking threshold is 20KB and 200KB. Here we use a bigger

minimal marking threshold to guarantee a higher bandwidth. 5KB, *i.e.* 5 packets, is a little shallow for real environment and large-scale scenarios.

PFC is necessary to prevent packet loss. Our PFC configuration is the same as DCQCN in [5]. This configuration has been calculated and verified when DCQCN is designed.

2) NP Algorithm: As we mentioned, CNP plays an important role in congestion control. With insufficient CNP supply, it is possible that the flow rate will never converge. DCQCN supposes that the NICs can generate CNPs as it wants. Differently, the CNP generation power of NICs is taken into consideration at NP in our design.

CNP interval of each flow is required to be rather fixed for setting recovery timers. To achieve this, NP maintains a list of all congested flows, and traverses the list to send CNPs for the flows that have received ECN-marked packets. A record in the list contains the flow identification, the ECN bit and a time field. If an ECN-marked packet is received, NP will set the ECN bit of the record to 1. NP checks one record at a time to decide whether to send CNP for this flow. CNP will be sent if the ECN bit is set to 1 and the flow has not been sent CNP to for a while (the minimal CNP interval, $45\mu s$ by default). The CNP period τ that is written into CNP to notify RP denotes the time span of checking the same flow record, so can be calculated by CNP generation interval δ and list length l as follows:

$$\tau = l * \delta. \tag{11}$$

Once NP sends a CNP, it resets the ECN bit and the time that it sent this CNP in the list. We use the Reserve field [2] of CNP packet to explicitly carry the CNP period to the sender. Thus, the sender can set the timer according to this value. This field is 16 Bytes and is not used by now.

3) RP Algorithm: When RP receives a CNP, it reduces the target rate and current rate R_T, R_C of the flow just like DCQCN:

$$R_T = R_C, \tag{12}$$

$$R_C = R_C(1 - \frac{\alpha}{2}), \tag{13}$$

$$\alpha = (1 - g)\alpha + g. \tag{14}$$

Besides, there is a minimal rate protection R_{min}:

$$R_C = max\left\{R_C, R_{min}\right\}. \tag{15}$$

Note that a minimum rate gives an upper bound of flow numbers for incast, we use a small value $R_{min} = R_C/10000$ at the bound of 10,000 flows.

There are timers of each flow for rate recovery and update of α. When a timer expires, the corresponding update is processed. RP gets the CNP interval from the PSN field of the CNP and resets the timer according to it. The timer must be long enough for the flow to send several packets and no shorter than the CNP period. Thus if the CNP interval τ is larger than 50us, the timer of α update K_α and the timer of rate increase K will update as follows:

$$K_\alpha = \lambda_\alpha \max\left\{\tau, \frac{M}{R_C}\right\}, \tag{16}$$

$$K = \lambda \max\left\{\tau, \frac{M}{R_C}\right\}, \tag{17}$$

where M denotes the maximal MTU of packets and λ, λ_α are the enlarging ratios. In our experiments, M is set to 1000KB, λ_α is set to 1 and λ is set to 2.

A state counter S is used to signify different phases of rate recovery. When the rate recovery timer expires, the state increases 1. If the state is less than the threshold $F = 5$, the flow rate increases as *FR* in DCQCN:

$$R_C = \frac{R_T + R_C}{2}. \tag{18}$$

If the state is more than F but less than $4F$, a different *AI* increase is used:

$$R_T = R_T + \begin{cases} \min\left\{\frac{1}{5}R_C, \frac{1}{50}R_l\right\}, & if\ \alpha > 0.1, \\ \min\left\{\frac{1}{10}R_C, \frac{1}{100}R_l\right\}, & otherwise, \end{cases} \tag{19}$$

and current rate increase as the same way:

$$R_C = \frac{R_T + R_C}{2}, \tag{20}$$

here R_l denotes the link capacity. Note that we use a different increase step when α is different. When α is large, bigger step is used for a rapid recovery. Small α means the flow rate is close to convergence, so smaller step should be set. The increase step R_{AI} is calculated in the form of a ratio of the current rate, thus total throughput increase of the congested point is only related to the total throughput now, and won't increase as the number of flows grows. Besides, we use a ratio of R_l to bound the increase step for small incast cases, this is necessary according to our experiments.

HI increase is used if the state is more than $4F$:

$$R_T = R_T + \min\left\{R_C, \frac{S - 4F}{100}R_l\right\}, \tag{21}$$

$$R_C = \frac{R_T + R_C}{2}, \tag{22}$$

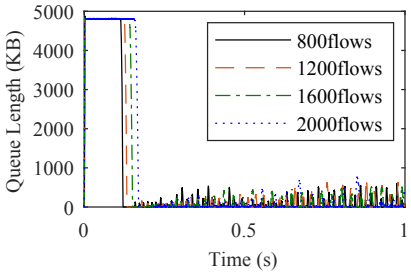
where S denotes the state counter.

HI is designed as a rapid growth to ensure high bandwidth. It only takes 10 periods for small flow rate to grow 1,000 times in *HI* phase. The increase is also bounded by another exponential increase step with no respect to flow rate to protect for the cases of large flows.

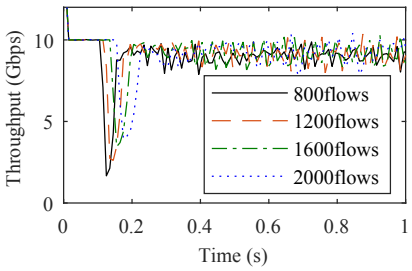
When the timer for α update is out, we update α as follows:

$$\alpha = (1 - g)\alpha. \tag{23}$$

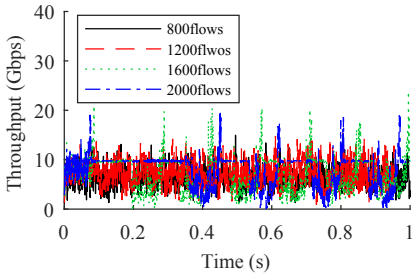
We do not use byte counter in our design since the rate increase will be ambiguous under different timers. Actually, in DCQCN, it takes $2ms$ to send 10MB data for increasing the state even at 40Gbps rate. This is 40 times of the recovery timer, and is even larger when using 10Gbps links. Thus this design with this configuration has little influence for the recovery.



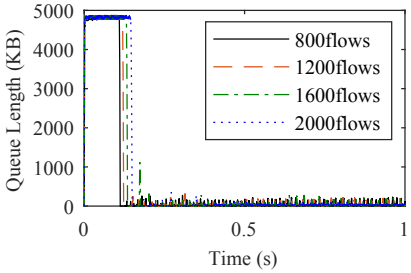
6.(a) Queue length, 10Gbps link.



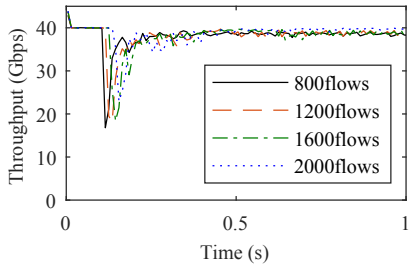
6.(b) Throughput, 10Gbps link.



7.(a) Throughput of TIMELY, 10Gbps links.

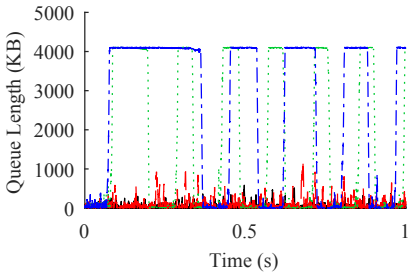


6.(c) Queue length, 40Gbps link.



6.(d) Throughput, 40Gbps link.

Fig. 6. DCQCN+ at large incast.



7.(b) Queue length of TIMELY, 10Gbps links.

Fig. 7. TIMELY at large incast.

V. Evaluation

A. Large-scale Incast

The initial design goal of DCQCN+ is to deal with the large incast problem. To verify our design, we test the performance of DCQCN+ under large incast in our simulation. We use the incast topology of 8 senders and 1 receiver as Fig. 2 shows. Each sender starts same number of flows. To ensure incast, all flows are started at a uniformly random time in 0.1 seconds. Fig. 6(a)(b) shows variation of the queue length of the congestion point as flow number grows. As we can see, DCQCN+ can handle the congestion of at least 2,000 flows both at 10Gbps and 40Gbps links. The queue length of the congested point is bounded by about 200KB when the flows are converged. In the worst case, the switch buffer can be pressed down in about 0.1 seconds after the time all flows are started. Recall that in Fig. 4, DCQCN cannot drain off congestion of up to 160 flows in the same scenario. When using DCQCN, the queue length is the maximal value that is prevented by PFC, i.e. 4.9MB. DCQCN+ has 20× smaller queue length, which means 20× smaller queuing delay. Besides, it will not suffer from the side effects caused by PFC storms while DCQCN will.

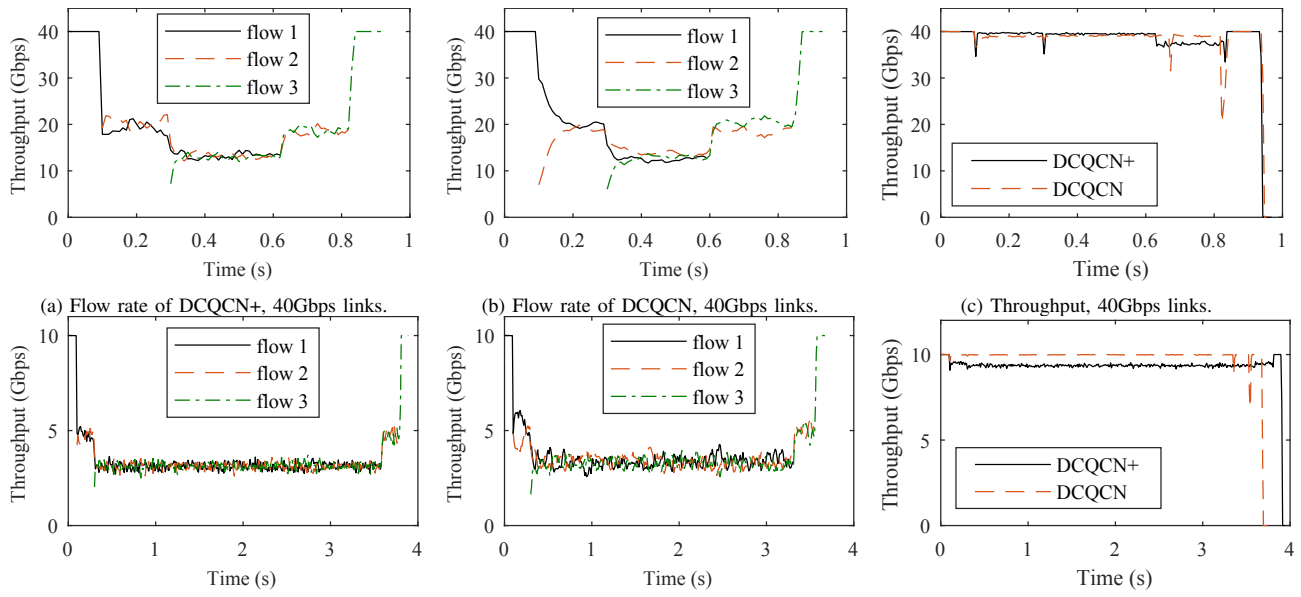
It is notable that 2,000 is not the limit of this congestion control scheme. However, we think that it is not practical to simulate under larger scale of incast since other parts (*e.g.*, queue pairs of NICs) could be the bottleneck of the system. However, there is a bound for the flow number $10,000$ given by R_{min}. The bound is not tight since flow rate must be lower than the convergence rate at some time to drain off the congestion. If we want to deal with a larger scale, we can just use smaller R_{min}.

TIMELY[12] is another congestion control scheme for RoCEv2 that is based on RTT. [13] compares DCQCN and TIMELY on different aspects of performance, and claims that ECN is a better congestion signal compared to delay. We use the TIMELY simulation[13] on NS3 to learn how TIMELY deals with large-scale congestion for comparison. Fig. 7 shows the performance of TIMELY under large-scale congestion. Much better than DCQCN, TIMELY can still limit the queue length of the congestion point at the scale of 1200 flows. However, TIMELY still fails for larger scale for the lack of specialized design. Besides, the throughput of flows is very unsteady and a little low when the flow number is large.

To deal with large congestion, our design sacrifices a small ratio of throughput. Throughput loss mainly happens in the short recovery time just after the end of the congestion. Fig. 6(c)(d) shows the relation between total throughput and time as flow number grows. Note that when we talk about throughput, we refer to *total sending rate* calculated by counting the packet size sent in a fixed time span at all senders. This value is more interesting since we can see how the total rate changes when DCQCN+ works. The throughput of receivers is shaped by the switch buffer, thus more steady. There is no packet loss under the protection of PFC, so all the throughput of senders is effective. We can see that throughput loss is always less than 0.1 second, which is an acceptable price. Convergence throughput is always more than 90% of the link.

B. Small-scale Incast

It is also important to concern the convergence and performance of DCQCN+ in small-scale incast scenarios. We



(a) Flow rate of DCQCN+, 40Gbps links.
(b) Flow rate of DCQCN, 40Gbps links.
(c) Throughput, 40Gbps links.
(d) Flow rate of DCQCN+, 10Gbps links.
(e) Flow rate of DCQCN, 10Gbps links.
(f) Throughput, 10Gbps links.

Fig. 8. Flow rate convergence at small incast.

conducted a 3:1 incast simulation using the same incast topology to verify this. 3 flows start at 0s, 0.1s and 0.3s thus we can see how the rate of the flows increase and recover. Fig. 8(a)(b)(d)(e) show the flow rate variation on 10Gbps and 40Gbps links. Flows in DCQCN and DCQCN+ converges similarly. Rates of flows are cut after a very short time, and converge with small variation. DCQCN+ has a little smaller variance than DCQCN on 10Gbps links.

Fig. 8(c)(f) shows the total throughput. DCQCN+ has similar throughput and flow completion time compared to DCQCN on 40Gbps links, but has lower throughput (about 4%) on 10Gbps links. This loss can be compensated by more meticulous adaption on design or tuning on parameters, especially on R_{AI}. However, this is an acceptable loss.

C. Large Topology

In large-scale incast tests, we use different numbers of flows on 8 hosts to get large incast. However, today's Datacenter network has hundreds or thousands of hosts using CLOS[14] topology. To be practical, we should simulate in such environments. We build a CLOS topology of 2410 nodes similar to [14] and test incast scenarios for DCQCN+ in such topology. But the number of core switches is 10 rather than 100 to get a contraction ratio of 0.1 for core congestion.

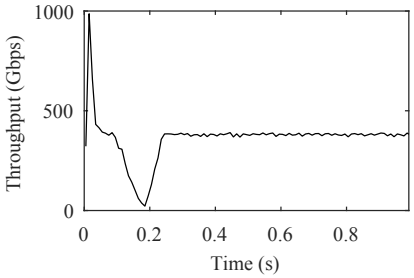
1900 senders in 19 racks send to 10 receivers in another rack. All flows are started randomly in 0.1 seconds. Fig. 9 shows total throughput and one example of buffer variation of switches. Buffer can be cut down to less than 200KB as we expected. The total throughput is converged to 400Gbps, which is equal to total throughput of 10 receivers. Note that our throughput denotes the total throughput of all the senders, thus throughput is larger than 400Gbps at the beginning, and is low for a short time to drain off the buffer queue. Average throughput when converged is about 95%.

D. Testbed Approximation

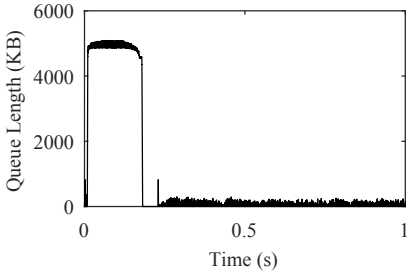
To verify our design, we figure out a way to mimic DCQCN+ using CX4 NICs in our testbed. The key point of design of DCQCN+ is to use different parameters for different scenarios. With similar parameters, DCQCN+ and DCQCN work similarly. Thus, we can use DCQCN with the static parameters of DCQCN+ at the converge point to mimic DCQCN+. For example, for 540 flows incast, we configure CX4 with $K = 2 * \max\left\{540 * 1us, 8000bits/(\frac{10Gbps}{540})\right\}$ and $R_{AI} = \frac{1}{10} \times \frac{10Gbps}{540}$. Since DCQCN+ adapts its parameter according to the flow rate and number of flows, at the beginning and the end of the traffic DCQCN+ has smaller CNP period and larger R_{AI}. But these parameters are close to the parameters that DCQCN+ has when converged in the same scenario. The time for DCQCN to reach convergence is much slower in this case, but they have similar convergence states. Fig. 10 shows the result of such experiments in our testbed. We use the incast topology of Fig. 2 on 10Gbps links. Here we choose some flow numbers near the threshold that CX4 fails on and a larger one to show scalability. We can see that the approximation, i.e. CX4 with static DCQCN+ parameters, does work for large incast in real environments. Congestion drains and the queue length is very short. Using these parameters gains 10× smaller buffer queue length and latency in this scenario.

E. Realistic Workload

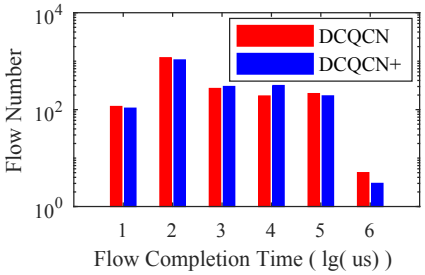
To test the performance of DCQCN+ under realistic workloads, we conducted experiments in NS3. The incast topology



9.(a) Total throughput of 1900 hosts.

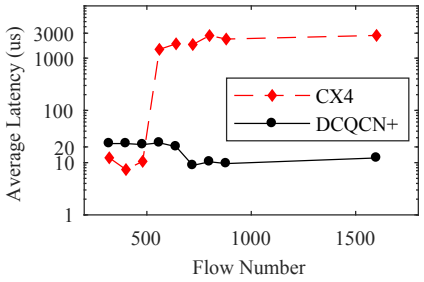


9.(b) Queue length of core switch.

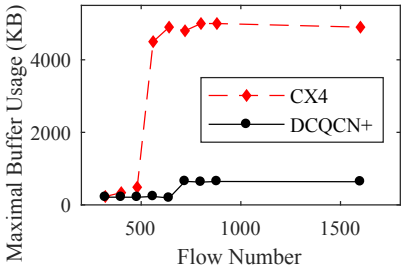


11.(a) FCT, Data Mining workload.

Fig. 9. 1900:10 incast in CLOS topology, DCQCN+.

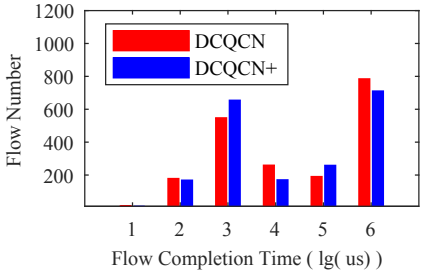


10.(a) Average latency.



10.(b) Maximal buffer usage.

Fig. 10. Using parameters of DCQCN+ in CX4 for approximation.



11.(b) FCT, Web Search workload.

Fig. 11. Realistic Workload (10Gbps, load = 0.8).

is still used, but this time all hosts can be both the sender and the receiver. We generate flows subjected to Poisson Process according to the load and the distribution of flow sizes in workloads. Fig. 11 shows the number of flows with different Flow Completion Time (FCT). FCT is measured with logarithm scale from $1\mu s$ to $10^6\mu s$. Compared to DCQCN, DCQCN+ has only slightly smaller flow completion time with similar distribution. This is not unexpected because incast scenario does not exsit in such realistic workload models. The maximal number of flow at one sender or receiver is about 200, and these flows are not running at the same time. Thus, the incast degree is even much smaller. In such cases, the performance of DCQCN+ is similar to that of DCQCN with limited improvement.

VI. DISCUSSION

A. Convergence and Adaptivity

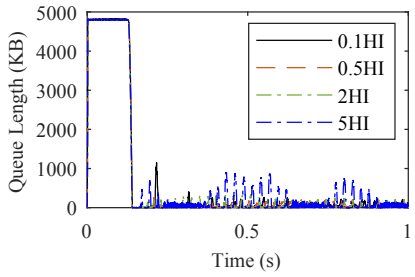
DCQCN+ adjusts values of timers by calculations using the information that the CNP carries, so it is adaptive to both heavy and light traffic. For convergence, we can make sure that at least one packet is sent during the interval of increases, and one CNP can be sent in the next period if RP receives an ECN-marked packet. This guarantee is irrelevant with traffic, flow numbers or CNP generation capability. Thus, in most scenarios a rate cut happens always before an increase if needed. Besides, the influence of missing rate reduction with small probability is small even the number of flows is large. Therefore, we believe this design is very strong for most cases. Our simulation has shown its power under 2,000 flows.

The strict proof of the convergence of DCQCN+ is left to future work.

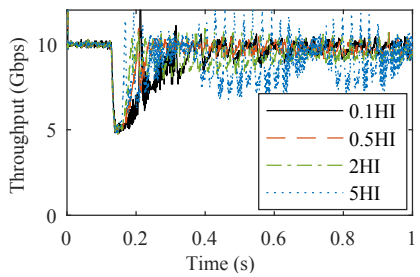
B. Parameter Tuning

Setting parameters is always a tradeoff among different performance indexes. For an example, to get a higher throughput, we considered using a lower rate cut ratio in equation (13). This value, i.e. the coefficient of α, is $1/2$ by default. Fig. 12(c)(d) show the buffer queue length and throughput using different maximal rate cut ratio. We use 720 flows at 8:1 incast topology of 10Gbps links, all flows start randomly in 0.1 seconds. Under higher maximal cut ratio, DCQCN+ can drain off the congestion more quickly, but have a bigger throughput loss at the same time. Choice can be made according to requirements, but we suggest using $1/2$ and our other simulation results are based on this value. However, such differences are very limited and not essential.

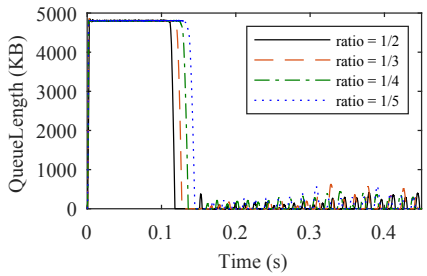
Actually, DCQCN+ is not very sensitive to such parameter tuning. Fig. 12(a)(b) show the performance of DCQCN+ when using different times of *HI* parameters. Here we use incast topology in Fig. 2. The incast scale is 1200:1 and the speed of links is 10Gbps. DCQCN+ works with high performance unless the parameters are very extreme ($5\times$ *HI*). $5\times$ means increasing 1Gbps or doubling the rate for a flow when increasing in *HI*. We believe that it is the pattern of dynamic rather than the parameters that matter. For further discussion, we plan to study the influence of parameters based on mathematical analysis when proving the stability of DCQCN+ in future work.



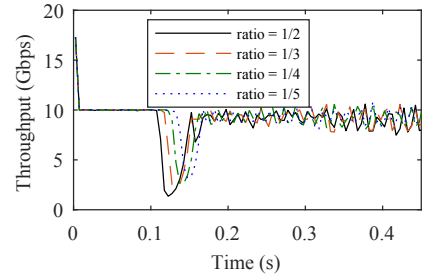
12.(a) Queue length, 10Gbps links.



12.(b) Throughput, 10Gbps links.

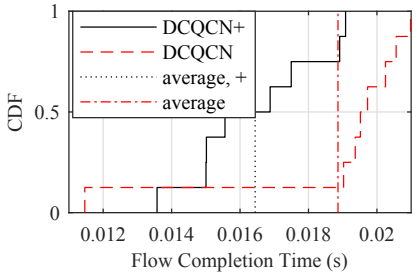


12.(c) Queue length, 10Gbps links.

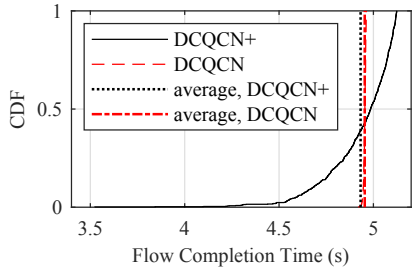


12.(d) Throughput, 10Gbps links.

Fig. 12. Insensitiveness of DCQCN+ parameters.



13.(a) Completion time CDF of 8 flows.



13.(b) Completion time CDF of 8 flows.

Fig. 13. Fairness of DCQCN+.

C. Fairness

Fairness is also an important index for network traffic. In DCQCN+, at one congestion point, all congested flows equally share the link capacity. This is ensured by the linear ECN marking probability since large flow has a larger possibility to be marked and thus get cut. This is similar to DCQCN.

We use CDF of flow completion time to measure fairness. We compare our design to DCQCN both in small and large incast using topology in Fig. 2 with 40Gbps links. Fig. 13(a) shows the CDF of 8 flows in total. Each flow has 10,000 packets of 1000KB and starts precisely at the same time. We can see that DCQCN+ is more fair and has smaller average completion time than DCQCN in such cases. Fig. 13(b) shows the CDF of 800 flows in total, i.e. 100 flows per host. To mimic the real environment, all flows are started in a uniform random time of (0,1) microseconds. Each flow has 30,000 packets of 1,000B. In such case, DCQCN cannot alleviate the congestion and PFC lasts till the end. Flows are periodically paused and resumed, thus DCQCN is good in fairness. It is not strange that DCQCN+ is not that fair in large incast cases since the recovery is very fast thus litter jitter may cause large throughput difference. Anyway, DCQCN+ has only 4% longer maximal flow completion time than DCQCN in such cases, and still has a better average flow completion time.

D. Implementation Cost

It is valuable to discuss the cost of implementing DCQCN+ and deploying it in datacenters. DCQCN+ does most part of jobs on endpoints, thus it does not have special requirements for switches. Basically, PFC and RED-based ECN should be equipped. Priority should be supported to transport control signals such as CNPs.

The NP algorithm is implemented in the smart NICs. NIC has to query, insert, update and delete the flow list. This is an additional computing cost compare to DCQCN. A record in the list needs at most 3 bytes for flow identification, 1 byte for the ECN bit and 4 bytes to record last CNP time, thus we need 8 bytes for each record. To conclude, the size of list increase 1KB for each 125 flows. This is an affordable space cost. It is lucky that we do not need to lock the list when updating since the worst result is one additional or missing CNP.

The RP algorithm is also implemented in the smart NICs. We use two timers and one state for each flow, and a rate-limiter should be implemented. These requirements are just the same as DCQCN has, thus can be implemented. We do not use byte counter and all kinds of rate recovery are based on timers, thus this cost can be cut.

E. PFC Influence on Rate Recovery

One may consider that the pause of flows caused by PFC has influence on rate recovery, and regard it as a possible reason that DCQCN fails. A reasonable choice is to pause corresponding timers when a priority queue is paused. But to pause a timer may be not that easy for hardware implementation in NICs. Besides, we find this actually not crucial in our experiments. DCQCN breaks convergence without it and our design remains working. But for robustness, we use a simplified solution in DCQCN+. When a timer is out, we check the pause state of the priority queue of that flow first, and only conduct rate recovery and state increase when this queue is not paused. This is much easier to implement and can limit the recovery of a paused flow.

VII. Related work

Reactive congestion control schemes start to work and deal with congestion after it happens. Congestion can be measured by many ways, such as queue length and RTT. DCQCN, just like DCTCP, QCN, are based on ECN while TIMELY is based on RTT. [13] compares DCQCN and TIMELY on different aspects of performance, and claims that ECN is a better congestion signal compared to delay. [15],[16] and [13] analyze the stability and convergence of similar congestion control schemes. [17] improves ECN with Micro-burst traffic using Combined Enqueue and Dequeue Marking (CEDM). [18][19] give theoretical analysis of ECN.

Proactive congestion control schemes use other special techniques to avoid congestion before it happens. [20][21] use additional control signals called *credit* or *token* to control the permission for flows to send, thus possibly avoiding incast before it happens. In such methods there are other design points and problems to be discussed.

VIII. Conclusion and Future Work

We have shown the problems that we observed when using DCQCN. Convergence are broken when there are a large number of congested flows. In this case, extremely large buffer queue length results in very high latency and more troubling long-lasting PFC for preventing packet loss. We analyze DCQCN's failure and give several reasons. Using these insights, we designed DCQCN+ and keep revising it. DCQCN+ has similar performance compared to DCQCN under small incast. Meanwhile it is capable to deal with large incast.

For future work, there are still improvements on throughput and fairness to explore. The strict analysis and proof of convergence and throughput of DCQCN+ remain. Discussion on parameters sensitiveness should also be included in this analysis. Besides, comparing different kinds of congestion control schemes to find the optimal solution for datacenter traffic is actually the final blueprint.

Acknowledgment

The authors would like to thank our shepherd Monia Ghobadi and anonymous reviewers for their valuable comments. We also thank Usama Naseer for proofreading our camera ready version. This research is supported by the National Key R&D Program of China 2018YFB1003202, the National Natural Science Foundation of China under Grant Numbers 61772265, 61602194, 61502229, 61672276, and 61321491, the Collaborative Innovation Center of Novel Software Technology and Industrialization, and the Jiangsu Innovation and Entrepreneurship (Shuangchuang) Program.

References

[1] *InfiniBandTM Architecture Specification Volume 1 Release 1.3*, https://cw.infiniband.org/document/dl/7781, Infiniband Trade Association, 2015.

[2] *RoCEv2*, https://cw.infiniband.org/document/dl/7781, Infiniband Trade Association, 2014.

[3] *802.1Qbb - Priority-based Flow Control*, http://www.ieee802.org/1/pages/802.1bb.html, IEEE DCB.

[4] C. Guo, H. Wu, Z. Deng, G. Soni, J. Ye, J. Padhye, and M. Lipshteyn, "Rdma over commodity ethernet at scale," in *Conference on ACM SIGCOMM 2016 Conference*, 2016, pp. 202–215.

[5] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, Y. Liron, J. Padhye, S. Raindel, M. H. Yahia, and M. Zhang, "Congestion control for large-scale rdma deployments," *Acm Sigcomm Computer Communication Review*, vol. 45, no. 4, pp. 523–536, 2015.

[6] H. Wu, Z. Feng, C. Guo, and Y. Zhang, "Ictcp:incast congestion control for tcp in data center networks," in *International Conference*, 2010, pp. 1–12.

[7] J. DEAN, "Mapreduce : Simplified data processing on large clusters," *OSDI'04, Sixth Symposium on Operating System Design and Implementation*, pp. 137–150, 2004. [Online]. Available: https://ci.nii.ac.jp/naid/10026764358/en/

[8] "The ns3 simulator." [Online]. Available: https://www.nsnam.org/

[9] "Dcqcn ns-3 simulator." [Online]. Available: https://github.com/bobzhuyb/ns3-rdma

[10] "Tcpdump." [Online]. Available: http://www.tcpdump.org/

[11] "Ib perftest." [Online]. Available: https://community.mellanox.com/docs/DOC-2802

[12] R. Mittal, V. T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats, "Timely: Rtt-based congestion control for the datacenter," in *ACM Conference on Special Interest Group on Data Communication*, 2015, pp. 537–550.

[13] Y. Zhu, M. Ghobadi, V. Misra, and J. Padhye, "Ecn or delay: Lessons learnt from analysis of dcqcn and timely," in *International on Conference on Emerging NETWORKING Experiments and Technologies*, 2016, pp. 313–327.

[14] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *Acm Sigcomm Computer Communication Review*, vol. 38, no. 4, pp. 63–74, 2008.

[15] M. Alizadeh, A. Javanmard, and B. Prabhakar, "Analysis of dctcp: stability, convergence, and fairness," in *ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, 2011, pp. 73–84.

[16] M. Alizadeh, A. Kabbani, B. Atikoglu, and B. Prabhakar, "Stability analysis of qcn: the averaging principle," *ACM SIGMETRICS Performance Evaluation Review*, vol. 39, no. 1, pp. 49–60, 2011.

[17] D. Shan and F. Ren, "Improving ecn marking scheme with micro-burst traffic in data center networks," in *INFOCOM 2017 - IEEE Conference on Computer Communications, IEEE*, 2017, pp. 1–9.

[18] H. Balakrishnan, N. Dukkipati, N. Mckeown, and C. J. Tomlin, "Stability analysis of explicit congestion control protocols," *Communications Letters IEEE*, vol. 11, no. 10, pp. 823–825, 2007.

[19] F. Kelly, G. Raina, and T. Voice, "Stability and fairness of explicit congestion control with small buffers," *Acm Sigcomm Computer Communication Review*, vol. 38, no. 3, pp. 51–62, 2008.

[20] D. Han, D. Han, and D. Han, "Credit-scheduled delay-bounded congestion control for datacenters," in *Conference of the ACM Special Interest Group on Data Communication*, 2017, pp. 239–252.

[21] M. Handley, C. Raiciu, A. Agache, A. Voinescu, A. W. Moore, G. Antichi, and M. Wjcik, "Re-architecting datacenter networks and stacks for low latency and high performance," in *the Conference of the ACM Special Interest Group*, 2017, pp. 29–42.