Poster: Congesting Distributed AI Workloads within the Host Network

Weijia Shao
Chalmers University of Technology and University of Gothenburg
Gothenburg, Sweden
weijias@chalmers.se

William Eriksson Kling
Chalmers University of Technology and University of Gothenburg
Gothenburg, Sweden
klingw@student.chalmers.se

Simon Gunnarsson
Chalmers University of Technology and University of Gothenburg
Gothenburg, Sweden
simongun@student.chalmers.se

Muoi Tran ✉
Chalmers University of Technology and University of Gothenburg
Gothenburg, Sweden
muoi@chalmers.se

Abstract

The host network has emerged as a major source of performance loss for distributed AI workloads in the data center. We show that an unprivileged tenant can weaponize this fabric, issuing only ordinary memory traffic to congest the shared network paths and slow the training steps of a co-located victim workload. Our tested attack inflates Transformer training by 2.3× and GNN training by 3.7×, without any elevated privilege, exploit, or escape. These results establish the host network as an integral part of the attack surface of multi-tenant AI and call for a deeper study of its mitigation.

CCS Concepts

• **Security and privacy → Denial-of-service attacks**.

Keywords

host network; contention attacks; distributed AI

ACM Reference Format:
Weijia Shao, William Eriksson Kling, Simon Gunnarsson, and Muoi Tran. 2026. Poster: Congesting Distributed AI Workloads within the Host Network. In *Proceedings of the 2026 ACM SIGSAC Conference on Computer and Communications Security (CCS '26), November 15–19, 2026, The Hague, Netherlands.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3830454.3846436

1 Introduction

Today, GPU compute has scaled far faster than the memory and interconnect bandwidth that supply it [4], so data movement now dominates the cost of AI workloads. Much of this movement never leaves the machine — inside a single server, input batches, parameters, and gradients travel between GPUs, host DRAM and network cards across an internal fabric of PCIe links, socket interconnects, and memory controllers, known as the *host network* [10]; see Figure 1. The contention on shared host network paths is known to cause performance loss of distributed AI workloads [1, 2, 5, 6, 8].

In practice, operators amortize costly accelerators by co-locating jobs from many tenants on the same node with little QoS isolation: e.g., PCIe, inter-socket links, and memory controllers are all shared best-effort. Such shared hardware enables deliberate resource contention, e.g., denial of memory service on multi-core CPUs [9]; yet, the AI host network remains unexamined as such a surface.

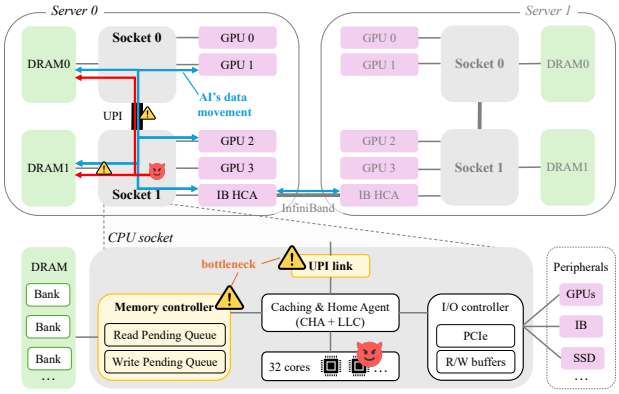

Figure 1: The (inter)-host networks of our tested AI cluster (cf. §4). The AI's data movement shares two bottlenecks: (1) each socket's memory controller [10] and, as we identify, (2) the cross-socket UPI link. An unprivileged attacker pinned to one socket issues memory traffic to local and remote DRAMs, saturating both bottlenecks and slowing the victim's training.

This work demonstrates that an unprivileged attacker can measurably degrade a co-located distributed AI workload simply by issuing ordinary memory traffic along strategically chosen host network paths vulnerable to congestion. Specifically, on a DDP Transformer, the attack inflates the end-to-end step time by 2.3× and the gradient all-reduce collective by 6.1×, saturating the memory controller and inter-socket link on the InfiniBand HCA's DMA path. On a graph neural network (GNN), it causes a 3.7× slowdown, but through CPU-side neighbor sampling instead of the collective.

We argue that the host network is now an integral attack surface of multi-tenant AI instead of just a performance concern, as previously suggested. Several questions thus remain open: how far such congestion attacks reach across the AI stack from training to inference and agentic serving; whether today's multi-tenant platforms can offer meaningful host-network isolation at all; and how far a defender should trust the fabric beneath a rented accelerator.

2 Background

Host network. The host network [10] is the fabric that connects the processors, memory, and peripherals (e.g., accelerators); see an example in Figure 1. In each network, CPU sockets reach their own DRAM through a memory controller, attach peripherals over PCIe, and connect to remote DRAMs or peripherals via UPI links [3]. Inter-server transfer via InfiniBand also traverses this fabric.

This work is licensed under a Creative Commons Attribution 4.0 International License.

CCS '26, The Hague, Netherlands

© 2026 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-2871-6/2026/11
https://doi.org/10.1145/3830454.3846436

Data path bottlenecks. There are two potential bottlenecks in the host network; see the orange boxes in Figure 1. First, the *memory controller* can be a contention point between the reads/writes that CPU cores issue to DRAM and the DMA that a co-located NIC or GPU issues to the same memory [1, 10]. We further identify the *inter-socket link* (e.g., UPI) as another bottleneck, since it may carry both memory access from another socket's cores and peripherals.

AI's data movement. A single AI training step drives a repeating sequence of transfers through the host network; see the blue lines in Figure 1. A training step first loads its input batch from host DRAM into the local GPUs over PCIe (e.g., `DRAM1` → `GPU2`). Data-parallel training then synchronizes gradients across nodes through an all-reduce: NCCL copies gradient chunks from GPU HBM into a pinned region of host DRAM (e.g., `DRAM1` ← `GPU2`), the InfiniBand Host Channel Adapter (HCA) reads that region over PCIe by DMA (e.g., `DRAM1` → `IB HCA`), and the HCA sends the gradient to the peer node (e.g., *Server 0* to *Server 1*) before the reduction returns along the reverse path. When the pinned buffer and the HCA sit on different sockets, this DMA (e.g., `DRAM0` → `IB HCA`) crosses the UPI link. A GPU-to-GPU transfer that finds no NVLink falls back to PCIe, and one between GPUs on different sockets (e.g., `GPU1` ↔ `GPU2`) crosses the UPI link as well.

3 Congestion Attacks

Threat model. We consider a multi-tenant HPC cluster with mutually untrusting users. The adversary is an unprivileged tenant who submits jobs and requests resources through the batch scheduler. The goal is to degrade a co-located distributed AI training job. We assume the adversary knows the host's static layout, including its own socket and the attachment of each GPU and IB HCA, from the OS and public vendor documentation.

Congesting the memory controller and UPI link. The attacker spawns one thread per core it controls, each streams read-modify-write requests over a private 64 MB buffer.[1] Such a request reads a byte, inverts its bits, and writes it back, thus involving both a read and a write to memory. If the attacker places the requests on the local socket (e.g., Socket 1 in Figure 1), the stream saturates that socket's memory controller and contends with victim workloads involving peripherals of the same socket (e.g., `DRAM1` ↔ `GPU2`). If the attack places the requests on the remote socket (e.g., Socket 0), they instead cross the UPI link and the remote memory controller, thus affecting both the victim's cross-socket traffic (e.g., `DRAM0` → `IB HCA`) as well as local traffic (e.g., `DRAM0` ↔ `GPU1`).

4 Evaluation

Research questions. We evaluate the congestion attack in a controlled testbed, aiming to answer the following research questions.

- RQ1 **Feasibility**: Does an unprivileged co-tenant measurably degrade a co-located training job at all?
- RQ2 **Effectiveness**: How large is the slowdown, which bottleneck delivers it, and how few attacker resources does it take?
- RQ3 **Generality**: Across which workloads and scales does it hold, and why do some resist more than others?

[1] 64 MB is larger than the LLC so that every access reaches DRAM instead of the cache.

Experimental setup. We run all experiments on a cluster of two servers, each equipped with 4 NVIDIA A40 GPUs and a dual-socket Xeon 6338 processor with 64 cores, under the Slurm scheduler (cf. Figure 1). In each server, each socket hosts 32 cores and 2 GPUs, while Socket 1 additionally hosts IB HCAs. We repeat each experiment five times and report the averaged results.

Metric. We report the *slowdown*, the ratio of a training step's latency under attack to that in an identical baseline run. We report slowdown for the whole step and for each phase within it. [2] Each GPU worker also logs its own timing, which separates a worker's own slowdown from waiting time on a slower peer. To explain where a slowdown originates, a separate monitor samples the host network every 100 ms for memory bandwidth, memory controller queue occupancy, and interconnect utilization.[3]

Attacker setup. To test the attack effectiveness, we experiment with attacker cores ranging from 1 to 32, all confined to a single socket. We also place the attacker's buffers either on its local socket, which loads that socket's memory controller, or on the remote socket with `numactl`, which drives traffic across the UPI link and the remote controller. To ensure the attacker reaches steady-state saturation before the first training step, we gate its start against the victim's launch with a ready/go handshake.

Victim setup. We deploy victim workloads in both *intra-node* configuration, in which all workers and the attacker are on one server, and *inter-node* configuration, in which workers are spread across two hosts connected over InfiniBand. In both, the collective uses the ring all-reduce algorithm, and each worker pins its cores and memory to its GPU's local socket before NCCL initialization.

We evaluate two workloads chosen to differ in memory access pattern and gradient size. The *Transformer* is a six-layer encoder trained on next-token prediction over synthetic sequences resident in host DRAM (batch size 16, sequence length 512). The *GNN* is a three-layer GCN for node classification, drawing 1024-node mini-batches by 2-hop neighbor sampling ([15, 10] fanout) from a DRAM-resident graph on CPU worker processes. We also sweep one width parameter per workload to vary gradient size.

5 Results

Congestion attacks are feasible (RQ1) and effective (RQ2). An unprivileged co-tenant can indeed slow a co-located training job by up to 3.7×, and does so with modest resources. Figure 2 reports the end-to-end slowdown for the GNN ($d_{\text{hidden}} = 512$) and the Transformer ($d_{\text{model}} = 1024$). The slowdown remains negligible up to 4 attacker cores, then rises sharply at 8, 16, and 32 cores. The GNN is far more sensitive than the Transformer, slowing by 3.4× against 1.3× in the intra-node setting. Crucially, due to synchronous all-reduce, the same attack causes greater slowdown for inter-node than intra-node, raising the Transformer's training time from 1.3× to 2.3× and the GNN's from 3.4× to 3.7×. The attacks against the memory controller and the UPI link have comparable impact in most configurations. The exception is a UPI attack from

[2] We disable DDP fusion to time backward and all-reduce separately.

[3] We source these from InfiniBand port counters (via sysfs) and uncore PMU events. We calculate the memory controller queuing latency by dividing the queue occupancy by the number of inserts, following Little's law [7].

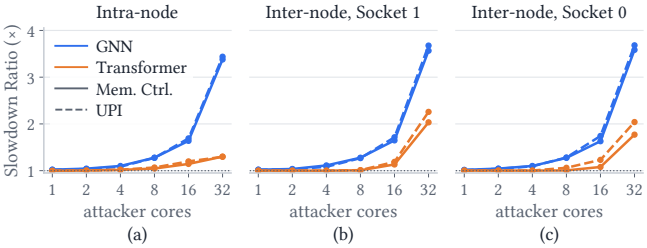

Figure 2: End-to-end slowdown vs. attacker core count (1–32, $\log_2$), under memory controller and UPI attackers. (a) Intra-node, 4 GPUs over PCIe. (b) Inter-node, 8 GPUs over InfiniBand, attacker on Socket 1. (c) As (b), attacker on Socket 0.

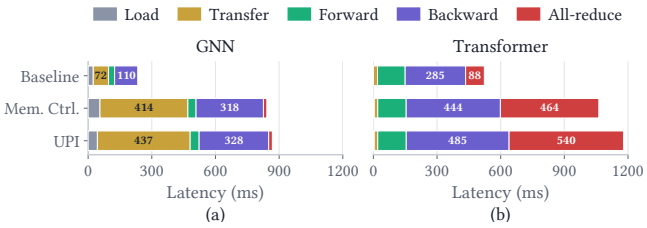

Figure 3: Per-stage mean iteration time (ms) under 32-core attackers on Socket 1.

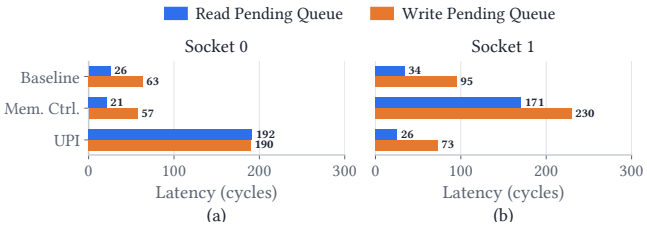

Figure 4: Per-socket read/write pending-queue (RPQ/WPQ) latency for the Transformer, 32-core attacker on Socket 1.

Socket 0 while the IB HCA is on Socket 1, as for the Transformer in Figure 2(c). There the UPI attack outweighs a memory controller attack, because it congests both the inter-GPU all-reduce and the HCA-to-DRAM DMA rather than the all-reduce alone.

Figure 3 attributes the slowdown to individual stages. The backward pass inflates under both workloads, by 3× for the GNN and 1.7× for the Transformer. The workloads diverge in their dominant stage, the GNN's slowdown falling on the transfer stage (6×) that streams each subgraph from DRAM to GPU HBM, and the Transformer's on the all-reduce stage (6.1×). This explains why a congestion attack against UPI impacts the all-reduce-led Transformer more than the transfer-led GNN.

Figure 4 confirms which physical resource is under contention. Under the memory attack, RPQ and WPQ latency inflate sharply on the attacker's own Socket 1, while Socket 0 stays near baseline. Under the UPI attack, the pattern inverts. Socket 0 is loaded even though the attacker's cores run on Socket 1, because its allocations are forced across the interconnect onto Socket 0's controller.

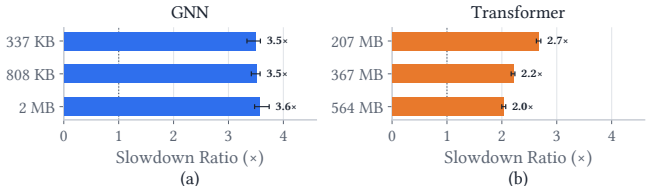

Figure 5: Slowdown vs. model width under a fixed 32-core memory attack on Socket 1, inter-node. Transformer $d_{\text{model}} \in \{512, 768, 1024\}$, GNN $d_{\text{hidden}} \in \{128, 256, 512\}$.

Congestion attacks are general (RQ3). The attack holds across both victim workloads and their model widths. Figure 5 sweeps the model width under a fixed memory attack. The GNN's slowdown stays ≈3.5× across all three hidden dimensions. The Transformer's decreases with model size from 2.7× to 2.0×, possibly because its baseline communication grows faster than the attacker's added latency, yet even the smallest Transformer tested is slowed by 2.7×.

6 Conclusion

We demonstrate an unprivileged congestion attack against distributed AI workloads in the host network of a multi-tenant cluster. We plan to test it in larger clusters and different platforms. Our study targets the memory controller and the UPI link, yet an NVLink fabric may expose a further shared path to a co-located attacker. It also covers only training, and the attack may extend across the AI stack to inference and agentic serving. We leave mitigation to future work, as the attacker's memory traffic is indistinguishable from that of a legitimately memory-intensive tenant and thus hard to defend against. Moving forward, we call for treating the host network as an integral part of the attack surface of multi-tenant AI.

Acknowledgments

We would like to thank our reviewers for their insightful feedback and suggestions. This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

References

[1] Saksham Agarwal et al. 2022. Understanding Host Interconnect Congestion. In *Proc. ACM HotNets.*
[2] Qizhe Cai, Shubham Chaudhary, Midhul Vuppalapati, Jaehyun Hwang, and Rachit Agarwal. 2021. Understanding Host Network Stack Overheads. In *Proc. ACM SIGCOMM.*
[3] David L Mulnix. 2022. *Intel Xeon Processor Scalable Family Technical Overview.*
[4] Amir Gholami, Zhewei Yao, Sehoon Kim, Coleman Hooper, Michael W Mahoney, and Kurt Keutzer. 2024. AI and Memory Wall. *IEEE Micro.*
[5] Raj Joshi, Saksham Agarwal, ChonLam Lao, and Minlan Yu. 2025. Your Network Doesn't End at the NIC: A Case for Unifying the Inter-Host and Intra-Host Networks in (AI) Datacenters. In *Proc. HotNets.*
[6] Qiang Li et al. 2023. More than Capacity: Performance-oriented Evolution of Pangu in Alibaba. In *Proc. USENIX FAST 23.*
[7] John DC Little and Stephen C Graves. 2008. Little's law. In *Building intuition: insights from basic operations management models and principles.* Springer.
[8] Kefei Liu, Zhuo Jiang, Jiao Zhang, Haoran Wei, Xiaolong Zhong, Lizhuang Tan, Tian Pan, and Tao Huang. 2023. Hostping: Diagnosing Intra-host Network Bottlenecks in RDMA Servers. In *Proc. USENIX NSDI.*
[9] Thomas Moscibroda and Onur Mutlu. 2007. Memory Performance Attacks: Denial of Memory Service in Multi-Core Systems. In *Proc. USENIX Security.*
[10] Midhul Vuppalapati, Saksham Agarwal, Henry Schuh, Baris Kasikci, Arvind Krishnamurthy, and Rachit Agarwal. 2024. Understanding the Host Network. In *Proc. ACM SIGCOMM.*