RDMA Congestion Control: It's Only for the Compliant

John Snyder
Duke University
jsnyder@cs.duke.edu

Alvin R. Lebeck
Duke University
alvy@cs.duke.edu

Danyang Zhuo
Duke University
danyang@cs.duke.edu

***Abstract*—RDMA networks enable low latency and low CPU utilization, and their widespread adoption in datacenters enables improved application performance. However, there are performance isolation concerns for RDMA deployed in a shared cloud environment. In particular, congestion control enforcement and congestion control algorithms in RDMA make the network susceptible to performance hacking attacks, which give the attacker extra bandwidth and cause severe congestion in the network. These attacks can increase short flow completion times by several orders of magnitude. We surface a fundamental tradeoff in congestion control between short flow completion time and performance isolation. We discuss this tradeoff and how existing approaches do not provide a robust solution. We also advocate that researchers incorporate performance isolation concerns into the design and evaluation of congestion control.**

Introduction

Cloud computing eases resource management and system administration for users while also enabling cloud providers to efficiently utilize resources. The widespread adoption of cloud computing introduces new requirements for datacenter hardware and software systems. One specific requirement is performance isolation, whereby the behavior of one application (user) cannot unfairly affect the performance of another. System designers introduced several mechanisms to support performance isolation (e.g., cache partitioning, cpu scheduling, virtual memory allocation, etc.)

As more workloads migrate to the cloud, their performance requirements are also changing. Many datacenter applications demand high throughput, low latency, and low CPU overheads. Remote Direct Memory Access (RDMA) is a possible solution to meet these increasing demands. RDMA allows applications to communicate without invoking system software on the data path by offloading various networking tasks to the hardware and exposing a simple Queue Pair (QP) interface. Today, many technology companies such as Alibaba [1] and Microsoft [2] deploy RDMA networks in their datacenters. Exploring RDMA in application design is now a major topic in the cloud systems community.

The natural next step is thus to bring the benefits of RDMA to cloud users, but this requires providing the performance isolation expected with multiple mutually untrusted users sharing the RDMA network. Since RDMA bypasses the system network stack, performance isolation is

limited to techniques implemented on the RDMA network interface (RNIC) and within the network switches, including the congestion control algorithm. Unfortunately, performance isolation in RDMA congestion control has received little attention. This work takes a first step toward answering the question: *Can misbehaved or malicious users gain more bandwidth than their fair share of bandwidth by exploiting current RDMA congestion control implementations?*

We focus our performance isolation analysis on Infiniband and RoCE [3], because they are the most commonly used RDMA standards. RoCE is an appendix on the Infiniband specification and implements the IB protocol layer in an Ethernet network. We use IB and RoCE congestion control as a proxy for RDMA networks. Malicious users trying to gain more bandwidth is not new in a traditional kernel-based datacenter networking setting. For example, users can open multiple TCP sockets to gain extra bandwidth [4]. Furthermore, a user can use UDP in an attempt to avoid congestion control. To disincentivize this behavior, researchers developed mechanisms that drop packets of users who are using too much bandwidth via fair-queuing-based approaches [5].

RDMA networks differ from TCP networks in two unique ways that render existing solutions useless. First, applications offload communication to an RNIC, and any software-based indirection on the data path would undermine the latency benefits of RDMA. Second, RDMA requires a lossless network, so a congestion control policy can not actively drop packets in the network as in the existing fair-queuing-based approaches.

We analyze two major RDMA congestion control mechanisms, DCQCN [6] and HPCC [1]. DCQCN is the default congestion control algorithm in RoCE NICs from NVIDIA Networking. Alibaba deploys HPCC and is currently in the process of being standardized by the Internet Engineering Task Force (IETF). We uncover several performance attacks that allow a user to obtain substantially more bandwidth than is fair. These attacks include:

- **Parallel QP attack**. Creating more than one queue pair.
- **Staggered QP attack**. Sending data through a set of queue pairs in a round-robin fashion to completely circumvent congestion control.
- **Shuffled Overlay Attack**. Using multiple overlay topologies for collective communication.

The key property that our attacks exploit is that RDMA congestion control algorithms fundamentally favor short flows, i.e., congestion control is enforced on a per-queue-pair basis and each queue pair starts at line rate.

These attacks allow a malicious user to harm the network performance of other well-behaving users. In our testbed experiments, an attacker can obtain 72% of the available bandwidth with a victim flow on a RoCE NIC using the DCQCN congestion control algorithm. Furthermore, ignoring congestion control causes switch packet buffers to fill with packets. This dramatically extends packet queuing delay. In a simulated RoCE datacenter using DCQCN, the 99.9% tail of small flow completion times increases by 7 times. Since RDMA networks are lossless and can suffer from tree saturation [7], these attacks could theoretically render an entire network unusable as congestion spreads through the network.

In addition to identifying the above attacks, we uncover a fundamental tradeoff between short flow completion times and the ability to mitigate these attacks. RDMA congestion control protocols start sending packets at line rate for several reasons, but primarily to allow flows smaller than the network Bandwidth Delay Product (BDP) to send all packets in one RTT. However, when a protocol allows a short flow to start at line rate, a user could imitate short flow behavior by breaking a long flow into several short flows and continuously send packets at line rate. Therefore, anytime a congestion control protocol provides exceptions for a certain flow type, it creates a vulnerability. Congestion control is an attack vector in RDMA networks, and performance isolation must be considered when designing and evaluating congestion control algorithms.

We next provide background about enforcing congestion control. We then describe various RDMA performance attacks and demonstrate them using a testbed and a simulator. This is followed by a discussion of the tradeoff between short flows and performance isolation and conclusion.

Background

Remote Direct Memory Access allows users to directly interface with hardware resources on an RDMA NIC (RNIC). RoCE and IB are the most popular RDMA standards and both follow the IB specification. In RoCE, users send messages along Queue Pairs (QPs), which are similar to sockets in TCP. There are many types of QPs, but we only consider the Reliable Connection (RC) QP type in our experiments because it is the only QP type that enables all operation types[1] and only has one destination, which works well with congestion control. However, RC has no specific characteristics that make it susceptible to our attacks, so our attacks likely work with any QP type.

When implementing RoCE/IB in hardware, there are three choices to enforce congestion control. First, congestion control is optional in IB/RoCE, so hardware is not required to support it. The second option is for the hardware to enforce congestion control per-QP, and the third option is to enforce per-service level (per-SL) [3]. Per-SL makes sense in certain topologies, like a ring, where all QPs may share a common path. However, in high radix topologies like a fat tree, per-QP congestion control is more intuitive because not all QPs on the same SL are throttled when one QP experiences congestion.

The dangers of per-flow fairness are well documented [4]. FairCloud [4] explores this space extensively, proposing several different methods to enforce rate-limiting to ensure that users cannot simply open more connections to increase their bandwidth allocations. However, only one of our proposed attacks involves using multiple QPs simultaneously to gain an advantage; the others exploit QPs starting at line rate and only require one QP to send packets at a time. Further, two of the proposed methods in FairCloud require fair queueing on switches [4]. This either requires a virtual lane per-tenant, which is unimplementable, or to approximate fair queueing with mechanisms like Core-Stateless Fair Queueing (CSFQ)[5]. However, CSFQ and similar mechanisms require dropping packets, so they are not suitable for RoCE/IB.

[1]The Dynamically Connected Transport available in Nvidia Networking NICs supports all types of operations, but it is not part of the IB standard.

RDMA Congestion Control Attacks

We introduce three performance attacks that work against the current IB/RoCE specification. The first attack involves opening and sending data on several QPs simultaneously. The second attack also sends data on several QPs but does so sequentially, continuously changing which QP sends data. This allows a user to ignore congestion control. The final attack involves changing between multiple equal-cost communication overlays. By constantly changing the source-destination pairs for communication, an application can again ignore congestion control completely. All attacks cause congestion and allow a malicious user to gain extra bandwidth.

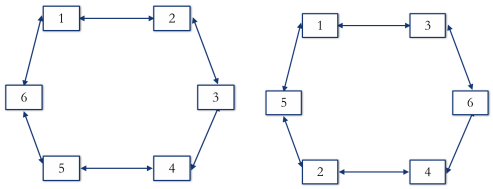
Parallel QP Attack

The Parallel QP attack requires a user to open several QPs to the same destination instead of a single QP per destination. Because IB/RoCE enforces congestion control on a per-QP basis, the share of a bottleneck link is distributed based on the number of QPs each host sends data along. The parallel QP attack is similar to opening multiple TCP sockets. However, since the RDMA network is lossless, switches cannot drop packets of misbehaving users, which is a solution to the issue in lossy networks [5].

Staggered QP Attack

The staggered attack allows a user to ignore congestion control. Unlike TCP sockets, each new QP initially sends packets at line rate. If a user continuously sends packets on new QPs, congestion control is never triggered. An RNIC waits for at least one RTT before it receives feedback from the network to reduce a QP's rate. This is because destinations generate negative feedback, either in the form of Backward Explicit Congestion Notifications (BECNs) or Congestion Notification Packets (CNPs). Sources take at least one RTT to receive congestion feedback. Assuming that the QP does not compete for RNIC resources, a QP can send at least BDP packets before it throttles its rate due to congestion.

Several factors affect the utility of a staggered attack. In a 3 tier fat tree with 100Gbps links and 1us of propagation delay, the expected RTT of the network is 12us. This means a QP sending at line rate can send 153KB before it throttles its rate.



(a) Default Rings (b) New Ring

Figure 1: Changing Ring to create more src/dst pairs.

As network characteristics change, such as longer delays or more hops, a QP can send more packets. Another concern is the size of the message. If the message size is 100MB and the BDP is 153KB, a user would need 650+ QPs to the same destination. While this is not impossible, using too many QPs can cause performance degradation due to cache misses of QPs' metadata. There may be several ways to reduce the number of QPs used. First, rates in IB/RoCE recover over time, so when a QP is left idle while other QPs send their data, the QPs rate eventually increases back to line rate. DCQCN has parameters that determine how quickly QP rates recover over time. Second, a user could destroy old QPs and set up new QPs while other QPs send data.

This attack is unique to RDMA networks. TCP starts a connection by only sending a single packet and slowly increasing the window over time, so staggering connections to have a new connection always starting harms performance. RDMA network endpoints aggressively inject traffic and only reduce their injection rate if they detect congestion. By continuously starting new connections, a user can send at line rate regardless of congestion.

Shuffled Overlay Attack

The shuffled overlay attack exploits common communication patterns to mitigate the effect of congestion control. For example, distributed data-parallel deep learning requires an all-reduce, which is often implemented with a ring communication pattern to maximize bandwidth utilization. Avoiding all to all communication allows a user to shuffle communication overlays and thus circumvent congestion control.

Figure 1 shows several unique rings a user can create with just six servers. In these rings, each server sends to a new destination, either by reversing the original communication direction, changing the overlay ring, or doing both. Figure 1(a) displays two rings, each going in a different direction. Figure 1(b) changes the overlay, so all the neighbors in the ring are new. As the system scales, there are more opportunities to create new rings. The number of possible overlays is proportional to the number of servers. Assuming all servers are connected in a clique, there are n-1 equal-cost overlays if $n > 8$ where n is the number of servers [8].

New servers, like the NVIDIA DGX A100 with multiple RNICs further exacerbate this issue. In this case, a user can create even more source-destination pairs.

Shuffling overlays enable a user to perform the staggered and parallel attacks even if a system enforces per-src/dst congestion control. A user can perform the staggered attack by sending across a new src/dst pair each RTT. This lets the attacker send at line rate and ignore congestion control. A user can perform the parallel attack by sending across several overlays simultaneously.

Attack Evaluation

We demonstrate the parallel and staggered attacks in a small cluster testbed and NS-3 simulations. We focus on the parallel and staggered attacks since the shuffled overlay attack is comprised of these primitive attacks. All evaluations take place on a cluster with 6 servers each with a 100Gbps single port ConnectX5 RoCE NIC connected to a 100Gbps Mellanox SN2100 Ethernet switch. There is a single switch, but we emulate a dumbbell topology by connecting the switch to itself and forcing all traffic through that link. Mellanox NICs use DCQCN as their congestion control algorithm. The NICs and the switch use the vendors default settings unless otherwise specified.

We experiment further in NS-3 to show the impact these attacks have in a larger setting. All simulations run with code released by Alibaba [9], which implements DCQCN [6] and HPCC [1]. The simulations demonstrate the effects of the attacks in a datacenter setting. We simulate a fat-tree with eight switches per pod, 16 core switches, and each switch is connected with a 400Gbps link. Each ToR is connected to 16

Figure 2: Experiment Dumbbell Topology

servers, and each pod has four ToRs and four Agg switches. Each server is connected to its ToR with a 100Gbps link. The flow size distribution is based on a Facebook Hadoop traffic pattern [10]. To demonstrate the severity of these attacks on network performance, we break up long flows (>1MB) into small flows of 150KB, which is approximately BDP. For the parallel attack, we start all the new small flows at the same time. When simulating the staggered attack, we space the start of each flow by the RTT of the network.

Testbed Experiments

Parallel Experiment: We perform the parallel attack by running the *ib_write_bw* test on 4 server testbed. Figure 2 illustrates the experimental topology. Both senders and receivers are on the same side of the dumbbell, so the source-destination pairs share the bottleneck link. On one sender and receiver, we open several extra QPs and run the write bandwidth test. The victim sender and receiver only open one QP. Figure 3 shows how the attacker gains more bandwidth as it opens more QPs in the default hardware configuration. We omit error bars because all standard deviations are within 1% of the mean.

This misallocation of bandwidth is due to congestion control. The bar labeled "16 No CC" shows the results when we disable congestion control in our RDMA NICs. Fair arbitration on the switch shares the bandwidth fairly between the two input ports.

Mellanox hardware does not allocate bandwidth on a per-QP basis. If congestion control enforces per-QP allocations, we expect that with an extra QP the attacker would get two-thirds of the bandwidth. However, the attacker receives far less than that. After further investigation, we discovered that our NICs did not follow the IB specification[2] and instead enforce congestion control per destination. On our NICs, all QPs to the same destination use the same send rate.

[2]Mellanox (now Nvidia) owns a patent on destination based congestion control [11]. Some QPs (UD and Mellanox's DCT [12]) can send packets to multiple destinations, so per-QP congestion control can throttle the rate of a QP even if the destination and bottleneck changed.

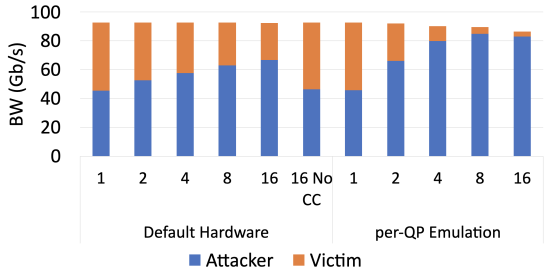


Figure 3: Parallel QP Attack Testbed Results

QPs on the same RNIC share congestion control information and send rate but do not split the rate between the QPs. For example, consider two sources sending to a shared destination on a 100Gbps link. Source 1 opens two QPs and source 2 opens 1 QP. Source 1 calculates that it should send to the destination at a rate of 30Gbps. However, instead of splitting this rate between the two QPs, both QPs send at a rate of 30Gbps. Source 1 then sends at 60Gbps to the destination. Source 2 only sends at 40Gbps. Source 1 calculates a lower rate than source 2 (30Gbps vs 40Gbps) because Source 1 receives more negative feedback from the network due to its overall higher rate (60Gbps). This results in neither per-QP fair nor per-src/dst fair.

To trick the hardware into doing per-QP congestion control, we create multiple IP addresses on the destination and open a new QP on each IP address. This allows us to emulate the IB/RoCE specification and per-QP congestion control. Figure 3 shows that in implementations that adhere to the IB/RoCE spec, the attack can get a far larger percentage of the bottleneck link.

Staggered Experiment: Next, we demonstrate the staggered attack's effectiveness in hardware. We run the staggered attack with and without a competing flow to show the upper bound on performance and also rerun the parallel attack to show the superiority of the staggered attack. We use 30 QPs, and the bandwidth delay product of our testbed's network is 37.5KB. This enables us to do a 1.125MB transfer for staggered attacks. Figure 4 shows the staggered attack results. Running the staggered attack without a competing flow achieves a throughput of about 74Gb/s; the maximum bandwidth we achieve on our 100Gb/s RNICs is 92Gb/s. When we run the staggered attack with a competing flow, the attacker receives

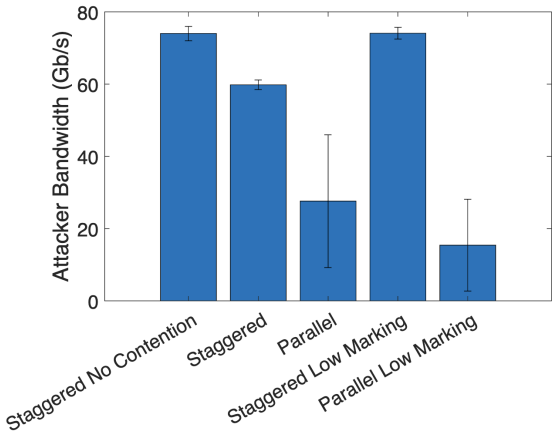


Figure 4: Parallel and Staggered Attack BW 1.125MB Transfer

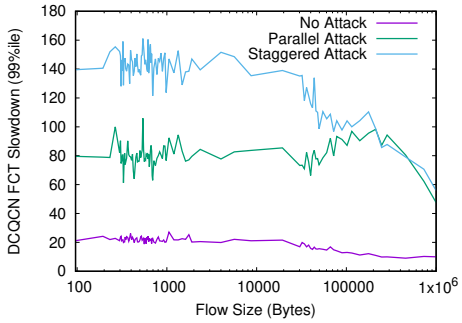


Figure 5: Victim Traffic DC Simulation

60Gb/s. Since the link is 100Gb/s, the maximum bandwidth the competing flow could receive is 40Gb/s. However, since we only achieve just over 90Gb/s, the competing flow receives less than 40Gb/s.

We can't achieve the same performance as no contention because of switch parameters and per-port fair sharing on the switch. If the victim's injection rate is at least half of line-rate, then the victim receives its fair share because the switch only allocates more bandwidth to the attacker's port if the victim's port does not send enough traffic to saturate the link. Because the attacker only sends 1.125MB and the switch does not mark packets until the switch queue depth exceeds 200KB, the victim does not reduce its rate enough for the attacker to achieve line rate in our system. To demonstrate that a longer-lived attack would be more detrimental to the victim flow, we lower the marking threshold to 8KB, so the victim backs off earlier. Figure 4 shows that "Staggered Low Marking" matches the theoretical limit of the staggered attack.

Datacenter Simulations

To show the drastic effect these attacks have on overall network health, we run 10ms of random traffic with flow sizes based on Hadoop traffic from Facebook [10]. We perform three experiments. First, we run DCQCN with per-QP fairness. We rerun the same traffic except we break every flow >1MB into smaller flows of 150KB each, so a 1MB flow becomes six 150KB flows and one 100KB flow. We start all these flows at the same time. In our third experiment, we simulate the staggered attack and break up the large flows but wait for 13us (RTT of the network) before starting each new flow. Figure 5 plots the 99% FCT slowdown of victim flows from the three experiments. The FCT slowdown is the relative slowdown to the flow's theoretical completion time without congestion (propagation delay plus serialization delay). Victim flows are smaller than 1MB because they did not break into smaller flows to get more bandwidth. The parallel and staggered attacks devastate the performance of small flows. The 99% slowdown of flows less than BDP goes from about ∼20x without the attacks to over ∼80x with the parallel attack and ∼140x with the staggered attack. We observe similar trends when we performed the same experiments with HPCC [1]. No matter the congestion control algorithm, these attacks create congestion because they allow a user to ignore the congestion control.

Potential Solutions and Future Work

The current IB/RoCE specification leaves the network susceptible to several performance attacks, and solutions to the hacks expose a fundamental tradeoff between starting flows at line rate and performance isolation. These attacks exist because congestion control is enforced per-QP and QPs start sending packets at line rate.

Solving this issue is difficult because the current design has performance benefits. For example, changing congestion control enforcement to per-src/dst easily renders the staggered and parallel attacks useless. However, enforcing congestion control on a per-QP granularity and allowing QPs to start at line rate is a performance optimization. It allows short flows to send all their bytes as quickly as possible. Enforcing congestion control on a per-src/dst granularity can throttle the rate of

	IB Spec.	CTX-5	Per-src/dst CC
Parallel	Vulner.	Vulner.	Secure
Staggered	Vulner.	Secure	Secure
Shuffled Overlay	Vulner.	Vulner.	Vulner.
SF. Penalty?	No	Yes	Yes

Table 1: Various Environments Susceptibility to Attack. Note: Vulner. = Vulnerable. SF. = Short Flow.

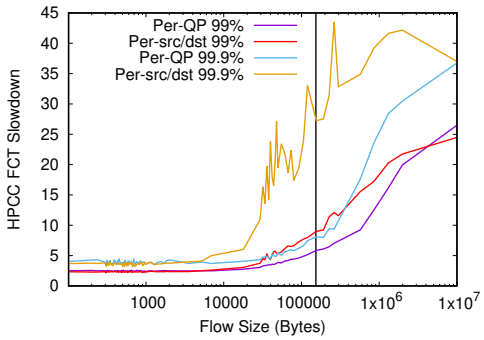


Figure 6: 99% and 99.9% tail latency of HPCC flows with different congestion control enforcement. The vertical line shows BDP.

flows when they start, and therefore short flows do not complete as quickly. Table 1 qualitatively summarizes this trade-off between isolation and short flow completion time (penalty) by showing attack vulnerability for various congestion control methods, including our Mellanox hardware.

To demonstrate the trade-off between small flow tail latency and performance isolation, we run datacenter simulations. We use HPCC instead of DCQCN since the trade-off is more apparent in protocols with a congestion window and that do not define idle behavior. Per-src/dst still causes performance issues in DCQCN, but they are not as pronounced as with HPCC.

Figure 6 shows the results of 50ms of Hadoop traffic in a network using HPCC with different congestion control enforcement. The 99.9% FCT slowdown of flows between the size of 20KB-110KB goes from 5-10x to 6-34x. This is a dramatic increase for flows that are smaller than the BDP (denoted with a vertical line in the Figure) of the network. The 99% of flows also take longer to complete, but the difference is less pronounced.

We gain two insights from this result. First, improving performance isolation in congestion control protocols can impact application performance. Second, researchers should design and evaluate congestion control algorithms with performance isolation in mind. We demonstrate that congestion control enforcement dramatically impacts performance and that it is vital to improving performance isolation. However, any solution must not drastically harm short flow performance.

Previous solutions [5] solve performance isolation issues if the network is lossy. However, IB and RoCE networks are lossless and perform better because bandwidth is not lost to drops, end-hosts do not often wait for long retransmit timeouts, and packets generally arrive in-order, which removes the need for expensive reorder resources on end-hosts.

To avoid short flows performing poorly, a solution should start flows at line-rate and then drop the new flow's packets if the new flow receives too much bandwidth. Further, when dropping packets, the solution should maintain packet ordering and detect loss without packet timeouts. This would maintain packet ordering without a go back N retransmission procedure or expensive reordering resources on destination end-hosts. A solution that met this criteria would likely ensure performance isolation while still maintaining strong performance for all flows.

Conclusion

Congestion Control ensures a network functions efficiently and users share the network fairly. However, this is only true if end users cannot abuse their network access abstractions. We present several issues with the IB/RoCE specification that allow a misbehaving user to gain an unfair advantage over other users through congestion control. We describe several performance attacks and show their effectiveness in hardware and in large-scale simulations. When exploring the solution space for this issue, we uncover a fundamental tradeoff between the completion time of small flows and performance isolation. Because of performance isolation issues, we advocate for several changes to how researchers design and evaluate congestion control. This issue is critical as applications that depend on small short-flow tail latencies move into shared environments.

Acknowledgment

This work was supported in part by the National Science Foundation (CNS-1616947).

REFERENCES

1. Y. Li, R. Miao, H. H. Liu, Y. Zhuang, F. Feng, L. Tang, Z. Cao, M. Zhang, F. Kelly, M. Alizadeh, and M. Yu, "Hpcc: High precision congestion control," in *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, (New York, NY, USA), p. 44–58, Association for Computing Machinery, 2019.
2. C. Guo, H. Wu, Z. Deng, G. Soni, J. Ye, J. Padhye, and M. Lipshteyn, "Rdma over commodity ethernet at scale," in *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, (New York, NY, USA), p. 202–215, Association for Computing Machinery, 2016.
3. Infiniband^{TM} Trade Association, *InfiniBand^{TM} Architecture Specification*, 4 2020. Volume 1 Release 1.4.
4. L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, and I. Stoica, "Faircloud: Sharing the network in cloud computing," *SIGCOMM Comput. Commun. Rev.*, vol. 42, p. 187–198, Aug. 2012.
5. I. Stoica, S. Shenker, and H. Zhang, "Core-stateless fair queueing: Achieving approximately fair bandwidth allocations in high speed networks," in *Proceedings of the ACM SIGCOMM '98 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '98, (New York, NY, USA), p. 118–130, Association for Computing Machinery, 1998.
6. 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," *SIGCOMM Comput. Commun. Rev.*, vol. 45, p. 523–536, aug 2015.
7. G. F. Pfister and V. A. Norton, ""hot spot" contention and combining in multistage interconnection networks," *IEEE Transactions on Computers*, vol. 100, no. 10, pp. 943–948, 1985.
8. T. W. Tillson, "A hamiltonian decomposition of $k_{2m}*$, 2m $\geq$ 8," *Journal of Combinatorial Theory, Series B*, vol. 29, no. 1, pp. 68–74, 1980.
9. "https://github.com/alibaba-edu/High-Precision-Congestion-Control," July 2020.
10. H. Zeng, J. Bagga, G. Porter, and A. Snoeren, "Inside the social network's (datacenter) network," *ACM SIGCOMM Computer Communication Review*, vol. 45, pp. 123–137, 08 2015.
11. N. Bloch, B. Shlomo, E. Zahavi, and Z. Yaakov, "Destination-based congestion control," Apr 2014.
12. D. Crupnicoff, M. Kagan, A. Shahar, N. Bloch, and H. Chapman, "Dynamically-connected transport service," Apr 2014.

John Snyder, is a research scientist at Nvidia Research. His research interests are congestion control in high-performance networks and parallel programming models. He received a Ph.D. in computer science from Duke University in 2022 and a B.S. in Computer Science and Mathematics from Rhodes College in 2018. Contact him at jsnyder@cs.duke.edu.

Alvin R. Lebeck, is a professor of Computer Science and Electrical and Computer Engineering at Duke University. He co-founded Phitonex in 2017, which was acquired by Thermo Fisher Scientific in 2020. His research interests are broadly in computer architecture and systems. Prof. Lebeck received the B.S. in Electrical and Computer Engineering (1989), and the M.S. (1991) and Ph.D. (1995) in Computer Science at the University of Wisconsin—Madison. He is a member of ACM and an IEEE Fellow. Contact him at alvy@cs.duke.edu.

Danyang Zhuo, is an assistant professor in the Computer Science Department at Duke University. His research interest is datacenter and cloud computing. Prof. Zhuo received his B.S. in Electrical Engineering (2013) at the University of Illinois - Urbana Champaign and his Ph.D. (2019) in Computer Science and Engineering at the University of Washington. Contact him at danyang@cs.duke.edu.