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, we find that 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.**

I. Introduction

Datacenter applications demand high throughput, low latency, and low CPU overheads. This leads to the growing adoption of Remote Direct Memory Access (RDMA). RDMA allows applications to communicate without invoking the system software on the data path by offloading the network protocol stack processing, congestion control, and packet retransmission to hardware. Today, many technology companies such as Alibaba [22] and Microsoft [13] deploy RDMA networks in their datacenters, and exploring how to use RDMA in application design has become a major topic in the networking community [19], [20], [18], [12], [16].

The natural next step is thus to bring the benefits of RDMA to cloud users, and this requires designing for multiple mutually untrusted users sharing an RDMA network. Unfortunately, the congestion control aspect has received little attention. TCP/UDP networks cannot assume that a user follows the congestion control protocol, but since RNICs enforce congestion control in hardware, which is outside the bounds of user control, RDMA users must follow the congestion control protocol. In shared environments, we must consider *can malicious users gain more bandwidth than their fair share by exploiting the current design of RDMA congestion control mechanisms?*

There are several production variants of RDMA [15], [3], [25], so answering this question broadly is impossible. However, the most common standards ~~is~~ are Infiniband (IB) and RoCE. 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 [27]. Furthermore, a user can simply use UDP to avoid congestion control. To disincentivize congestion control avoidance, researchers developed mechanisms that drop packets of users who are using too much bandwidth via fair-queuing-based approaches [28], [30].

RDMA networks differ from TCP networks in two unique ways that render existing solutions useless. First, applications offload communication to an RDMA NIC, and thus we cannot have any software-based indirection on the data path. The second and more important property is that RDMA requires a lossless network, so we can no longer drop packets actively in the network as in the existing fair-queuing-based approaches.

We analyze two major RDMA congestion control mechanisms, DCQCN [33] and HPCC [22]. DCQCN is the default congestion control algorithm in RoCE NICs from NVIDIA Networking, a major RDMA NIC vendor worldwide. 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 take substantially more bandwidth than is fair. These attacks include creating more than one queue pair (parallel QP attack), sending data through a set of queue pairs in a round-robin fashion to completely circumvent congestion control (staggered QP attack), and using multiple overlay topologies for collective communication (shuffled overlay attack). 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 times

small flow completion times increases by 7~~x~~. Since RDMA networks are lossless and can suffer from tree saturation [26], 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.

structure of paper missing

II. Background

Remote Direct Memory Access allows users to directly interface with hardware resources on an RDMA NIC (RNIC). RoCE and Infiniband (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.

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). Per-SL may make sense in certain topologies, like a ring, where all QPs may share a common path. However, in high radix topologies like a Clos [5] or a fat tree [21], [2], per-QP congestion control is more intuitive because not all QPs on the same SL are throttled when one QP experiences congestion.

reference

Not addressed

The dangers of per-flow fairness are well documented [27], [29]. FairCloud [27] explored this space extensively. They proposed several different methods of enforcing rate-limiting to ensure that users could not 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 previously ~~proposed bandwidth allocation enforcement policies~~ require fair queueing in switches. ~~This either requires a~~ virtual lane

Not clear what it refers to

[1]We omit the Dynamically Connection Transport available in Nvidia Networking NICs because it is not part of the IB standard

Only 2 references are listed

per-tenant, which is unimplementable, or to approximate fair queueing with mechanisms like Core-Stateless Fair Queueing [30] or Approximate Fair Queueing [28]. However, AFQ and CSFQ require dropping packets, so they are not suitable for RoCE/IB. Backpressure Flow Control [11] is a promising alternative. BFC dynamically assigns flows to queues and has a very low drop rate. ~~However, it still drops packets~~ and can suffer from HoL blocking in certain traffic patterns.

acronym

not defined

Previous work allocates bandwidth fairly on switches. S-perc [17] and RCP [9] add fair rates to packet headers. S-perc uses a novel distributed max-min algorithm, and RCP uses processor sharing to allocate bandwidth. All flows in S-perc follow the rate assigned by the switch. However, it provides exceptions for flows smaller than BDP, which leaves susceptible to the performance attacks in this paper. RCP requires a handshake to establish initial rate, which adds an RTT of delay to short flows [8]. Receiver based congestion control algorithms can achieve fair sharing quickly [14], [23]. Receiver based congestion control suffers from scheduling overhead and requires a non-oversubscribed network and to our knowledge no RDMA vendor uses receiver based congestion control.

III. 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.

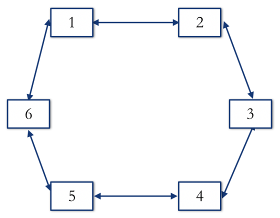
No need to explain this here: it is redundant wrt subsections

A. Parallel QP Attack

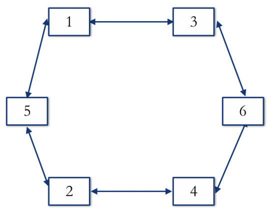
The Parallel QP attack requires a user to open several QPs to the same destination instead of a single QP per destination. Because IB/RoCE enforces congestion control on a per-QP basis, the share of a bottleneck link is distributed based on the number of QPs each host sends data along. The parallel QP attack is analogous to opening multiple TCP sockets [10]. However, since the RDMA network is lossless, switches cannot drop packets of misbehaving users, which is a solution to the issue in lossy networks [30], [28].

B. 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



(a) Default Rings



(b) New Ring

Fig. 1: Ways to change communication pattern to create more source/destination pairs.

undefined acronym

of BECNs or CNPs. Sources take at least an 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. 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 isn't impossible, using too many QPs can cause performance degradation due to cache misses of QPs' metadata [7], [19]. 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. 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.

C. 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 [31].

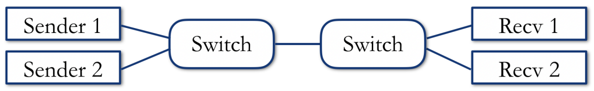


Fig. 2: Experiment Dumbbell Topology

New servers with multiple RNICs further exacerbate this issue[24]. 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.

IV. Attack Evaluation

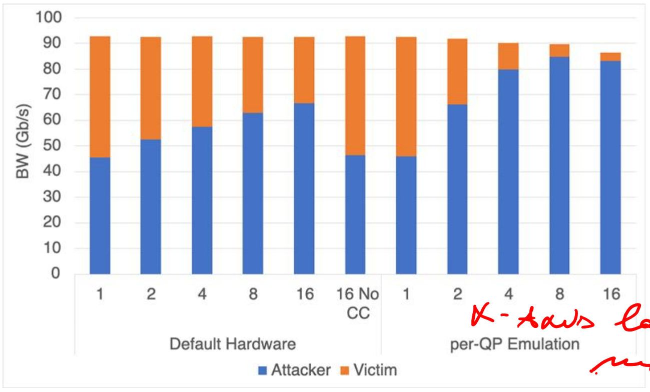
We demonstrate the parallel and staggered attacks in a small cluster testbed and NS-3 simulations. We focus on the parallel and staggered attacks since the shuffled overlay attack is comprised of these primitive attacks. All evaluations take place on a cluster with 6 servers each with a 100Gbps single 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 [1], which implements DCQCN [33] and HPCC [22]. The simulations demonstrate the effects of the attacks in a datacenter setting. We simulate a fat-tree [21], [2] with eight switches per pod, 16 core switches, and each switch is connected with a 400Gbps link. Each ToR is connected to 16 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 [32]. 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.

A drawing of the network topology would be beneficial

A. Testbed Experiments

Parallel Experiment: First, we demonstrate the parallel attack on the testbed. We run the *ib_write_bw* test on 4 servers. 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



X-axis label messy

Fig. 3: Parallel QP Attack Testbed Results

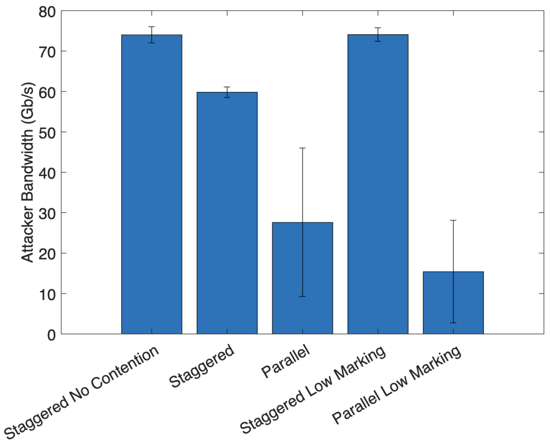


Fig. 4: Parallel and Staggered Attacker Bandwidth 1.125MB Transfer

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.

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.

[2]Mellanox (now Nvidia) owns a patent on destination based congestion control [4]. Some QPs (UD and Mellanox's DCT [6]) 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.

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 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.

B. 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 [32]. 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

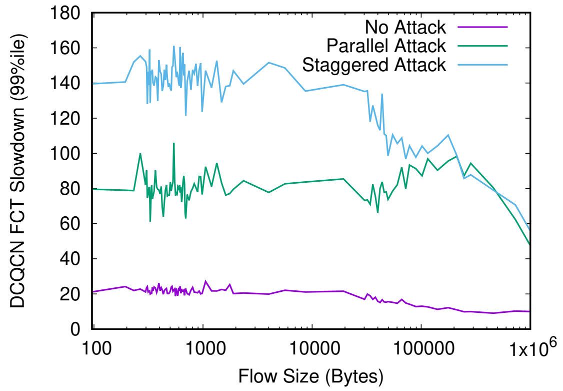


Fig. 5: Parallel and Staggered Attack Victim Traffic DC Simulation

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. 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 [22]. No matter the congestion control algorithm, these attacks create congestion because they allow a user to ignore the congestion control.

V. 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. 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. By enforcing congestion control on a per-src/dst granularity and potentially throttling the rate of flows when they start, short flows do not complete as quickly. This trade-off between isolation and short flow completion time is summarized in Table I. We also include the susceptibility of our Mellanox hardware. If congestion control allows short flows to start at line-rate, a long flow could pretend to be short ~~flows~~ and hack the system.

To demonstrate the trade-off between small flow tail latency and perforance isolation, we run datacenter simulations. However, instead of showing this trade-off in DCQCN, we

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

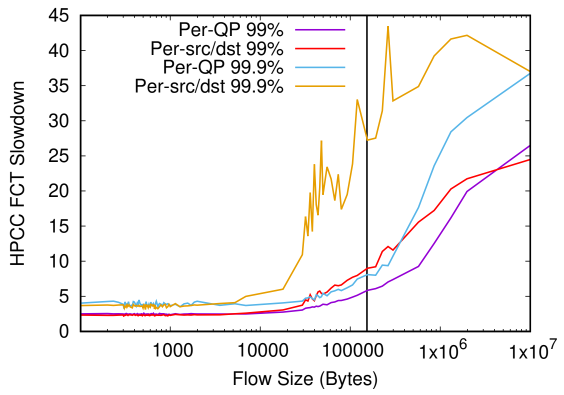


Fig. 6: 99% and 99.9% tail latency of HPCC flows with different congestion control enforcement. The vertical line shows the bandwidth-delay product.

run simulations with HPCC. This is because the trade-off is more apparent in protocols with a congestion window and that do not define idle behavior. DCQCN allows a flows rate to recover over time. Per-src/dst still causes performance issues in DCQCN, but they were not as pronounced in our experiments.

Figure 6 shows the results of 50ms of Hadoop traffic in a network using HPCC with different CC 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 garner two insights from this result. First, that making a congestion control protocol more secure has an impact on the performance of the application. Second, researchers should design and evaluate congestion control algorithms with performance isolation in mind. We demonstrated that congestion control enforcement dramatically impacts performance and that it is vital to improving performance isolation.

Further, HPCC and other congestion control algorithms, like Timely and Swift, omit characteristics that should be defined. For example, designers should consider the idle behavior of a congestion control algorithm. In DCQCN, the rate of a flow increases over time, while in HPCC an idle flow's rate does not change. Recovering the rate over time in HPCC may improve performance.

We need to explore this trade-off more extensively and determine how best to navigate it. There may be mechanisms that make the trade-off less severe without compromising performance isolation. Further, we need to determine the optimal idle behavior of a congestion control algorithm.

If rates recover too quickly, the protocol may again be susceptible to attacks.

Additionally, per src/dst fairness only solves the parallel and staggered attacks; solving the shuffled overlay attack requires a more complex solution. Because the src-dst pairs change completely, the solution is unlikely to be implementable on a NIC. Therefore, we plan on pursuing in-network solutions. Solutions to similar problems required in-network computing [28], [30], and we plan to look at these for inspiration on how to enforce fairness on switches in a lossless network.

VI. 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 found 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.

References

[1] https://github.com/alibaba-edu/High-Precision-Congestion-Control, July 2020.

[2] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. *SIGCOMM Comput. Commun. Rev.*, 38(4):63–74, Aug. 2008.

[3] M. S. Birrittella et al. Intel® omni-path architecture: Enabling scalable, high performance fabrics. In *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*, pages 1–9. IEEE, 2015.

[4] N. Bloch, B. Shlomo, E. Zahavi, and Z. Yaakov. Destination-based congestion control, Apr 2014.

[5] C. Clos. A study of non-blocking switching networks. *Bell System Technical Journal*, 32(2):406–424, 1953.

[6] D. Crupnicoff, M. Kagan, A. Shahar, N. Bloch, and H. Chapman. Dynamically-connected transport service, Apr 2014.

[7] A. Dragojević, D. Narayanan, M. Castro, and O. Hodson. Farm: Fast remote memory. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 401–414, Seattle, WA, Apr. 2014. USENIX Association.

[8] N. Dukkipati, M. Kobayashi, R. Zhang-Shen, and N. McKeown. Processor sharing flows in the internet. In *International Workshop on Quality of Service*, pages 271–285. Springer, 2005.

[9] N. Dukkipati and N. McKeown. Why flow-completion time is the right metric for congestion control. *SIGCOMM Comput. Commun. Rev.*, 36(1):59–62, Jan. 2006.

[10] L. Eggert, J. Heidemann, and J. Touch. Effects of ensemble-tcp. *ACM SIGCOMM Computer Communication Review*, 30(1):15–29, 2000.

[11] P. Goyal, P. Shah, K. Zhao, N. K. Sharma, M. Alizadeh, and T. E. Anderson. Backpressure flow control, 2019.

[12] J. Gu et al. Tiresias: A {GPU} cluster manager for distributed deep learning. In *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*, pages 485–500, 2019.

[13] 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, page 202–215, New York, NY, USA, 2016. Association for Computing Machinery.

[14] M. Handley et al. Re-architecting datacenter networks and stacks for low latency and high performance. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 29–42, New York, NY, USA, 2017. ACM.

[15] Infiniband^{TM} Trade Association. *Supplement to InfiniBand^{TM} Architecture Specification*, 9 2014. Volume 1 Release 1.2.1.

[16] Y. Jiang, Y. Zhu, C. Lan, B. Yi, Y. Cui, and C. Guo. A unified architecture for accelerating distributed {DNN} training in heterogeneous gpu/cpu clusters. In *14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20)*, pages 463–479, 2020.

[17] L. Jose, S. Ibanez, M. Alizadeh, and N. McKeown. A distributed algorithm to calculate max-min fair rates without per-flow state. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 3(2):1–42, 2019.

[18] A. Kalia, M. Kaminsky, and D. Andersen. Datacenter rpcs can be general and fast. In *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*, pages 1–16, 2019.

[19] A. Kalia, M. Kaminsky, and D. G. Andersen. Design guidelines for high performance {RDMA} systems. In *2016 {USENIX} Annual Technical Conference ({USENIX}{ATC} 16)*, pages 437–450, 2016.

[20] A. Kalia, M. Kaminsky, and D. G. Andersen. Fasst: Fast, scalable and simple distributed transactions with two-sided ({RDMA}) datagram rpcs. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 185–201, 2016.

[21] C. E. Leiserson. Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers*, C-34(10):892–901, 1985.

[22] Y. Li et al. Hpcc: High precision congestion control. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, page 44–58, New York, NY, USA, 2019. Association for Computing Machinery.

[23] B. Montazeri, Y. Li, M. Alizadeh, and J. Ousterhout. Homa: A receiver-driven low-latency transport protocol using network priorities. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, pages 221–235, New York, NY, USA, 2018. ACM.

[24] Nvidia. Nvidia dgx a100 the universal system for ai infrastructure. Technical report, 2020. [Online].

[25] C. Peterson, J. Sutton, and P. Wiley. iwarp: a 100-mops, liw microprocessor for multicomputers. *IEEE Micro*, 11(3):26–29, 1991.

[26] G. F. Pfister and V. A. Norton. "hot spot" contention and combining in multistage interconnection networks. *IEEE Transactions on Computers*, 100(10):943–948, 1985.

[27] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, and I. Stoica. Faircloud: Sharing the network in cloud computing. *SIGCOMM Comput. Commun. Rev.*, 42(4):187–198, Aug. 2012.

[28] N. K. Sharma, M. Liu, K. Atreya, and A. Krishnamurthy. Approximating fair queueing on reconfigurable switches. In *Proceedings of the 15th USENIX Conference on Networked Systems Design and Implementation*, NSDI'18, page 1–16, USA, 2018. USENIX Association.

[29] A. Shieh, S. Kandula, A. Greenberg, C. Kim, and B. Saha. Sharing the data center network. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, page 309–322, USA, 2011. USENIX Association.

[30] 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, page 118–130, New York, NY, USA, 1998. Association for Computing Machinery.

[31] T. W. Tillson. A hamiltonian decomposition of $k_{2m}*$, 2m $\geq$ 8. *Journal of Combinatorial Theory, Series B*, 29(1):68–74, 1980.

[32] H. Zeng, J. Bagga, G. Porter, and A. Snoeren. Inside the social network's (datacenter) network. *ACM SIGCOMM Computer Communication Review*, 45:123–137, 08 2015.

[33] Y. Zhu et al. Congestion control for large-scale rdma deployments. *ACM SIGCOMM Computer Communication Review*, 45:523–536, 08 2015.