Addressing Endpoint-induced Congestion for Accelerator Scale-Out in a Medium-Scale Domain

Timothy Chong*, Venkata Krishnan
Intel Corporation, USA
{Timothy.Chong, Venkata.Krishnan}@intel.com

***Abstract*—The rapid advancement of network link bandwidth in modern data centers, coupled with the relatively slower processing capabilities of host systems that include accelerators, has given rise to a new challenge: endpoint congestion. This paper presents a reactive scheme built upon a standard reliability protocol to mitigate the impact of endpoint congestion in a medium-scale domain of endpoints—one that is reachable within a few switch hops. This is arguably the sweet-spot for an accelerator scale-out domain. The proposed policy, which overloads duplicate-ACK as a reactive congestion signal, enables controlled pacing of packets from the initiator to match with target processing bandwidth, thereby avoiding packet loss due to endpoint congestion. Our results demonstrate that, for unicast and incast streaming PUT (RDMA write) flows, the proposed scheme effectively mitigates packet drops and achieves minimal, or in most cases zero, packet retransmissions when there is a drop in the endpoint processing speed. Traditional approaches fail to achieve this behavior even with high target queue capacity. Thus, our scheme also has the potential benefit of reducing the buffering requirements at the endpoint and consequently, the cost. To the best of our knowledge, our scheme is the first to explicitly consider and mitigate packet loss due to endpoint congestion, offering an effective approach to address this emerging challenge.**

***Index Terms*—congestion, transport protocol, scale-out, accelerators, network endpoint**

I. INTRODUCTION

Conventional wisdom has placed emphasis on congestion within networking hardware, specifically in network switch buffers[1]–[3]. Recently, endpoint congestion (EC) has emerged as a significant challenge for future networking systems[4]. This research work introduces an enhancement to a traditional reliable transport protocol to mitigate endpoint congestion in a medium-scale domain of accelerators.

A. Endpoint Congestion

Endpoint congestion (EC) refers to the congestion that occurs when packets arrive at an endpoint and are subsequently queued up within the networking hardware buffer due to the endpoint's limited processing capacity. Two primary factors are contributing to this problem. The first is the remarkable improvement in networking bandwidths in recent years, with 400 Gbps[5] and 800 Gbps[6] switches on the horizon, which places increased demands on endpoint processing. Secondly, technological limitations, including constraints in power consumption [7] and memory speed [8], have impeded significant improvements in host processing capabilities relative to their networking counterparts.

*Timothy is presently pursuing a doctoral degree at Stanford University while also serving as an intern at Intel Corp.

An analysis presented in [4] delves into the primary sources of host congestion observed in modern-day data centers, encompassing factors such as IOMMU translation costs and LLC cache misses. Our work aims to provide network-protocol level support that addresses congestion resulting from the growing disparity between networking and host processing speeds.

B. The Need for an EC-Aware Scheme

Endpoint congestion leads to an accumulation of incoming packets in the network interface buffer. In a lossy fabric, which is the focus of our work, this eventually forces the target to drop packets. An oblivious initiator continues to transmit packets that are ultimately discarded at the congested endpoint, leading to wasteful network activity. Consequently, mitigating or eliminating packet drops stemming from endpoint congestion yields benefits that extend beyond mere reduction in packet loss. These advantages include:

- *Improved effective network bandwidth*: With fewer drops and packet retries, wasted bandwidth due to retransmissions is minimized, enhancing overall network efficiency.
- *Enhanced end-to-end throughput*: An EC-aware congestion control mechanism has the potential to reclaim idle host cycles that would otherwise be wasted when the buffer is either instantaneously depleted or stalled due to packet loss.
- *Power reduction*: Limiting packet drops and needless retries also results in power savings, a critical factor in the data center.

Although beyond the immediate scope of our work, it is worth noting that an EC-aware scheme also brings significant benefits to lossless fabrics. In such a system, endpoint congestion can propagate back into the network, potentially inducing catastrophic congestion on upstream traffic [9]. For next generation data centers, where endpoint processing is anticipated to become a bottleneck, a networking scheme explicitly designed to address the challenges of endpoint-induced congestion is vital.

C. Existing Schemes Unsuitable for Endpoint Congestion

In this study, our focus is on a medium-scale domain where processor or accelerator nodes are interconnected through a commodity networking fabric to collectively execute a single task [10]. The medium-scale domain refers to a configuration with few switch hops where the round-trip latency is bounded, for instance, to $< 10\mu$s. Arguably, this is the sweet-spot for a scale-out accelerator domain. Conventional networking protocols, such as TCP [11], [12], are designed with large-scale deployments in mind, typically involving tens of thousands

of nodes with several switch hops, and hence not suitable for a medium-scale domain. Furthermore, accelerator attached fabric demands swift recovery and responsiveness, which TCP fails to provide due to its conservative congestion avoidance and slow start phases. Hence, a reactive solution is desired.

Other receiver-driven schemes, such as HOMA[13], pHost[14], and NDP[15], empower targets to proactively throttle senders. However, these methods operate under the assumption that congestions occur within the network infrastructure (i.e. network cores and downlinks). They are still vulnerable to EC. Furthermore, these software-based protocol solutions are primarily designed for networks with downlinks up to 10Gbps. They may present a bottleneck when handling vast volumes of data that arrive at a time in future data centers awaiting processing.

Our contributions in this work are as follows:

- We propose a reactive hardware-based scheme tailored to address endpoint congestion within a medium-scale domain of nodes. To the best of our knowledge, this is the first endeavor to incorporate endpoint-induced congestion considerations into the design of a network protocol.
- Our policy assumes a lossy commodity network like Ethernet, thereby obviating the need for link level credit mechanisms (e.g. RoCEv2 [16]).
- Our approach is proactive in throttling and recovering transmission streams by employing duplicate acknowledgments to regulate the transmission pace of initiators.
- We evaluate the performance of our proposed policy in both unicast and incast PUT (RDMA write) packet scenarios, achieving near-zero packet drops.

II. ACK-based Endpoint congestion policy

This section presents our host-congestion policy. In this work, we define the *initiator* as one that initiates a PUT and the *target* as one that receives the PUT.

A. Reliable Transmission

Our policy builds upon the end-to-end reliability protocol for lossy networks that uses a traditional acknowledgement model. A PUT packet sent from the initiator to the target is acknowledged by the target with an ACK (standalone or piggybacked) that contains the sequence number of the accepted packet. In the absence of packet drops, the initiator increments the sequence number for each outgoing packet, and the target responds with an ACK containing the highest valid sequence number received (that may span multiple packets). If an out-of-order packet is received, it is dropped and a NACK (negative acknowledgement) with the last accepted packet sequence number is sent to the initiator. Initiator-side retransmission can be triggered by either a NACK from the target or a timeout if the ACK or NACK gets dropped by the switching network.

B. Design Goals

We center our EC-aware scheme around four goals:

Minimizing Modification: Our scheme builds upon an ACK infrastructure, avoiding extensive modifications to the baseline end-to-end reliability protocol.

Maximizing Throughput: In the absence of host congestion, our policy will maximize link utilization.

Congestion Prevention: To avert future packet drops, a congestion impending signal based on the processing queue occupancy is proactively sent to the initiator.

Rapid Recovery: Our scheme ensures timely notification to the sender when endpoint congestion subsides, allowing for swift recovery and transmission resumption.

These design objectives underpin our policy development approach, guiding our EC-aware protocol formulation.

C. ACK Types

We designate labels for ACKs used in our policy as follows:

Duplicate ACK (DACK)] is an ACK that contains a sequence number identical to that of a previously sent ACK within the same connection.

Incremented ACK (IACK)] is an ACK that contains a sequence number greater than that of a previously sent ACK within the same connection.

In the TCP protocol, three consecutive duplicate ACKs indicate a packet drop, triggering the initiator to start retransmission. Unlike this, our baseline end-to-end reliability policy uses NACKs to signify packet drops, without generating duplicate ACKs. In our proposed scheme, we incorporate duplicate ACKs based on the target buffer occupancy (§II-D). These signals guide the initiator in adjusting its data transmission rate (§II-E).

D. Target Logic

When the host processing bandwidth exceeds the link bandwidth (no EC), the target queue should be either short or empty. We define the system as congested when:

$$\texttt{system\_congested} = \texttt{target\_queue\_occupancy} > \texttt{policy\_threshold}$$

During congestion periods, we temporarily suppress IACKs that are sent as soon as a packet enters the processing queue.

When a packet enters the system, it is placed in the processing queue. An IACK is sent in two scenarios: when the system is not congested and a packet finishes processing, and when the system is congested, but a packet transitions from above to below the `target_queue_occupancy` threshold position within the processing queue (Fig. 1). This typically occurs after a packet is processed, and a DACK is immediately sent before an IACK to signal congestion. The IACK can be sent as a standalone or as a piggyback, both are considered by the initiator to adjust transmission rate. Additionally, when a packet enters an already congested target, a standalone DACK is instantly sent back to the initiator to indicate congestion and potential packet drop.

Under normal host-congestion-free conditions, IACKs are sent as soon as they are accepted in the system. In this scenario, no DACKs are generated, and the link bandwidth can be fully utilized.

When the queue occupancy exceeds the policy threshold, IACKs contain the sequence number of the packet occupying

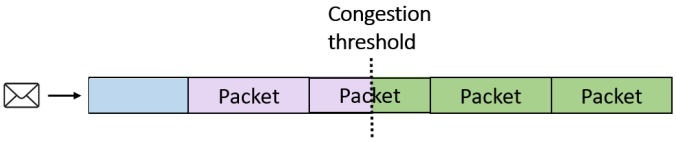


Fig. 1: Target queue ACKs based on buffer occupancy. ACKs dispatched from the target to the initiator carry the highest sequence number of the packet positioned just below the threshold (green).

the space just below the threshold (Fig. 1). In other words, the sent IACKs always carry the highest sequence number of the packets occupying the space below the threshold—not the packets that have been successful received. This is where we differ from traditional schemes.

E. Initiator Logic

The transmission rate of the initiator is steered by an internal state machine, depicted in Fig. 2. Under EC-free conditions, the system remains in the halt state H∞, where the transmission rate is not throttled, allowing back-to-back packet transmission. The packet transmission is governed by a fixed outstanding packet or data size threshold, but the rate itself is otherwise unregulated.

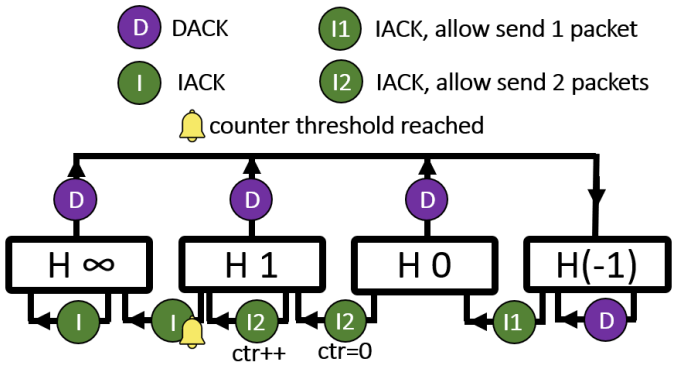


Fig. 2: Initiator internal state transition diagram. Here H∞ denotes the normal non-congested state.

When a DACK is received, the initiator transitions to halt state H(-1), where only one packet can be sent per received IACK. When the target is congested, the initiator receives alternating DACKs and IACKs, oscillating between H(-1) and H(0). With two consecutive IACKs received, it enters state H1, allowing two packets to be sent per IACK.

Upon transitioning from state H0 to H1, an internal counter resets. Within H1, each IACK increments the counter and permits the transmission of two packets. Should the counter reach a predefined threshold during receipt of a new IACK while in H1, the system reverts to the unthrottled state H∞, relinquishing rate regulation. The threshold governs how quickly the initiator returns to unthrottled transmission. Empirically, we chose a threshold value of 6.

The arrival of a DACK signals target congestion, immediately throttling the initiator to send only one packet per IACK. This mechanism allows the target to control the initiator transmission rate during endpoint congestion, and the initiator can swiftly resume unthrottled transmission once the congestion subsides.

F. Reacting to EC-Induced Packet Drop

In the event of packet loss, the target sends a NACK to the initiator. The initiator, upon receiving it, is not allowed to retransmit at an unthrottled rate, as the target's congestion may persist. To maintain pacing of the initiator after the packet drop, the target continues to send DACKs for the packets that have arrived before the packet drop and finished processing. The initiator transmits one packet per DACK during retransmission, avoiding further congestion. After retransmission, the initiator returns to the H(-1) state and resumes normal throttled transmission, as described in Section §II-E.

G. DACK Suppression and ACK Coalescing

Once the initiator enters H(-1), the target can pace the transmission based on the buffer's processing rate. While targets can transmit as many DACKs as desired, the transmission of IACKs is more restricted since a packet needs to be accepted at the target to send an IACK.

Working in the presence of endpoint congestion (EC), our policy allows for controlled rate increase and decrease from the initiator while ensuring reliability and correctness of sequence numbers. By suppressing DACKs, the target can allow the initiator to observe two consecutive IACKs and progress to the H1 state, effectively increasing the transmission rate. Similarly, to decrease the rate, we can manually coalesce two IACKs into one by skipping the first IACK.

H. Fairness Among Multiple Streams

If an initiator is in a halt state (non-H∞), it relies on receiving IACKs from the target in order to send packets. There is however a bias towards streams that already have packets accepted at the target. Using DACK suppression and ACK coalescing, targets can balance individual transmission rates from multiple initiators. Our policy periodically samples packet counts from each stream. If a stream's rate significantly surpasses others, the target performs DACK suppression on the lower-rate stream and ACK coalescing on the higher-rate one. This ensures fairness between data flows.

III. Evaluation

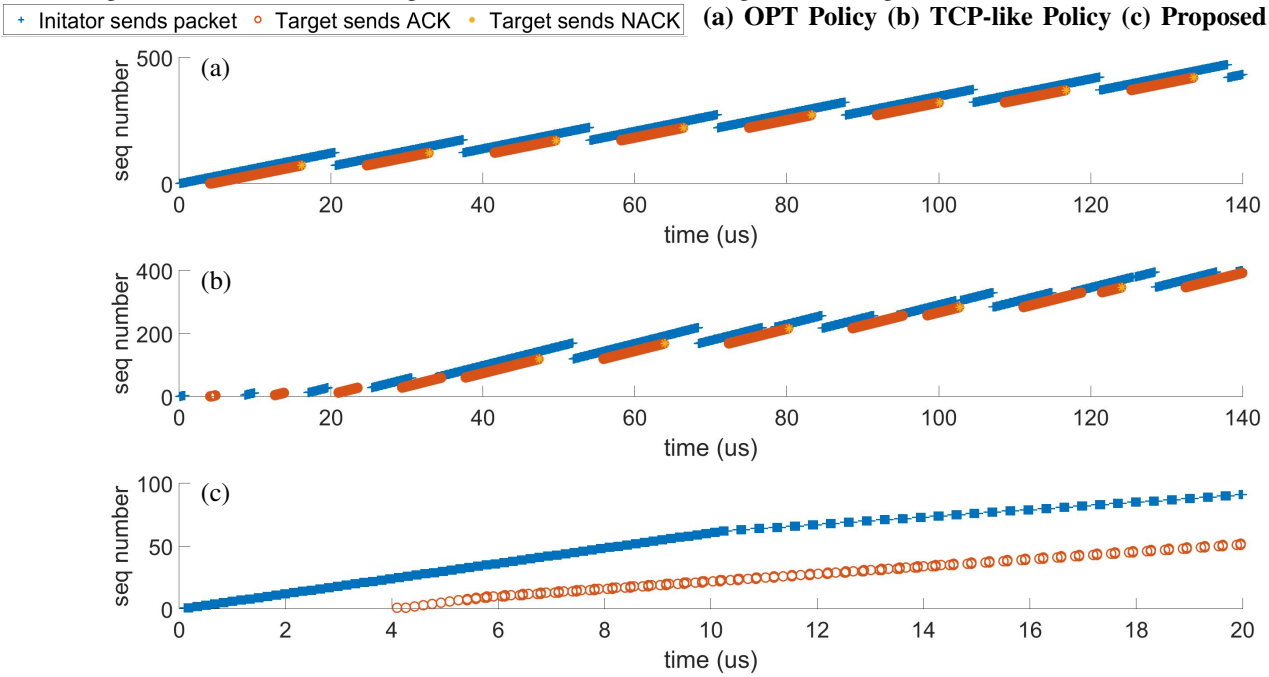
A. Baseline and Proposed Policies

Our evaluation involves two baselines and the proposed policy, with all using explicit NACKs to signal packet loss from target to ensure fair baseline for reactivity.

Outstanding Packet Threshold (OPT): The OPT policy permits unlimited data transmission as long as the total outstanding packets does not surpass a specified threshold for data size and packet count. Retransmissions follow the same threshold, with no transmission rate restriction.

Additive Increase/Multiplicative Decrease (TCP-like): This policy employs a standard additive increase, multiplicative decrease (AIMD) congestion window that governs the amount of outstanding data allowed at any given time [17]. The congestion window grows linearly and shrinks in half depending on NACKs and ACKs that it receives, thereby indirectly controlling the transmission rate.

Fig. 3: Unicast PUT streaming trace. Packet Size: 2KB. Target Processing Bandwidth: 50% of link bandwidth
(a) OPT Policy (b) TCP-like Policy (c) Proposed Policy



Proposed Policy (detailed in §II): Under non-EC conditions, our scheme operates identically to the OPT scheme and follows a fixed outstanding packet threshold.

We set the maximum segment size (MSS) and target queue occupancy threshold at 9KB for our evaluation.

B. Experimental Setup

The three policies, baselines and proposed scheme, were simulated using BookSim [18], a network simulator designed for research in lossless, credit-based interconnection networks. The simulator was modified to simulate a lossy fabric with packet drops at switches and nodes. OPT policy results were validated against hardware results, with hardware-level parameters updated in BookSim.

In our experimental setup, we assumed a 100Gbps network with fully non-blocking fat-tree topology where the round-trip traversal between two endpoints takes approximately $8\mu s$. We implemented a backend interface to support Bale kernels [19] and OpenSHMem[20], which replaces the default traffic manager's packet injection. In this study, we focus on PUT packets, which often cause endpoint congestion when queued for memory writes, and used streaming unicast and incast PUT Bale kernel workload for evaluation. A target queue, implemented on each endpoint, holds incoming packets. We compare our policy to the baseline policies at different queue drain rates as a percentage of the full link bandwidth. We present results for target queue depth of 81KB in this paper.

IV. Results

A. Unicast Trace

Fig. 3 shows unicast streaming PUT traffic trace over time. Each marker signifies a packet event, with the y-value indicating the packet's sequence number.

The OPT policy (Fig. 3a) allows continuous packet streaming until an outstanding threshold is reached, even during retransmission. This leads to packet drops and a cycle of packet drops and retransmissions.

The TCP-like policy (Fig. 3b), on the other hand, dynamically adjusts its window size in response to NACKs from packet drops. Although it transmits packets less aggressively than the OPT policy, it still experiences similar packet-drop-and-retransmission cycles, albeit at a lower drop rate.

In the proposed scheme (Fig. 3c), the target promptly responds to imminent congestion by issuing DACKs when the target queue threshold is reached (around the $6\mu s$ mark). The initiator immediately throttles its transmission rate upon receiving this signal (around the $10\mu s$ mark). Note that only the first $20\mu s$ of the trace is shown compared to the other policies because the system quickly stabilizes to match the host processing bandwidth, resulting in a subsequent steady stream of packets without any packet drops.

B. Unicast PUT Performance

We compare the normalized goodput and packet retransmission rate among the three policies over a range of target processing bandwidths under endpoint congestion, where the target queue is drained at a rate lower than the link bandwidth. All goodput, packet retransmission rates, and endpoint bandwidths are normalized to the maximum link bandwidth. We consider streaming packet data sizes ranging from 512B to 8KB, excluding a 64B header.

Goodput refers to the data rate successfully accepted by the target, excluding the header. An ideal goodput scenario would exhibit a linear relationship with a goodput figure of $y = x$.

Fig. 4 shows the goodput results for all three policies. At first glance, all three policies exhibit goodput performance close to the ideal scenario for a unicast PUT streaming workload. However, with smaller packet sizes of 512B and 1KB, the less-than-ideal results are a result of a combination of the initiator not being able to generate back-to-back packets

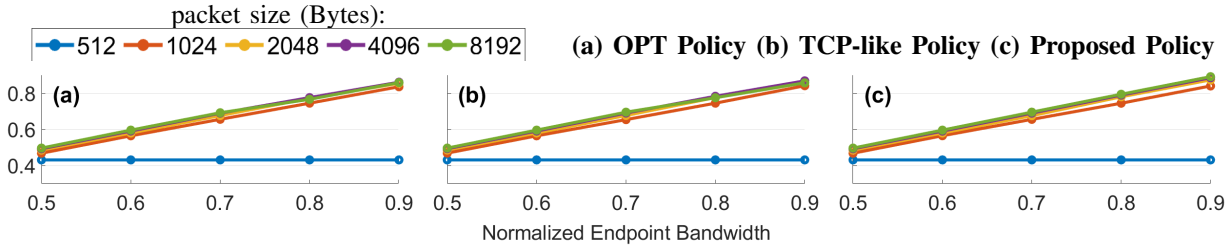


Fig. 4: Normalized Goodput for Unicast streaming PUT workload

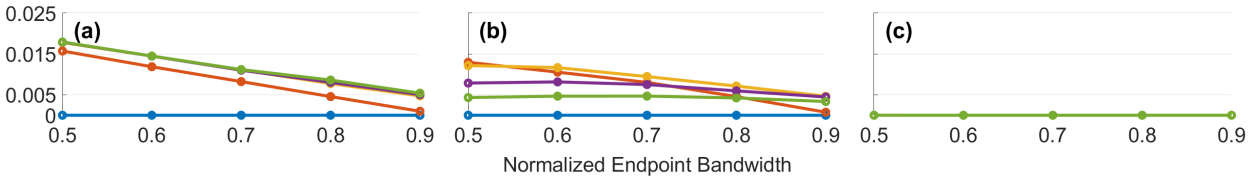


Fig. 5: Normalized Packet retransmission for Unicast streaming PUT workload

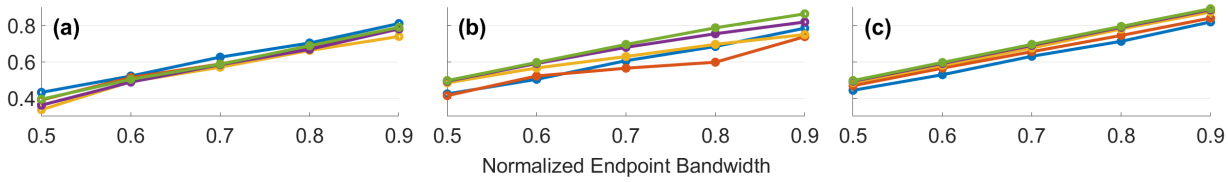


Fig. 6: Normalized Goodput for 8-to-1 incast streaming PUT workload

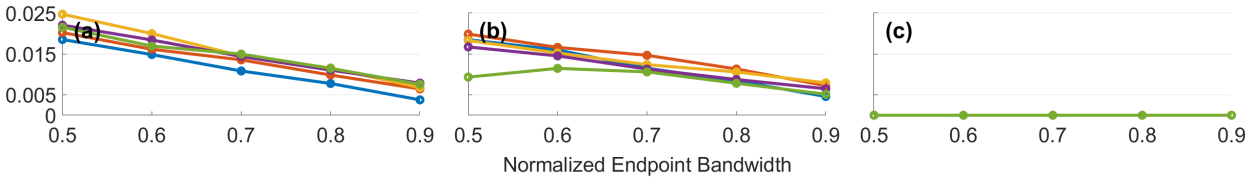


Fig. 7: Normalized Packet retransmission for 8-to-1 streaming PUT workload

due to processing overhead as well as packet header size. Specifically, for 512B packet size, the initiator is unable to transmit packets at the full link bandwidth, regardless of the presence of endpoint congestion.

In both the OPT and TCP-like scenarios, the round-trip time for NACKs to arrive at the initiator and for retried packets to reach their destination is fast enough that the target queue is not fully drained. Consequently, we are able to fully utilize the reduced endpoint throughput, resulting in overall goodput close to the ideal scenario. However, this should not overshadow the substantial drops incurred when initiators using these two policies blindly retransmit without knowledge of target EC. Fig. 5a and 5b illustrate that up to 1.8% of the link bandwidth is wasted on useless work when the endpoint can only accept traffic at 50% of the link bandwidth. Higher endpoint processing bandwidth leads to lower packet drop rates as the target queue drains more quickly.

The TCP-like policy reduces packet retransmission compared to the OPT policy by being less aggressive. Generally, for TCP-like policy, larger packet sizes result in lower packet drop rates because the additive increase nature of the congestion window allows for smaller packets to be streamed out at a higher rate than larger packets, resulting in a higher number of packets being dropped in the presence of endpoint congestion. The lowered packet drop rates for both 512B and 1KB packets can be attributed to the previously mentioned header overhead and initiator processing overhead. These results starkly contrast with the proposed policy, which experiences zero packet drops for the workload, as discussed in Section §IV-A.

C. Incast PUT Performance

Figs. 6 and 7 show the results for an 8-to-1 incast streaming PUT workload, which are generally similar to the unicast case, with our proposed policy demonstrating substantial improvements in reducing packet retransmissions. The gap between the retransmission rates of the OPT and TCP-like policies has narrowed, suggesting that the advantages of the TCP-like policy's conservatism in growing its congestion window diminish as the number of nodes in the system increases in the presence of endpoint congestion. Additionally, the processing overhead for smaller packet sizes has disappeared because each individual initiator no longer needs to send packets back-to-back to saturate target link.

The lower-than-ideal goodput numbers in the proposed policy are due to header overhead, which is more apparent in smaller packets. Despite this, the proposed policy still substantially outperforms baseline policies in terms of packet retransmission rates, enabling more useful network traffic.

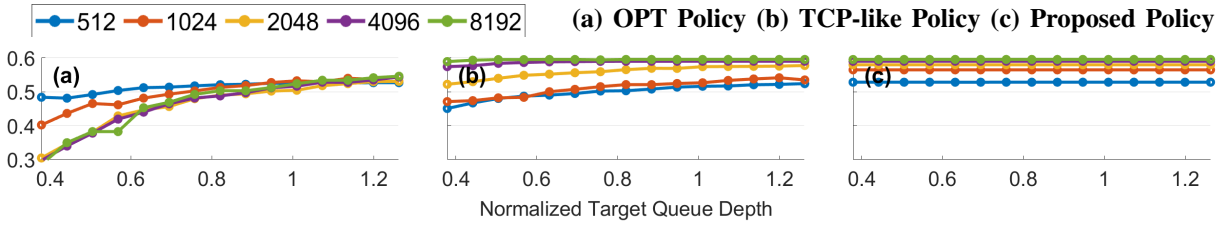


Fig. 8: Normalized Goodput at 70% of link bandwidth for 8-to-1 incast streaming PUT workload

Fig. 9: Normalized Packet Retransmission at 70% of link bandwidth for 8-to-1 incast streaming PUT workload

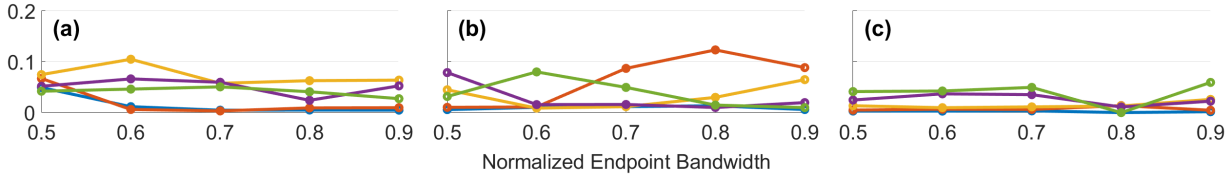


Fig. 10: Maximum Normalized PUT rate difference for 8-to-1 incast streaming PUT workload

D. Varying Queue Depth

Fig. 8 presents the goodput and packet retransmission rate for the 8-to-1 incast PUT workload, where the queue depth is normalized to the bandwidth delay product of a 64B header-only packet (link bandwidth $\times$ round trip time without processing delay). The proposed policy achieves near-maximum goodput even at low queue depths, while the other two policies result in idle host cycles during packet drops when the queue drains temporarily and the host is not performing any useful work, lowering goodput. Additionally, our policy exhibits near-zero packet retransmissions for this workload. It is worth noting that the OPT and TCP-like policies continue to experience packet drops even at high queue depths due to their EC-agnostic nature, as shown in Fig. 9. In fact, the queue depth does not significantly impact packet retransmissions when persistent endpoint congestion is present for this streaming incast workload. This is because the target queue will eventually fill up regardless of the total available space, and blindly retransmitting packets without considering target congestion does not alleviate the issue. Thus, our scheme has also the potential benefit of reducing the buffering requirements at the endpoint and consequently, the cost.

E. Fairness

Fig. 10 shows the maximum put rate difference normalized to maximum link bandwidth between any two initiator streams for the 8-to-1 incast PUT workload across a range of EC conditions. Although our policy strategy has an implicit bias towards streams that already have packets at the target (§II-H), our scheme utilizing DACK suppression and ACK coalescing effectively counteracts this effect and ensures fairness comparable to the baseline policies. Overall, our policy consistently outperforms the baseline policies, particularly at lower endpoint processing bandwidth and smaller packet sizes.

V. Conclusion

We presented an EC-aware policy that tackles the issue of endpoint-induced congestion. Our policy leverages ACKs to incorporate reactive measures to regulate initiator transmits, avoiding packet drops while enabling swift recovery in the context of a medium-scale accelerator domain where the round-trip latency is bounded to a few switch hops. Through simulation, we demonstrated the effectiveness of our policy in reducing packet retransmissions and achieving high throughput across various endpoint congestion scenarios. Additionally, our mechanism ensures fairness, achieving a balanced transmission among initiators compared to baseline approaches. Even with increased buffering, traditional policies fall short of achieving the same performance. Thus our scheme can potentially reduce the cost of endpoint interface.

Future work will evaluate our EC-aware policy for different workloads and packet types, exploring its effectiveness in diverse network configurations, and implementing the policy in hardware for performance analysis and optimization. As network link bandwidth continues to outpace host processing bandwidth, the impact of endpoint congestion is expected to become more pronounced in the near future. Therefore, addressing the challenges posed by endpoint congestion becomes paramount for ensuring optimal network performance and accommodating future scalability requirements in data centers and distributed computing systems.

VI. Acknowledgement

We extend our gratitude to Carl J. Beckmann, Halit Dogan, and Christos Kozyrakis for their help and support throughout this research endeavor.

References

[1] N. Jiang, D. U. Becker, G. Michelogiannakis, and W. J. Dally, "Network congestion avoidance through speculative reservation," in *IEEE International Symposium on High-Performance Comp Architecture*, 2012, pp. 1–12. DOI: 10.1109/HPCA.2012.6169047.

[2] D. Tran and H. Raghavendra, "Congestion adaptive routing in mobile ad hoc networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, no. 11, pp. 1294–1305, 2006. DOI: 10.1109/TPDS.2006.151.

[3] V. Kushwaha and R. shwer, "A review of router based congestion control algorithms," *International Journal of Computer Network and Information Security*, vol. 6, pp. 1–10, Nov. 2013. DOI: 10.5815/ijcnis.2014.01.01.

[4] S. Agarwal, R. Agarwal, B. Montazeri, *et al.*, "Understanding host interconnect congestion," in *Proceedings of the 21st ACM Workshop on Hot Topics in Networks*, ser. HotNets '22, Austin, Texas: Association for Computing Machinery, 2022, pp. 198–204, ISBN: 9781450398992. DOI: 10.1145/3563766.3564110. [Online]. Available: https://doi.org/10.1145/3563766.3564110.

[5] *Cisco Introduces New SaaS Tool and Line of 400G Data Center Switches — datacenterknowledge.com*, https://www.datacenterknowledge.com/cisco/cisco-introduces-new-saas-tool-and-line-400g-data-center-switches, [Accessed 12-Jul-2023].

[6] M. Cooney, *Cisco data-center switches promise 800Gb Ethernet, deliver 400GbE today — networkworld.com*, https://www.networkworld.com/article/3663703/cisco-data-center-switches-promise-800gb-ethernet-deliver-400gbe-today.html, [Accessed 12-Jul-2023].

[7] H. Zhang and H. Hoffmann, "Maximizing performance under a power cap: A comparison of hardware, software, and hybrid techniques," *SIGPLAN Not.*, vol. 51, no. 4, pp. 545–559, Mar. 2016, ISSN: 0362-1340. DOI: 10.1145/2954679.2872375. [Online]. Available: https://doi.org/10.1145/2954679.2872375.

[8] J. Jeddeloh and B. Keeth, "Hybrid memory cube new dram architecture increases density and performance," in *2012 Symposium on VLSI Technology (VLSIT)*, 2012, pp. 87–88. DOI: 10.1109/VLSIT.2012.6242474.

[9] Y. Zhang, Y. Liu, Q. Meng, and F. Ren, "Congestion detection in lossless networks," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, ser. SIGCOMM '21, Virtual Event, USA: Association for Computing Machinery, 2021, pp. 370–383, ISBN: 9781450383837. DOI: 10.1145/3452296.3472899. [Online]. Available: https://doi.org/10.1145/3452296.3472899.

[10] V. Krishnan, O. Serres, and M. Blocksome, "Configurable network protocol accelerator (copa): An integrated networking/accelerator hardware/software framework," in *2020 IEEE Symposium on High-Performance Interconnects (HOTI)*, 2020, pp. 17–24. DOI: 10.1109/HOTI51249.2020.00018.

[11] V. Cerf and R. Kahn, "A protocol for packet network intercommunication," *IEEE Transactions on Communications*, vol. 22, no. 5, pp. 637–648, 1974. DOI: 10.1109/TCOM.1974.1092259.

[12] M. Alizadeh, A. Greenberg, D. A. Maltz, *et al.*, "Data center tcp (dctcp)," in *Proceedings of the ACM SIGCOMM 2010 Conference*, ser. SIGCOMM '10, New Delhi, India: Association for Computing Machinery, 2010, pp. 63–74, ISBN: 9781450302012. DOI: 10.1145/1851182.1851192. [Online]. Available: https://doi.org/10.1145/1851182.1851192.

[13] 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*, ser. SIGCOMM '18, Budapest, Hungary: Association for Computing Machinery, 2018, pp. 221–235, ISBN: 9781450355674. DOI: 10.1145/3230543.3230564. [Online]. Available: https://doi.org/10.1145/3230543.3230564.

[14] 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 *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '15, Heidelberg, Germany: Association for Computing Machinery, 2015, ISBN: 9781450334129. DOI: 10.1145/2716281.2836086. [Online]. Available: https://doi.org/10.1145/2716281.2836086.

[15] M. Handley, C. Raiciu, A. Agache, *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*, ser. SIGCOMM '17, Los Angeles, CA, USA: Association for Computing Machinery, 2017, pp. 29–42, ISBN: 9781450346535. DOI: 10.1145/3098822.3098825. [Online]. Available: https://doi.org/10.1145/3098822.3098825.

[16] T. Hoefler, D. Roweth, K. Underwood, *et al.*, *Datacenter ethernet and rdma: Issues at hyperscale*, 2023. arXiv: 2302.03337 [cs.NI].

[17] *RFC 5681: TCP Congestion Control — rfc-editor.org*, https://www.rfc-editor.org/rfc/rfc5681, [Accessed 14-Jul-2023].

[18] N. Jiang, D. U. Becker, G. Michelogiannakis, *et al.*, "A detailed and flexible cycle-accurate network-on-chip simulator," in *2013 IEEE International Symposium on Performance Analysis of Systems and Software (IS-*

PASS), 2013, pp. 86–96. DOI: 10.1109/ISPASS.2013.6557149.

[19] *Bale/docs/Bale-StGirons-Final.pdf at master · jdevinney/bale — github.com*, https://github.com/jdevinney/bale/blob/master/docs/Bale-StGirons-Final.pdf, [Accessed 12-Jul-2023].

[20] B. Chapman, T. Curtis, S. Pophale, *et al.*, "Introducing openshmem: Shmem for the pgas community," in *Proceedings of the Fourth Conference on Partitioned Global Address Space Programming Model*, ser. PGAS '10, New York, New York, USA: Association for Computing Machinery, 2010, ISBN: 9781450304610. DOI: 10.1145/2020373.2020375. [Online]. Available: https://doi.org/10.1145/2020373.2020375.