Receiver-Driven RDMA Congestion Control by Differentiating Congestion Types in Datacenter Networks

Jiao Zhang[†‡], Jiaming Shi[†], Xiaolong Zhong[†], Zirui Wan[†], Yu Tian[†], Tian Pan[†‡], Tao Huang[†‡]
[†]Beijing University of Posts and Telecommunications, Beijing, China
[‡]Purple Mountain Laboratories, Nanjing, China
Email: {jiaozhang, ricardoming, xlzhong, wanzr, tianyu2992, pan, htao}@bupt.edu.cn

***Abstract*—The development of datacenter applications leads to the need for end-to-end communication with microsecond latency. As a result, RDMA is becoming prevalent in datacenter networks to mitigate the latency caused by the slow processing speed of the traditional software network stack. However, existing RDMA congestion control mechanisms are either far from optimal in simultaneously achieving high throughput and low latency or in need of additional in-network function support. In this paper, by leveraging the observation that most congestion occurs at the last hop in datacenter networks, we propose RCC, a receiver-driven rapid congestion control mechanism for RDMA networks that combines *explicit assignment* and *iterative window adjustment*. Firstly, we propose a network congestion distinguish method to classify congestions into two types, last-hop congestion and in-network congestion. Then, an Explicit Window Assignment mechanism is proposed to solve the last-hop congestion, which enables senders to converge to a proper sending rate in one-RTT. For in-network congestion, a PID-based iterative delay-based window adjustment scheme is proposed to achieve fast convergence and near-zero queuing latency. RCC does not need additional in-network support and is friendly to hardware implementation. In our evaluation, the overall average FCT (Flow Completion Time) of RCC is $4\sim79\%$ better than Homa, ExpressPass, DCQCN, TIMELY, and HPCC.**

***Index Terms*—Datacenter, RDMA, Congestion control, Receiver-driven, PI controller**

I. Introduction

Datacenters are increasingly dominating the market for different types of high-end computing and distributed data storage services [1], [2]. These workloads put enormous pressure on datacenter networks to deliver ever faster throughput and extremely low latency at a low cost. More specifically, with the tendency of deploying high I/O speed storage media in datacenters, such as NVMe (Non-Volatile Memory express), the storage speed and access latency are significantly improved [2]. Therefore, datacenters become a good fit for applications with great demand for computation and storage capacity. However, to take full advantage of the distributed and high-speed computation and storage resources in datacenters, the networking stack requires to guarantee high throughput and microsecond latency communications among distributed nodes. Otherwise, the communication latency will become the bottleneck of the applications [3], [4].

Unfortunately, the traditional TCP/IP network stack incurs a lot of overhead [5]. CPU spends much time managing data transfers for write-intensive workloads, reducing the overall performance of these tasks. To solve this issue, the RDMA (Remote Direct Memory Access) technique is becoming widely used in datacenter networks [3]–[5]. The direct connection of RDMA NICs reduces the involvement of the CPU during data transmission. Meanwhile, combined with fast storage like NVMe, the RDMA can cut end-to-end communication latencies down from milliseconds to microseconds.

978-1-6654-4131-5/21/$31.00 ©2021 IEEE

However, deploying RDMA in datacenters poses great challenges on datacenter networking. Limited by the hardware resources in NICs, current RDMA congestion control relies on a simple go-back-N method to recover lost packets. Once the loss rate becomes higher, the performance of RDMA connections will dramatically deteriorate. Thus, PFC (Priority Flow Control) is used to guarantee in-network losslessness. However, PFC potentially brings fatal problems like PFC deadlock and PFC pause frame storm [5], [6]. Therefore, much attempt has been conducted to design RDMA-dedicated congestion control mechanisms to avoid packet dropping.

The goal of congestion control mechanisms is to allocate the bandwidth of congested links efficiently. The key challenge lies in that end-hosts can not obtain accurate information on network conditions easily. Most of the existing RDMA congestion control mechanisms use various metrics, such as ECN mark and RTT, to detect network conditions at the sender side [4], [7], [8]. Then iterative window adjustment schemes are proposed to solve network congestion. HPCC [3] suggests using INT (In-band Network Telemetry) to obtain accurate information on network conditions and then precisely controls the congestion window at the sender side to achieve faster convergence and lower latency. It can not be deployed if INT support is absent.

In this paper, we instead ask, *is it essential to accurately measure in-network information for congestion control mechanisms in all cases*? Most congestion happens at the last hop due to the many-to-one traffic pattern in datacenter networks, even in oversubscribed datacenter networks [9]–[12]. We call this kind of congestion *last-hop congestion*. The other congestion, which happens at other places, is referred to as *in-network congestion*. A study of Google's production datacenters reveals that *the predominant source of congestion, accounting for 62.8%, comes from the last hop in datacenter networks* [12]. Fortunately, receivers can easily obtain the last-hop congestion information. Therefore, *there is potential for designing a simple and efficient mechanism to solve the major last-hop*

congestion without obtaining in-network congestion information, while the remaining small part of in-network congestion can be further addressed by another more complicated scheme.

In this paper, enlightened by the above investigation, we propose a novel RDMA congestion control mechanism, RCC, that *combines explicit assignment and iterative window adjustment* at the receiver side. Firstly, we propose a network congestion differentiation method to detect whether the last-hop congestion happens. Inspired by the fast recovery mechanism in TCP, we use n consecutive measured RTT values to infer whether network congestion occurs or not. Then the last-hop average throughput is used to further distinguish whether last-hop congestion happens. For *last-hop congestion*, we propose an Explicit Window Assignment scheme to adjust the sending rate according to the connection number at each receiver side and piggyback the sending window value through ACK packets to senders. Besides, we combine per-ACK window adjustment and packet pacing to avoid instantaneous large queuing caused by Incast flows. In this way, the last-hop congestion that takes the majority of network congestion in datacenters can be solved in one RTT. For *in-network congestion*, we design a new iterative window adjustment scheme based on the PID (Proportional Integral Derivative) control theory. By combining the proportional and derivative terms, RCC can converge to a unique fixed point and achieve high utilization with near-zero queuing latency. Besides, RCC sets the upper bound of sending rate through the Explicit Window Assignment mechanism for each flow. In this way, RCC can avoid overlarge PID-based iterative window adjustment results.

The main advantages of RCC include: 1) it can achieve high throughput, near-zero queuing latency, fast convergence, and fairness simultaneously, 2) it does not need additional in-network features and thus can be readily deployed with traditional commodity switches, 3) it requires only a small amount of extra memory for each RDMA connection, which makes it friendly to hardware implementation.

We analyze the stability of RCC based on a mathematical model and the PID control system theory. Then we use phase margin analysis to show how to configure parameters in RCC to ensure stability and convergence.

Furthermore, we evaluate the performance of RCC both in testbed and ns-3 simulator by conducting micro-benchmark experiments as well as large-scale simulations using realistic workloads from Google and Facebook datacenters. We show RCC outperforms Homa, ExpressPass, DCQCN, TIMELY, and HPCC in terms of mean and tail flow completion time, convergence rate, fairness, and queuing latency. Large-scale simulations show that RCC achieves 55% lower average FCT and 79% lower 99^{th} percentile FCT than TIMELY and DCQCN for typical datacenter topology and workload settings. Compared with HPCC, RCC is fairer and achieves better performance in multiple scenarios.

In summary, our key contributions are:

- We leverage the characteristic that most of congestion happens at the last hop to design RCC, a high-performance transport for RDMA in datacenter networks;
- We propose an Explicit Window Adjustment mechanism to fairly assign the last-hop bandwidth to senders in one-RTT for last-hop congestion. And we design a PID-based window adjustment mechanism to simultaneously achieve fairness and a guaranteed steady-state latency for in-network congestion. Besides, per-ACK window adjustment and packet pacing are combined to mitigate instantaneous large queuing latency;
- We theoretically analyze RCC on its stability and convergence and show how to tune parameters of RCC under various network conditions;
- We evaluate RCC using a DPDK implementation and large-scale simulations and compare it to existing typical RDMA and receiver-driven schemes. Our results show that the overall average FCT of RCC can be $4 \sim 79\%$ better than Homa, ExpressPass, DCQCN, TIMELY, and HPCC.

II. Background and Motivation

A. RDMA in Ethernet

Traditional TCP suffers from high CPU overhead and large latency [13]. By offloading the transport layer function to the hardware chip, RDMA is able to access (i.e., read from or write to) memory on a remote machine without interrupting the processing of the CPU(s) on that system. RDMA was previously used in lossless InfiniBand networks. To use RDMA in Ethernet and IP networks, RoCE [14] is proposed. RoCE follows the original design of RDMA for lossless networks, using PFC [15] to avoid packet loss in Ethernet and using a go-back retransmission mechanism to recover lost packets.

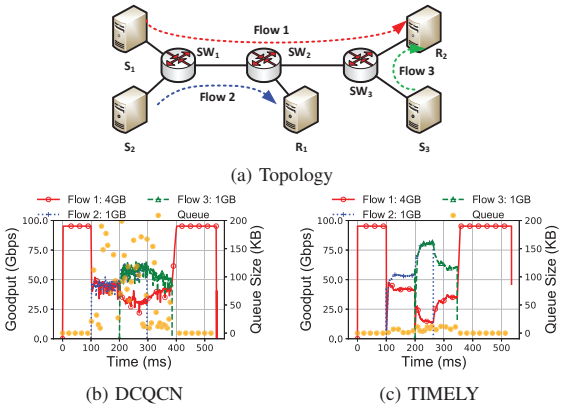
PFC is a hop-by-hop flow control mechanism to prevent buffer overflow on Ethernet switches and end-host NICs. It works in the queue granularity and sends PAUSE/RESUME frames from downstream devices to notify upstream devices to pause/resume sending packets. Because of a coarse-grained queue-level operation, PFC possibly leads to poor performance for individual flows, such as unfairness, flow transmission and head-of-line blocking. Even worse, unexpected interaction between PFC and Ethernet packets flooding possibly breaks the up-down routing and could lead to occasional deadlocks.

Go-back retransmission. The original design of RDMA in Ethernet employs go-back-0 retransmission to handle occasional packet drops, which suffers from the live-lock problem. To address this issue, a modified go-back-N retransmission is employed. Go-back-N scheme solves the live-lock problem but still wastes time and bandwidth for sending redundant packets, potentially increasing the probability of congestion.

B. State-of-the-art Congestion Control Solutions

In order to reduce the side effects of the above go-back mechanisms, flow-based RDMA congestion control solutions like DCQCN, TIMELY and HPCC, have been proposed.

DCQCN is an end-to-end rate-based congestion control mechanism [4] proposed for RoCEv2. It achieves high link utilization by fast increasing sending rates similar to QCN and fairness through fine-grained rate control similar to DCTCP [16]. However, due to coarse-grained ECN marks, DCQCN fails to



(a) Topology

(b) DCQCN (c) TIMELY

Fig. 1: Performance of DCQCN and TIMELY in a multi-bottleneck topology. Flow 1, 2, 3 starts at 0, 100, 200 ms, respectively.

know the exact congestion in networks and can not converge to the expected sending rate quickly, as shown in Fig. 1(b). Furthermore, tunning for parameters in different environments is tedious and annoying to obtain good performance.

TIMELY is an RTT-based and rate-based RDMA congestion control mechanism [8]. It iteratively adjusts the sending rate of flows based on the measured RTT and base RTT. The measured RTT can provide a more accurate estimation of congestion in networks compared with the ECN mark. However, as shown in Fig. 1(c), TIMELY can not achieve fairness and a guaranteed steady-state delay simultaneously by using only delay as the feedback signal. Its performance is comparable to or worse than DCQCN [17].

HPCC [3] requires the INT support from switches to obtain precise link load and uses an iterative algorithm to obtain appropriate sending rates for connections. However, the fairness and tail latency performance of HPCC degrades under some circumstances, as shown in §V-B.

C. Most of Congestion Happens at Edge

Topology. Datacenter network topology plays a vital role in determining the communication bandwidth and latency between each pair of nodes. The tree-based hierarchical topology with two or three tiers according to the network scale is widely used in practice [18]–[21]. These topologies generally have sufficient cross-sectional bandwidth and the core network will not become a bottleneck [9].

Communication pattern. Datacenters employ the scale-out method to support large-scale applications. Generally, an application is supported by tens of hundreds of servers [25]. To complete a task, these servers need to communicate with each other frequently. Due to the sufficient cross-sectional bandwidth in datacenters and the widely existed many-to-one or many-to-many communication patterns [10], *most congestion happens at network edges in both non-oversubscribed and over-subscribed datacenters [9], [22], [23]*. For example, some popular applications, such as key-value stores [24], data mining [25], web search, and parameter server [26] used in distributed machine learning frameworks, often generate a number of scatter-gather [10], [27] and batch computing tasks [28], causing the many-to-one communication pattern. A study of Google's production datacenters reveals that **the predominant source of congestion, accounting for 62.8%, comes from the last hop in datacenter networks** [12].

D. Brief Summary

Existing state-of-the-art RDMA congestion control solutions follow a similar rationale: senders iteratively adjust sending rates of flows according to the network congestion signals. There is still room for improvement from the perspective of fairness, convergence rate, and end-to-end latency. It is potential to design a more concise and efficient RDMA congestion control algorithm by leveraging the special characteristic that congestion often happens at the last hop in datacenters.

III. Design

A. Basic Idea and Challenges

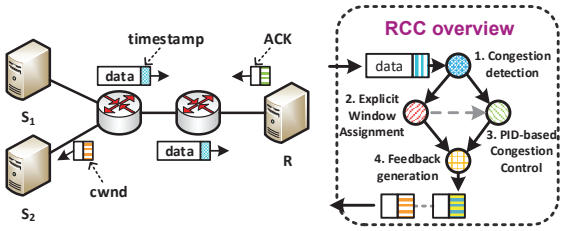
The key idea of RCC is to leverage the characteristic that most of the congestion happens at the receiver edge and allocate bandwidth to connections according to different kinds of congestion types. We classify network congestion into two types: **C1** happens at the last hop; **C2** happens at other places. The first type of congestion takes the majority of network congestion in datacenter networks [29] and can be easily solved at the receiver in one RTT. For the other in-network congestion, RCC adjusts the bandwidth allocated to each connection based on the network delay feedback.

To realize the basic idea of RCC, there are three main challenges to be solved.

1. How to differentiate different types of network congestion at receivers. To obtain the network congestion type, we first need to detect whether and where network congestion happens accurately and responsively. Generally, network congestion can be detected based on some widely-used information, such as RTT, ECN, loss rate. However, directly using the instantaneous value of them will possibly lead to overreaction. On the other hand, using the average value of them will possibly be unresponsive to network congestion.

2. How to obtain accurate bandwidth sharing when addressing congestion C1. In order to assign the last-hop bandwidth accurately and fairly to RDMA connections, a straightforward method is that receivers count the precise number of active flows and then explicitly assign the average bandwidth to each connection. However, consider an Incast scenario, flows usually start one by one with a quite short interval. In this case, at first, the receiver may assign a higher congestion window to senders, causing the aggregate sending rate of all the flows higher than the last-hop bandwidth.

3. How to achieve fast convergence and near-zero queuing latency when addressing congestion C2. Due to the limited information provided by ECN marks, it is hard to utilize ECN to achieve fast convergence as well as near-zero queuing latency. Existing delay-based congestion



(a) Workflow (b) RDMA NIC components

Fig. 2: The overview of RCC framework.

control protocols such as TCP vegas [30], FAST TCP [31], and Compound TCP [32] have inherent limitations to achieve both fast convergence and low latency in current high-speed datacenter networks. They only react after queue build-up. Although TIMELY mitigates the problem by using the delay gradient, it fails to converge to a fixed point.

B. Framework

Before proceeding to describe the framework of RCC, we first explain why RCC is window-based and delay-based.

Window-based or rate-based. In rate-based congestion control schemes, packets are continuously sent before receiving feedback, which may further aggravate congestion when feedback is delayed due to congestion. Window-based solutions can avoid this problem by limiting the number of inflight packets even if the feedback is delayed. In this way, congestion will not be magnified, making the network more stabilized.

Delay-based or ECN-based. ECN is per-hop feedback, which can prevent packet loss efficiently. However, ECN-based schemes fail to effectively control the end-to-end queuing length as the number of hops increases, while RTT is end-to-end feedback information, which can be used to control the end-to-end queuing length more effectively.

Fig. 2 shows the framework of RCC. It includes three main functions: Differentiating Congestion Types (§III-C), Explicit Window Assignment (§III-D) and PID-based Iterative Adjustment (§III-E). Each flow starts at line rate like other RDMA congestion control mechanisms [3], [4], [8]. Each data packet has a timestamp field to indicate the packet's sending time. As shown in Alg. 1, the receiver calculates the one-way delay by subtracting the timestamp value from the current time when the packet arrives (Line 3). Besides, if the packet belongs to a new flow or is the last packet for an existing flow, the receiver will update the number of active flows and calculate the new fair share (Line 4).

If the flow has been in PID-based congestion control procedure, it will stay in this state until the end (Line 6-9). This is because most flows are quite short in high-speed datacenter networks and switching between Explicit Window Assignment and PID-based congestion control may cause in-network queue oscillation. And the PID-based congestion control results will be limited by Explicit Window Assignment. Thus, last-hop congestion will not happen again.

Algorithm 1 RCC Algorithm at Receiver Side

```
1:  INPUT: data packet pkt
2:  OUTPUT: sending window cwnd
3:  rtt ← CalculateRTT(pkt)
4:  num ← UpdateFlowNumber(pkt)
5:  fair_share ← ExplicitWindowAssignment(num)
6:  if flow already in PID-based congestion control then
7:      cwnd ← PIDControl(rtt, fair_share)
8:      return
9:  end if
10: if RX rate >= NIC speed * η then
11:     cwnd ← fair_share
12: else
13:     in_network ← CongestionDetection(rtt)
14:     if in_network == true then
15:         cwnd ← PIDControl(rtt, fair_share)
16:     else
17:         cwnd ← fair_share
18:     end if
19: end if
```

Otherwise, the receiver uses one-way delay and other information to determine if in-network congestion occurs (Line 13). If in-network congestion does not happen, the receiver explicitly assigns the sending window to the fair share (Line 11 and 17). For in-network congestion, the receiver adjusts the sending window using the PID-based congestion control mechanism and the upper bound of the sending window is set to be the fair share (Line 15). After the adjustment of sending window, the receiver piggybacks this information by ACK packets to senders. The sender adjusts its sending window after receiving each ACK packet.

C. Differentiating Congestion Types

Detecting network congestion. In RCC, each packet carries the sending timestamp in its header. Upon receiving a packet, a receiver can obtain the real-time one-way delay of the corresponding connection. Note that here we assume that the clock at senders and receivers are synchronized [33].

Let RTT_i^{base} and $RTT_i(t)$ represent the base and measured one-way delay of connection i, respectively. We can use the difference between $RTT_i(t)$ and RTT_i^{base} to infer whether network congestion happens or not. If the difference between $RTT_i(t)$ and RTT_i^{base} exceeds a threshold, then RCC will decrease the congestion window of connections.

However, many flows in datacenter networks are extremely short [34], maybe containing only several packets. Besides, each connection starts at line rate. Thus, these extremely short flows possibly incur ephemerally high $RTT_i(t)$. If we directly use $RTT_i(t)$ to detect network congestion and decrease the congestion window of all connections once the instantaneous $RTT_i(t)$ is larger than RTT_i^{base}, network bandwidth will possibly suffer from being under-utilized.

Enlightened by the fast recovery mechanism in TCP, we use n consecutive $RTT_i(t)$ values to infer whether network

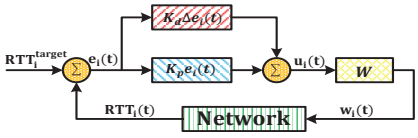


Fig. 3: Structure of PID-based congestion control.

congestion happens or not. Specifically, if n consecutive $RTT_i(t)$ values satisfy the following inequation:

$$RTT_i(t) > RTT_i^{base} \times (1+\delta),\ 0 < \delta < 1, \tag{1}$$

then we can infer that network congestion happens. δ represents the allowed congestion level caused by queuing.

Determining congestion type. First, we calculate the received bytes of all connections, B_R, in the last round[1]. The sample duration is set to RTT_i^{base}. Let $\eta \in (0,1)$ represent the expected link utilization of the last hop. If $B_R > c \times \min_i(RTT_i^{base}) \times \eta$, then we can infer that the bandwidth of the last hop has been fully utilized, where c represents the bandwidth of the last hop. Thus, network congestion happens at the last hop. Otherwise, congestion happens at other places.

D. Explicit Window Assignment

Counting the number of messages, N. Unlike the stream-oriented protocol TCP, RDMA is a message-oriented one. Thus, we can easily count the number of transmitted messages based on the begin/end mark in `IB BTH` (InfiniBand Basic Transport Header). For example, in an `RDMA Write` operation, the first packet's `opcode` field in `BTH` header is set to `RDMA Write First`; the final packet of the message has an `opcode` as either `RDMA Write Last` or `RDMA Write Last With Immediate`. Thus, we can track the number of messages accurately in RDMA NICs by checking the `opcode` field of each packet. Similarly, we can also count the number of messages generated in other RDMA operations.

Computing accurate bandwidth sharing. A receiver computes the congestion window for each connection i, $W(i) = \frac{C}{N}$. This computed value will be delivered to senders by ACKs. We mitigate the impacts of the second challenge by combining per-ACK window adjustment and packet pacing. Receivers piggyback the assigned window in each ACK packet based on the current active flow number. Therefore, the improper larger window sent to the senders will only last a very short time, that is, the time between two consecutive ACK packets. Moreover, NICs at senders use the packet pacing to add space between consecutive packets of all flows. Through these mechanisms, each ACK packet that carries larger window information can only trigger a small number of extra data packets. Correspondingly, the overall sending rate of Incast flows will not cause too large instantaneous queuing.

[1]The term 'round' here refers to RTT. That is, B_R is the sum of the bytes received by all connections in the previous RTT.

E. PID-based Congestion Control

For in-network congestion, RCC uses a PID controller to govern the dynamics of the sending window. The controller continuously adapts the window to the estimated delay in order to match RTT_i^{target}. RTT_i^{target} controls the tradeoff between the bandwidth utilization and the steady-state queue length. It should be a little larger than RTT_i^{base} and smaller than the RTT value in congestion scenarios. Thus, we let

$$RTT_i^{target} = RTT_i^{base} \times (1 + \frac{\delta}{2}) \tag{2}$$

Computing the control factor. Fig. 3 illustrates the structure of PID-based congestion control by which RCC handles in-network congestion. It continuously calculates the error value $e_i(t)$ of connection i as the difference between the actual measured value $RTT_i(t)$ and RTT_i^{target}, that is,

$$e_i(t) = RTT_i(t) - RTT_i^{target} \tag{3}$$

To compute the control factor $u_i(t)$, which is used to adjust the sending window, the controller applies a correction based on proportional, integral, and derivative terms. In RCC, the parameter of integral term K_i is set to 0. Equation (4) expresses the overall control function. The proportional term gives an instantaneous response to the error value $e_i(t)$, while the derivative term is an estimation of the future trend of $e_i(t)$.

$$u_i(t) = u_i(t-1) + K_p \times e_i(t) + K_d \times (e_i(t) - e_i(t-1)) \tag{4}$$

The proportional term can ensure that the PID-based congestion control mechanism converges to a fixed point, while the derivative term is used to achieve rapid convergence speed. Finally, with proper settings of these two parameters, RCC can maintain near-zero steady-state queues in the network without compromising other performance metrics.

Computing the sending window. RCC uses the control factor $u_i(t)$ to adjust the sending window of flows. If $u_i(t) > 0$, which indicates that the inflight packets is larger than the network capacity, RCC will perform a multiplicative window decrement. Otherwise, a multiplicative window increment will be conducted since the network has available bandwidth. For ease of deployment, we use $tanh(\cdot)$ (a function ranges in $(-1,1)$) to scale the window size as follows:

$$W_i(t) = W_i(t-1) \times (1 - tanh(u_i(t))) \tag{5}$$

Due to the derivative term in (4), the increment of window size will gradually decrease as the window becomes larger, eliminating the unfairness caused by pure MIMD algorithms.

F. Parameters and Overhead of RCC

Parameters of RCC. The congestion differentiation module of RCC has three parameters: n, δ, η. n and δ control the tradeoff between throughput and transient queue length. To maximize throughput, transient queues are inevitable. We set $n = 3$ and $\delta = 0.2$ for high throughput and near-zero queuing. And the expected utilization of the last hop, η is set to 0.95.

The PID-based congestion control mechanism has two additional parameters: K_p and K_d. They control the speed of

TABLE I: State variables of RCC

	State Variable	Description	Size (Byte)
Sender	cwnd	Congestion Window	4
	snd_una	Send Unacknowledged	4
	snd_nxt	Send Next	4
Receiver	flow_num	Current Flow Number	4
	rcv_nxt	Receive Next	4
	last_rtt	The Last Measured RTT	8
	last_rtt_diff	The Last RTT Diff	8
	cwnd	Congestion Window	4
	update_timer	RTT Update Timer	4
	base_rtt	Base RTT	8
Total			52

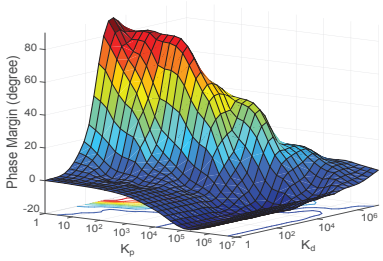


Fig. 4: Phase margin as a function of parameter K_p and K_d.

convergence to fairness and steady-state. The larger the K_d is, the faster the convergence speed will be. However, larger K_d will cause oscillation. We will discuss this in detail in §IV.

Overhead of RCC. Table I summarizes all state variables required to be maintained per RCC connection. Collectively, RCC uses 52 bytes in the sender and receiver for each RDMA connection. This memory footprint is comparable to other state-of-the-art RDMA congestion control protocols. For example, DCQCN adds $\sim$ 60 bytes for its ECN-based congestion control [4].

G. Deployment

Clock Synchronization. The deployment of RCC relies on high precision clock synchronization throughout the datacenter network. Some recent research efforts can reduce the upper bound of clock synchronization within a datacenter to a few hundred nanoseconds, which is sufficient for our work [35], [36]. And the recent work, On-Ramp, is also trying to use the one-way delay to solve datacenter network congestion [37].

Even without high precision clock synchronization, RCC can still be practically deployed by moving the delay calculation and PID-based congestion control to the sender side; the receiver side only calculates and feeds back the flow number N and the throughput B_R. This solves the problem of not being able to obtain the one-way delay.

IV. Theoretical Analysis

In this section, we analyze the stability of RCC by developing a fluid model for it, along with the control system theory [38]. Considering N long-lived flows traversing a single bottleneck link with capacity C, taking account of the relationship between q and RTT, i.e., $R_i(t) = \frac{q(t)}{C} + d$, the non-linear, delay-differential equations below describe the dynamics of $W_i(t)$, $q(t)$ and $R_i(t)$:

$$\frac{dW_i(t)}{dt} = -\frac{W_i(t)tanh(u_i(t))}{R_i(t)} \tag{6}$$

$$\frac{dq(t)}{dt} = \sum_{i}^{N} \frac{W_i(t)}{R_i(t)} - C \tag{7}$$

$$\frac{dR_i(t)}{dt} = \frac{1}{C}\frac{dq(t)}{dt} \tag{8}$$

where d is the shared propagation delay, $R_i(t)$ and $\frac{q(t)}{C}$ denote the RTT and shared queuing delay, respectively.

As for the control factor $u_i(t)$ in (4), we take bilinear transformation [39] to convert it into a continuous one:

$$\frac{du(t)}{dt} = \frac{K_p}{R_i(t)}[R_i(t) - R_{ref}] + (K_d + \frac{K_p}{2})\frac{dR_i(t)}{dt} \tag{9}$$

where $R_{ref} = RTT^{target}$ is the expected value in equilibrium.

Equation (6) describes the variation of the sending window along with the difference between the real value and the reference value of RTT measured at receiver side, while Equation (7) indicates the queuing process at the switch. Equations (8) and (9) capture the evolution of direct and indirect control signals, respectively. By letting the LHS of (6)-(9) equal 0, with the assumption that all flows are synchronized and peak simultaneously (which is obvious), it is easily verified that W, q and RTT do not reach the steady-state until they satisfy:

$$tanh(u_i)* = 0 \quad and \quad RTT_i* = R_{ref} \tag{10}$$

$$\frac{W_1*}{RTT_1*} = \frac{W_2*}{RTT_2*} = \cdots = \frac{W_N*}{RTT_N*} \tag{11}$$

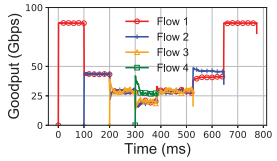
where symbol $*$ indicates the value at the fixed points, which also represents the fairness of all flows in equilibrium.

Denoting all W_i* as W_0, referring to the linearization and Laplace transformation method used in [40]–[42], we finally get the open-loop transfer function of the whole system:

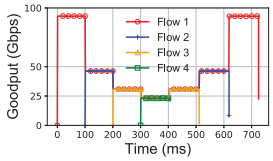
$$G(s) = K\frac{1 + \frac{s}{z}}{s^2(s + \frac{1}{R_{ref}})} \tag{12}$$

where $K = K_p/R_{ref}^2$ and $z = K_p/((K_d + K_p/2)R_{ref})$.

According to the Bode Stability Criteria, the system is stable only if the phase margin in the Bode diagram of the transfer function $G(s)$ is above 0. And the higher the phase margin, the more stable the system. Taking the setting mentioned above for parameter δ with 0.2, i.e., $R_{ref} = 13.2\mu s$, Fig. 4 shows the variation of the phase margin relative to different pairs of K_p and K_d. As shown in Fig. 4, setting $K_p \in [1, 10^4]$ and $K_d \in [10^3, 10^5]$ ensures an acceptable phase margin above 30 degrees. Besides, to make the system more stable, K_p should be one or two orders of magnitude smaller than K_d. Larger values of K_p and K_d provide a faster response to the control signal but will lead to a heavy reduction on stability.

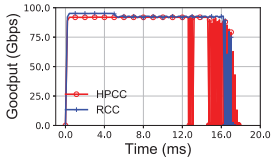


(a) HPCC

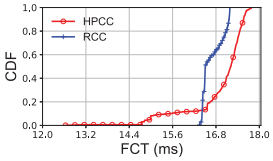


(b) RCC

Fig. 5: Fast convergence and fairness.



(a) Goodput



(b) CDF of FCT

Fig. 6: Goodput and FCT in the Incast scenario.

V. Evaluation

In this section, we evaluate the performance of RCC in hardware testbed and ns-3 simulator by conducting both micro-benchmarks and large-scale experiments. For simulations, we use the ns-3 RDMA open-source code [17] to implement RCC.

A. Evaluation Setup

Network topology. A fat-tree topology is used in large-scale simulations. It consists of 320 hosts, 20 ToR switches, 20 aggregation switches, and 16 core switches. Hosts and ToRs are connected with 100 Gbps links, while all switches are connected via 400 Gbps links. The delay of each link is set to $1\mu s$, which gives a $12\mu s$ base RTT. The switch buffer size is set to $32MB$ according to real device configurations [3].

Schemes compared. We compare RCC with DCQCN, TIMELY, HPCC, Homa [43] and ExpressPass [44]. Homa and ExpressPass are typical receiver-driven transport protocols, although they are not designed for RDMA networks. We use the open-source code of DCQCN [17], Homa [45] and HPCC in our evaluation with $\eta = 0.95$ and $maxStage = 5$ for HPCC. As for ExpressPass and TIMELY, we implement them in ns-3 based on their algorithms.

Parameter settings. For RCC, we set $\delta = 0.2$, which leads to a $13.2\mu s$ RTT_i^{target}. Then, we set $K_p = 1 \times 10^4$ and $K_d = 1 \times 10^5$ based on the analysis in §IV. For other schemes, we use the parameters suggested in the corresponding papers. We also scale the ECN marking threshold proportional to the link bandwidth suggested in [3] for DCQCN.

Benchmark workloads. We use four kinds of realistic data-center workloads that are widely used in prior literature, web search [16], data mining [19], web server [11], and cache follower [11]. Due to the high overhead of end-host processing caused by the traditional TCP/IP network stack, there is a trend to employ RDMA in these workloads [46], [47].

Performance metrics. The performance of RCC is evaluated using the following metrics: (1) goodput, (2) convergence speed/fairness, (3) average FCT, (4) tail FCT, and (5) in-network queuing length.

B. Micro-benchmarks

We compare RCC with HPCC in several micro-benchmarks since HPCC is regarded as the best solution for the time being.

1) Fast convergence and fairness. We show that RCC flows can quickly react to changes in available bandwidth and rapidly converge to appropriate flow rates. Besides, we evaluate the fairness of RCC.

Setup: We use a dumbbell topology, where 4 senders and 1 receiver are connected with the same 100 Gbps bottleneck link. 4 senders transmit one flow to the receiver in turn with an interval of $100ms$. The lengths of the 4 flows are 4.4, 2.2, 1.1 and 0.27 GB, respectively.

Result: Fig. 5 shows the goodput of the 4 flows in RCC and HPCC. RCC quickly throttles the rate of existing flows upon a new flow starts, and recovers the rate of other flows after a flow ends. This is because the receiver has precise information about the flow number being transmitted. When a flow finishes, RCC only needs one RTT to re-assign the sending window for existing flows to fully utilize the newly available bandwidth. As a result, the overall goodput of RCC is more stable. However, HPCC needs several RTTs to converge to a stable rate after a flow arrives/finishes.

Fig. 5 also illustrates the fairness of RCC and HPCC. RCC provides better fairness even in a short time scale. All flows evenly share the bottleneck bandwidth and grab their fair share quickly. Specifically, when N varies from 1 to 4, each flow's goodput quickly converges to $\sim \frac{100 \times 0.95}{N} Gbps$, giving a Jain fairness index within $0.998 \sim 0.999$ (1 is optimal). However, flows in HPCC can not get the fair share under a various number of concurrency and the goodput suffers oscillation.

2) Incast. Next, we evaluate the performance of RCC under the Incast scenario.

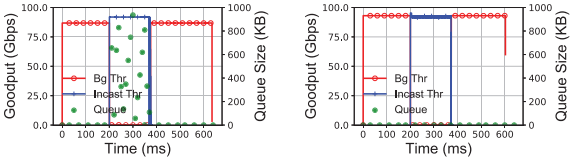
Setup: One receiver initiates connections with 1000 senders and requests $200KB$ data from each sender simultaneously. This workload is similar to the workload used in [48]. Besides, there is also a long-lived background flow to the receiver. We evaluate the overall FCT and the goodput of flows.

Result: Fig. 6 illustrates how RCC and HPCC react to congestion caused by *Incast*. The aggregate goodput of RCC and HPCC is similar. The total goodput of the 1000 flows remains stable as time goes on, at around $94.98Gbps$. As for the average FCT of Incast flows, RCC performs much better than HPCC. RCC improves the median of FCT by about 5.2% and improves the 99^{th} percentile FCT by about 4.1%.

3) Low queuing latency. Incast traffic likely causes instantaneous large queuing latency due to its burstiness. We next show the queuing latency of RCC in this scenario.

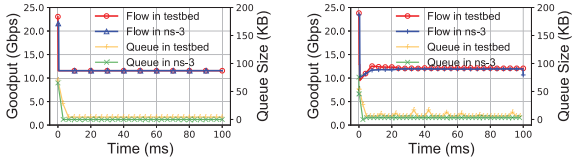
Setup: We use the same Incast traffic as above and measure the queue length at the bottleneck switch.

Result: Fig. 7 shows the queue length of HPCC and RCC under the Incast scenario. RCC keeps near-zero queuing length since it explicitly assigns window size and combines per-ACK feedback and packet pacing. In HPCC, the queue builds up to



(a) HPCC (b) RCC

Fig. 7: Queuing length under Incast scenario.



(a) Last-hop congestion scenario. (b) In-network congestion scenario.

Fig. 8: Testbed experiments.

$900KB$ due to its slow responsiveness to the large number of concurrent flows.

C. DPDK Evaluation

Because current RDMA NICs do not provide flexible programming capabilities to implement the algorithms in RCC[1], we use DPDK [49] to cross-validate with simulation results.

We plug in a Mellanox ConnectX5-EN NIC on each of the two Dell PowerEdge R730 servers to act as sender and receiver, respectively. Each NIC has two $25Gbps$ Ethernet ports, so the server can work as two senders or receivers. We build a two-tier network topology with three switches and then connected the servers to the two leaf switches. The rate of all links is $25Gbps$. Each server in this topology is equipped with two Intel Xeon E5-2609 v4 CPUs (8 cores, 1.7 GHz), 128 GB 2133 MT/s DDR4 RAM. We run two different experiments on both the testbed and ns-3 simulator, and measure throughput and switch egress queue length. In both testbed and simulator, the RTT is about $12\mu s$. So the parameters are set to be the same in hardware experiments and ns-3 simulations.

In the first experiment, each sender generates one flow to the same receiver, which results in the last-hop congestion. Fig. 8a shows that the sending rate stabilizes at $12Gbps$ with a near-zero queue length for both the testbed and simulation.

In the second experiment, each sender generates one flow to different receivers, which results in the in-network congestion. Fig. 8b shows that the sending rate stabilizes at $12Gbps$ for both the testbed and simulation. In the steady-state, the throughput of two senders oscillates with low amplitude in both testbed and simulation and the queue length is limited to a very low level. From the testbed experiments, we conclude that RCC would behave as expected under real datacenter network environments, and the results of the simulations are valid.

D. Large-scale Simulations

Now, we evaluate the performance of RCC in large-scale scenarios. We use the fat-tree topology mentioned in §V-A and generate traffic according to the four realistic workloads.

1) Overall performance.

Result: Fig. 9 shows the average FCT achieved by each scheme under the four types of workloads and different link loads. The performance of RCC is better than Homa, ExpressPass, DCQCN, TIMELY and HPCC. In the web search workload, the overall average FCT with RCC is up to 9%, 11%, 11%, 30% and 45% lower compared with HPCC, Homa, ExpressPass, DCQCN and TIMELY, respectively. And the results change to 7%, 4%, 15%, 18% and 19% under the data mining workload. Besides, RCC delivers the best performance under the web server and cache follower workloads. The improvement of RCC over HPCC in the overall average FCT is also obvious: $5 \sim 11\%$ for the web server workload and $\sim 14\%$ for the cache follower workload.

2) FCT breakdown based on flow size.

Result: We break down the FCT across flow sizes as shown in Fig. 10 and Fig. 11. We omit the results of web server and cache follower workloads since the performance is consistent with the other workloads.

For short flows, though under which Homa is proved to be highly effective, RCC performs better than it and HPCC, and greatly outperforms ExpressPass, DCQCN and TIMELY. This is because RCC keeps near-zero queue length and handles congestion rapidly by combining the Explicit Window Assignment mechanism with the PID-based congestion control, while the core packet spraying scheme used in Homa may encounter various issues. Fig. 11(a) and 11(b) show that RCC has a slightly better performance than HPCC both in average FCT and 99^{th} percentile FCT under the data mining workload. Besides, RCC improves the average FCT by $10 \sim 25\%$ and improves the 99^{th} percentile FCT by $21 \sim 50\%$ compared to DCQCN. In the web search workload, RCC still achieves better performance than HPCC and gains similar FCT reduction over the other four schemes as under data mining workload.

As for long flows, RCC performs better than all other schemes. In the data mining workload, RCC performs better than HPCC in terms of the average FCT and 99^{th} percentile FCT. RCC also achieves great performance for long flows in the web search workload. Compared to DCQCN, RCC reduces by $25 \sim 31\%$ for the average FCT and up to 37% for the 99^{th} percentile FCT. This is because RCC can rapidly reach the fair share by the Explicit Window Assignment algorithm.

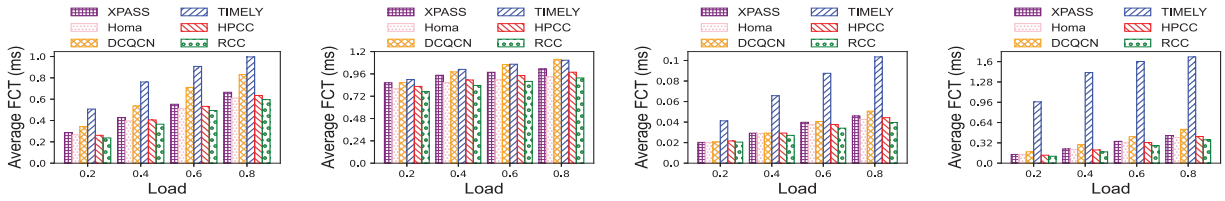
E. RCC Deep Dive

In this section, we dig deeper into RCC's design by conducting a series of targeted simulations.

1) Impact of time jitter and parameter n.

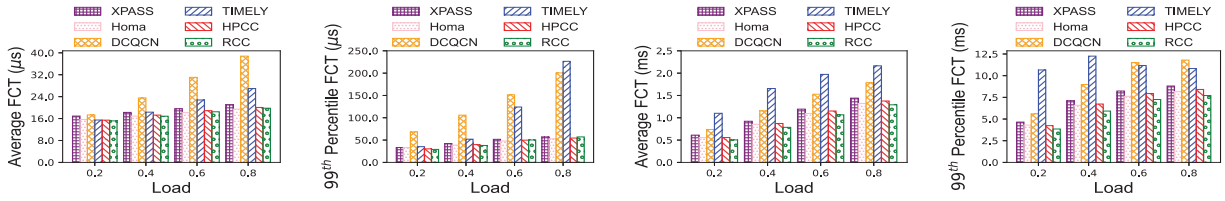
Setup: To explore the impact of clock synchronization and congestion detection parameter n on RCC, we run RCC, RCC-async (with random time jitter), RCC-sender (modified as elaborated in §III-G) and RCC(n=1) (setting parameter n to 1 based on RCC-async) using the same settings in §V-B-2).

[1]This is claimed from the perspective of the implementation of self-defined algorithms.



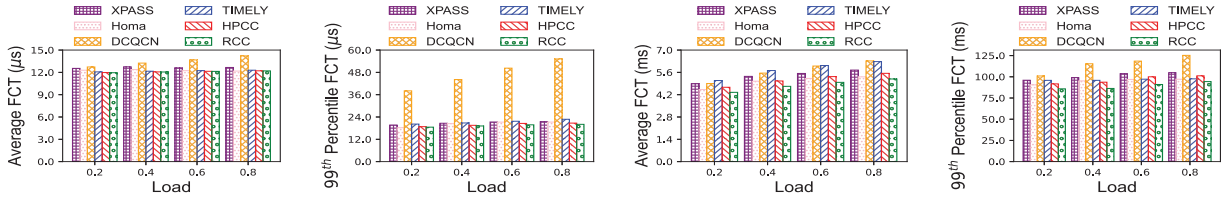
(a) Web Search (b) Data Mining (c) Web Server (d) Cache Follower

Fig. 9: Overall performance.



(a) 0-100KB average FCT (b) 0-100KB tail FCT (c) >100KB average FCT (d) >100KB tail FCT

Fig. 10: Overall performance in web search workload.



(a) 0-100KB average FCT (b) 0-100KB tail FCT (c) >100KB average FCT (d) >100KB tail FCT

Fig. 11: Overall performance in data mining workload.

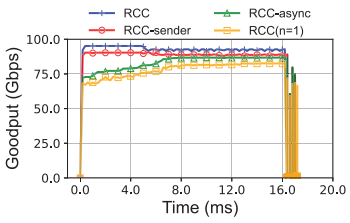


Fig. 12: The impact of time jitter and parameter n on RCC.

Result: Fig. 12 illustrates that time jitter caused by clock synchronization does lead to a deterioration on the goodput, but not pretty severe as the result in RCC-sender achieves similar steady and high throughput compared with RCC, which again verifies the deployment choice in §III-G. Besides, the difference between RCC-async and RCC(n=1) reveals that instantaneous delay rag would fuzz up congestion detection. Actually, RCC gives a similar performance when $n > 2$, and we choose $n = 3$ as the default one for simplicity.

2) Effect of Explicit Window Assignment and PID-based Congestion Control.

Setup: To evaluate the effect of the schemes in §III-D and §III-E, we compare the complete RCC with RCC-PID (RCC without Explicit Window Assignment) and RCC-EWA (RCC without PID-based Congestion Control) in the fat-tree topology with the web search workload.

Result: Fig. 13 shows the average FCT of RCC, RCC-PID and RCC-EWA. We observe that there is always a gap in the FCT results with RCC in the last two schemes, which indicates that both EWA and PID are vital for better performance in RCC. Specifically, RCC outperforms RCC-PID and RCC-EWA by up to 28% and 19% on average, and the result of tail FCT breakdown is similar. The gap would be remedied by equipment with either the EWA or PID part, as detailed in the next simulation.

3) The effects of EWA and PID in more detail.

Setup: We run RCC-PID, RCC-EWA and RCC using the multi-bottleneck topology in Fig. 1(a) with three flows to quantify the benefits of combining these two schemes. Flow 1 begins to transmit data at line rate, then we start flow 2, 3 at $200ms$ and $300ms$ to construct both in-network and last hop congestion with flow 1. Besides, we add flags in EWA and PID units to check the validity and accuracy of congestion detection

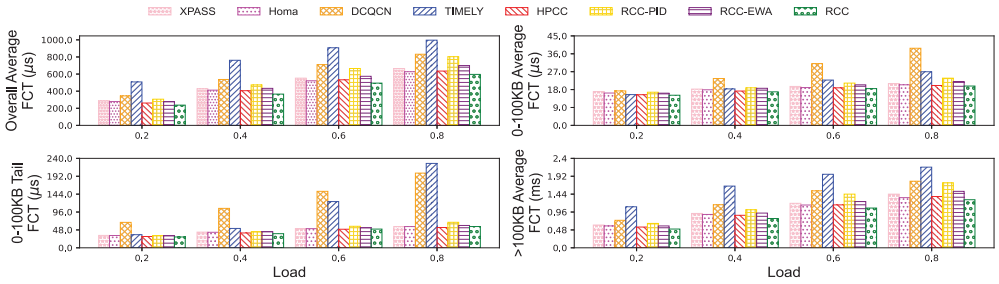
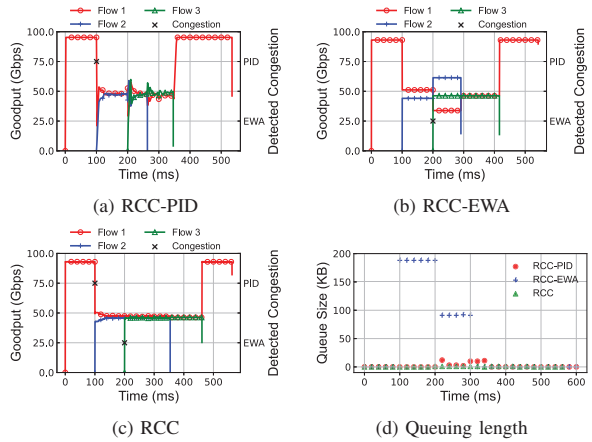


Fig. 13: RCC deep dive in web search workload.



(a) RCC-PID (b) RCC-EWA

(c) RCC (d) Queuing length

Fig. 14: RCC deep dive with multi-bottleneck topology.

and differentiation algorithms. All links are $100Gbps$.

Result: Fig. 14 shows the goodput of 3 flows in each scheme. We observe from Fig. 14(a) that RCC-PID requires several iterations to reach the proper sending rate every time the number of flows changes. Fig. 14(b) and 14(d) show that flow 1 and 2 can not share the bottleneck bandwidth fairly and build a large queue in the network. However, Fig. 14(c) illustrates that RCC handles different kinds of congestion perfectly by combining EWA and PID parts and achieves near-zero queuing length. Moreover, the consistency of detected congestion type with constructed congestions (denoted as $\times$) in Fig. 14(a)-(c) indicates that the congestion detection and differentiation in RCC are effective and accurate.

VI. Related Work

Congestion control is a sustained topic, and here we briefly introduce some closely related work.

Sender-based congestion control: DCTCP [16] is the first ECN-based congestion control algorithm used in datacenter. It adjusts each flow's sending rate by observing the ECN marks in each RTT. With the popularity of RDMA in datacenter networks, some sender-based congestion control mechanisms are proposed, which passively adjust the transmission behavior of each flow according to various feedback congestion signals. DCQCN [4] reacts to ECN marks, TIMELY [8] and Swift [50] use RTT variation, while HPCC [3] relies on precise link load information. These solutions require several RTTs to iteratively converge to the fair share, which is relatively slow for short flows that last only a handful of round-trips.

Receiver-driven congestion control: To avoid possible performance degradation under the Incast scenario, some proposals suggest shifting the control entity to the receiver side. ExpressPass [44] prevents congestion by explicitly sending a proper number of tokens to senders. However, RDMA NICs are hard to maintain different timers to implement the pacing of tokens for every flow. Homa [43] uses priority queues to schedule packets dynamically. However, since the core packet spraying [51] scheme likely incurs packet reordering, it is not well-supported in RDMA networks. Besides, pHost [52] and Homa can address congestion at the last hop, but their effectiveness depends on the assumption that the congestion occurs on the ToR downlink, which is not always true [12].

Switch-driven solutions: RoCC [39] and BFC [53] are two typical mechanisms whose control entity locates at the switch. RoCC proactively computes the fair flow rate and piggybacks it to the sender based on PI controller at the switch, while BFC tracks active flows to achieve accurate per-hop per-flow flow control and is compatible with end-to-end solutions.

VII. Conclusion

This paper presents RCC, a receiver-driven transport for RDMA in datacenters. It can efficiently utilize the network bandwidth while keeping near-zero in-network queue length. By differentiating network congestion types, RCC employs novel *Explicit Window Assignment* and *PID-based congestion control* to address the last-hop and in-network congestion, respectively. The results of the testbed experiments and large-scale simulations validate that RCC achieves low queuing latency, high bandwidth utilization, and fairness simultaneously.

Acknowledgement

We gratefully appreciate the anonymous reviewers and our Shepherd, Chen Qian, who helped us improve the quality of this paper. This work is supported in part by National Natural Science Foundation of China (NSFC) under Grant No. 61872401 and Fok Ying Tung Education Foundation under Grant No. 171059.

REFERENCES

[1] K. Hazelwood, S. Bird, D. Brooks, S. Chintala, U. Diril and D. Dzhulgakov, "Applied machine learning at facebook: a datacenter infrastructure perspective," in Proc. IEEE HPCA, 2018, pp. 620–629.

[2] A. Klimovic, C. Kozyrakis, E. Thereska, B. John and S. Kumar, "Flash storage disaggregation," in Proceedings of the Eleventh European Conference on Computer Systems, 2016, pp. 1–15.

[3] Y. Li, R. Miao, H.H. Liu, Y. Zhuang, F. Feng and Z. Cao, "HPCC: high precision congestion control," in Proc. ACM SIGCOMM, 2019, pp. 44–58.

[4] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn and Y. Liron, "Congestion control for large-scale RDMA deployments," ACM SIGCOMM Computer Communication Review, 2015, 45(4), pp. 523–536.

[5] C. Guo, H. Wu, Z. Deng, G. Soni, J. Ye and J. Padhye, "RDMA over commodity ethernet at scale," in Proc. ACM SIGCOMM, 2016, pp. 202–215.

[6] K. Qian, W. Cheng, T. Zhang and F. Ren, "Gentle flow control: avoiding deadlock in lossless networks," in Proc. ACM SIGCOMM, 2019, pp. 75–89.

[7] Y. Gao, Y. Yang, T. Chen, J. Zheng, B. Mao and G. Chen, "DCQCN+: taming large-scale incast congestion in RDMA over ethernet networks," in IEEE ICNP, 2018, pp. 110–120.

[8] R. Mittal, V.T. Lam, N. Dukkipati, E. Blem, H. Wassel and M. Ghobadi, "TIMELY: RTT-based congestion control for the datacenter," in Proc. ACM SIGCOMM, 2015, pp.537–550.

[9] M. Handley, C. Raiciu, A. Agache, A. Voinescu, A.W. Moore and G. Antichi, "Re-architecting datacenter networks and stacks for low latency and high performance," in Proc. ACM SIGCOMM, 2017, pp. 29–42.

[10] S. Kandula, S. Sengupta, A. Greenberg, P. Patel and R. Chaiken, "The nature of data center traffic: measurements and analysis," in Proc. ACM SIGCOMM, 2009, pp. 202–208.

[11] A. Roy, H. Zeng, J. Bagga, G. Porter and A.C. Snoeren, "Inside the social network's (datacenter) network," in Proc. ACM SIGCOMM, 2015, pp. 123–137.

[12] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead and R. Bannon, "Jupiter rising: a decade of Clos topologies and centralized control in google's datacenter network," ACM SIGCOMM computer communication review, 2016, 45(4), pp.183–197.

[13] I. Marinos, R.N. Watson and M. Handley, "Network stack specialization for performance," ACM SIGCOMM Computer Communication Review, 2014, 44(4), pp.175–186.

[14] Infiniband Trade Association. Supplement to InfiniBand Architecture Specification Volume 1 Release 1.2.2 Annex A17: RoCEv2 (IP routable RoCE). 2014.

[15] "802.1Qbb - Priority-based Flow Control," http://www.ieee802.org/1/pages/802.1bb.html.

[16] M. Alizadeh, A. Greenberg, D.A. Maltz, J. Padhye, P. Patel and B. Prabhakar, "Data Center TCP (DCTCP)," in Proc. ACM SIGCOMM 2010, pp. 63–74.

[17] Y. Zhu, M. Ghobadi, V. Misra and J. Padhye, "ECN or delay: lessons learnt from analysis of DCQCN and TIMELY," in Proc. ACM CoNEXT, 2016, pp. 313–327.

[18] M. Al-Fares, A. Loukissas and A. Vahdat, "A scalable, commodity data center network architecture," ACM SIGCOMM Computer Communication Review, 2008, 38(4), pp.63–74.

[19] A. Greenberg, J.R. Hamilton, N. Jain, S. Kandula, C. Kim and P. Lahiri, "VL2: a scalable and flexible data center network," in Proc. ACM SIGCOMM, 2009, pp. 51–62.

[20] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang and Y. Shi, "BCube: a high performance, server-centric network architecture for modular data centers," in Proc. ACM SIGCOMM, 2009, pp. 63–74.

[21] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang and S. Lu, "DCell: a scalable and fault-tolerant network structure for data centers," in Proc. ACM SIGCOMM, 2008, pp. 75–86.

[22] V. Jeyakumar, M. Alizadeh, D. Mazières, B. Prabhakar, A. Greenberg and C. Kim, "EyeQ: Practical network performance isolation at the edge," in Proc. USENIX NSDI, 2013, pp. 297–311.

[23] Q. Zhang, V. Liu, H. Zeng and A. Krishnamurthy, "High-resolution measurement of data center microbursts," in Proc. ACM IMC, 2017, pp. 78–85.

[24] A. Kalia, M. Kaminsky and D.G. Andersen, "Using RDMA efficiently for key-value services," in Proc. ACM SIGCOMM, 2014, pp. 295–306.

[25] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," Communications of the ACM, 2008, 51(1), pp.107–113.

[26] M. Li, D.G. Andersen, J.W. Park, A.J. Smola, A. Ahmed and V. Josifovski, "Scaling distributed machine learning with the parameter server," in Proc. USENIX OSDI, 2014, pp. 583–598.

[27] 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, pp. 139–150.

[28] M. Isard, M. Budiu, Y. Yu, A. Birrell and D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," in Proc. ACM SIGOPS/EuroSys, 2007, pp. 59–72.

[29] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D.G. Andersen and G.R. Ganger, "Safe and effective fine-grained TCP retransmissions for datacenter communication," ACM SIGCOMM computer communication review, 2009, 39(4), pp.303–314.

[30] L.S. Brakmo and L.L. Peterson, "TCP Vegas: end to end congestion avoidance on a global Internet," IEEE Journal on selected Areas in communications, 2005, 13(8), pp.1465–1480.

[31] C. Jin, D.X. Wei and S.H. Low, "FAST TCP: motivation, architecture, algorithms, performance," in IEEE INFOCOM, 2004, pp. 2490–2501.

[32] K. Tan, J. Song, Q. Zhang and M. Sridharan, "A compound TCP approach for high-speed and long distance networks," in Proc. IEEE INFOCOM, 2006.

[33] IEEE. 1588-2019 - IEEE Approved Draft Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. 2019.

[34] A. Kalia, M. Kaminsky and D. Andersen, "Datacenter RPCs can be general and fast," in Proc. USENIX NSDI, 2019, pp. 1–16.

[35] Y. Geng, S. Liu, Z. Yin, A. Naik, B. Prabhakar and M. Rosenblum, "Exploiting a natural network effect for scalable, fine-grained clock synchronization," in Proc. USENIX NSDI, 2018, pp. 81–94.

[36] Y. Li, G. Kumar, H. Hariharan, H. Wassel, P. Hochschild and D. Platt, "Sundial: fault-tolerant clock synchronization for datacenters," in Proc. USENIX OSDI 2020, pp. 1171–1186.

[37] S. Liu, A. Ghalayini, M. Alizadeh, B. Prabhakar, M. Rosenblum and A. Sivaraman, "Breaking the transience-equilibrium nexus: a new approach to datacenter packet transport," in Proc. USENIX NSDI, 2021, pp. 47–63.

[38] F. Gene, Franklin, J. David, Powell and Abbas Emami-Naeini. Feedback Control of Dynamic Systems. Prentice Hall PTR, 2001.

[39] P. Taheri, D. Menikkumbura, E. Vanini, S. Fahmy, P. Eugster and T. Edsall, "RoCC: robust congestion control for RDMA," in Proc. ACM CoNEXT, 2020, pp. 17–30.

[40] M. Alizadeh, A. Kabbani, B. Atikoglu and B. Prabhakar, "Stability analysis of QCN: the averaging principle," in Proc. ACM SIGMETRICS, 2011, pp. 49–60.

[41] C.V. Hollot, V. Misra, D. Towsley and W.B. Gong, "A control theoretic analysis of RED," in Proc. IEEE INFOCOM, 2001, pp. 1510–1519.

[42] R. Pan, P. Natarajan, C. Piglione, M.S. Prabhu, V. Subramanian and F. Baker, "PIE: a lightweight control scheme to address the bufferbloat problem," in Proc. IEEE HPSR, 2013, pp. 148–155.

[43] B. Montazeri, Y. Li, M. Alizadeh and J. Ousterhout, "Homa: a receiver-driven low-latency transport protocol using network priorities," in Proc. ACM SIGCOMM, 2018, pp. 221–235.

[44] I. Cho, K. Jang and D. Han, "Credit-scheduled delay-bounded congestion control for datacenters," in Proc. ACM SIGCOMM, 2017, pp. 239–252.

[45] Homa simulation. https://github.com/PlatformLab/HomaSimulation/

[46] C. Mitchell, Y. Geng and J. Li, "Using one-sided RDMA reads to build a fast, CPU-efficient key-value store," in USENIX ATC, 2013, pp. 103–114.

[47] J. Xue, Y. Miao, C. Chen, M. Wu, L. Zhang and L. Zhou, "Fast distributed deep learning over RDMA," in Proc. ACM EuroSys, 2019, pp. 1–14.

[48] Y. Chen, R. Griffith, J. Liu, R.H. Katz and A.D. Joseph, "Understanding TCP incast throughput collapse in datacenter networks," in Proc. ACM WREN, 2009, pp. 73–82.

[49] "DPDK," https://www.dpdk.org/.

[50] G. Kumar, N. Dukkipati, K. Jang, H.M. Wassel, X. Wu and B. Montazeri, "Swift: delay is simple and effective for congestion control in the datacenter," in Proc. ACM SIGCOMM, 2020, pp. 514–528.

[51] 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.

[52] P.X. Gao, A. Narayan, G. Kumar, R. Agarwal, S. Ratnasamy and S. Shenker, "pHost: distributed near-optimal datacenter transport over commodity network fabric," in Proc. ACM CoNEXT, 2015, pp. 1–12.
[53] P. Goyal, P. Shah, N.K. Sharma, M. Alizadeh and T.E. Anderson, "Backpressure flow control," in Proc. Workshop on Buffer Sizing, 2019, pp. 1–3.