ASPAN: DIAGNOSING FAULTS IN DISTRIBUTED LLM TRAINING FROM PACKET TIMING ALONE

Bekmukhamed Tursunbayev [1]

ABSTRACT

As large language models scale to tens of thousands of GPUs, communication bottlenecks both training and inference. Diagnosing a fault becomes costly, because instrumenting the job is intrusive and too heavy to run always-on; per-rank agents flag a slow machine but not the degraded link; reading the fabric requires control of the network and conflates network with compute; and the coarse signals are too aggregated to localize the fault. Yet a collective already sends its packets on a fixed, known schedule, so the information needed is already present. We present Aspan, a passive, always-on system that makes a distributed job's execution visible from the outside: by turning the job's own traffic into the measurement, it detects, localizes, and grades faults. On hardware, Aspan separates a compute straggler from a congested link where throughput cannot, names the congested link in every single-link flood, and recovers the magnitude of a remote rank's added compute, with no instrumentation of the job, the network, or the collective library. In a Monte-Carlo simulation of the forward model, inverting the full schedule recovers per-rank compute and per-link delay across the topology. Reframing execution observability from a problem of instrumentation into one of inference, Aspan makes the faults that waste GPU-scale compute diagnosable without touching the job, wherever collective communication runs a known schedule.

1 INTRODUCTION

A modern large language model is trained by tens of thousands of GPUs that step forward together. They communicate through collectives at every step: gradients under data parallelism, activations under tensor and pipeline parallelism (Gangidi et al., 2024). At that scale communication takes a large fraction of each step's time, and any component that falls behind stalls the whole job. Such faults are common. A 54-day snapshot of Llama 3 pre-training logged a failure roughly every three hours across 16,384 GPUs (Grattafiori et al., 2024). A study of production clusters found stragglers in 42% of jobs, wasting a tenth of GPU-hours (Lin et al., 2025). On one 4,096-GPU run, fail-slow degradation cost 23,758 GPU-hours, seven percent of its compute (Zhou et al., 2026). At the price of a GPU-hour, each such fault is large and recurring.

When a run slows, the difficulty is not the slowdown but its cause. The answer lives inside the execution: how long each GPU spent computing, and how long each transfer spent on the wire. A single throughput number reports the slowdown, not its cause. An operator needs an account of where the time went, rank by rank and link by link.

[1]Illinois Institute of Technology, Chicago, IL, USA. Correspondence to: Bekmukhamed Tursunbayev <btursunbayev@hawk.illinoistech.edu>.

That account is hard to obtain. The execution is not exposed from outside the job, and every way to expose it carries a cost. Instrumenting the job or its collective library gives the most detailed view, and production systems do so (Zhou et al., 2026; Deng et al., 2025a). The price is a change to every job and enough overhead that the heaviest tracing cannot run always-on. Per-rank agents instead correlate host-level signals across ranks and do run always-on (Deng et al., 2025b; NVIDIA, 2026a), but they read host state, not the collective's timing, so they name a slow machine without naming the link or the amount. Reading the fabric, the switched interconnect, avoids touching the job. Switch-side mirroring reconstructs a run's timeline to within a fraction of a percent (Jiang et al., 2025), but it needs the network's cooperation and sees the traffic, not the compute behind it. The signals that need neither, aggregate step time and transport counters, run everywhere at no cost, yet stay too coarse to say which component is at fault. Each is a sensible choice for what it optimizes, and systems at this scale invest heavily in these tools (Jiang et al., 2024; Zhou et al., 2026). None sees inside a running job from the endpoint alone, without instrumenting it or the fabric.

Yet the information those methods work to collect is already present in the traffic the job sends. A collective runs a fixed, known schedule. For a ring all-reduce, every GPU sends and receives chunks in a set order and size, repeated each step (Patarasuk & Yuan, 2009). Because the schedule fixes when each packet should arrive, the times at which

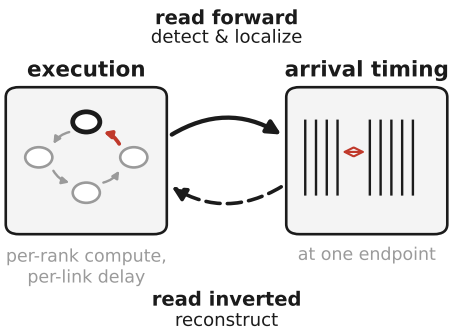


Figure 1. **One map, read two ways.** The schedule maps the execution, the compute at each rank and the delay on each link, to the arrival timing at one endpoint. Read forward, it detects and localizes a fault (§5); read inverted, it reconstructs the execution (§6).

packets do arrive measure the execution that produced them. Recovering that execution is then not a matter of collecting more data but of inference: reading a known signal to solve for the state that shaped it.

We present Aspan, a passive system that computes this inference from the packet timestamps a network adapter already records. It runs at each endpoint, with no change to the job, the network, or the collective library. Aspan treats the schedule as a forward model and reads it in two directions (Figure 1): forward to detect and localize a fault without instrumenting the job and always-on, and inverted to reconstruct the per-component execution behind it.

On a hardware testbed running GPU collectives over RDMA over Converged Ethernet (RoCE), Aspan recovers the magnitude of a remote rank's compute and names the congested link in every single-link flood. Monte-Carlo simulation extends the same reconstruction to per-link delay and per-rank compute across the topology and characterizes how its accuracy grows with the number of endpoints, and packet-level simulation checks the detection signature as the job grows.

This paper makes four contributions.

- **A forward model of collective timing.** We cast execution observability as an inverse problem over a collective's known schedule, and study in simulation what a given set of endpoints can recover (§3, §6.3).
- **Detection and localization.** Aspan identifies a fault, names the degraded link, and flags it while its severity is still low, all from the endpoint (§5).
- **Execution reconstruction.** From endpoint timing alone, Aspan recovers the magnitude of a remote rank's compute on hardware, and per-rank compute and per-link delay in simulation, the quantities that say where a job's time goes (§6).
- **Generality.** The signature holds across collective types, algorithms, and job sizes, on hardware for all-reduce and in packet-level simulation for the rest (§7).

Aspan gives an operator a way to see inside a running job that until now required instrumenting it. Its principle, that a known, repeated communication schedule is enough to infer the execution from its timing, applies wherever computation communicates on such a schedule.

2 Motivation

2.1 Faults at scale

A large distributed job faces many kinds of performance fault. A GPU falls behind, a link carries data slowly, a switch queues packets under load, a pipeline stage stalls, or a machine degrades without failing (Gunawi et al., 2018; Lin et al., 2025). Each is a different problem and each demands a different response: reroute traffic, repair a port, migrate work off a machine, or change the collective's algorithm. Yet every one surfaces the same way, as a run that has grown slower. The symptom alone never says which response to reach for. These faults are frequent, and because every rank waits on the slowest, their waste climbs with the size of the cluster (Zhou et al., 2026).

2.2 Existing approaches

Operators already watch a running job through four kinds of signal, and production clusters invest heavily in all of them (Jiang et al., 2024; Zhou et al., 2026). Two instrument the job, one reads the fabric, and one reads only coarse always-on counters. Each is strong for what it was built for, and each makes a trade-off that leaves the per-component execution unobserved from the endpoint.

In-job profilers. Tracing inside the job or its collective library gives the most detailed view of per-kernel and per-transfer behavior (NVIDIA, 2026c;b), and it is the tool of choice for debugging a known problem. Its cost is the trade-off: full tracing perturbs the job and is too heavy to leave on (Zhou et al., 2026), and its volume grows with the job, too large to run on every job (Deng et al., 2025a).

Per-rank host agents. Agents that correlate machine-level signals across ranks run always-on and flag a faulty machine reliably (Deng et al., 2025b; NVIDIA, 2026a). They read host-level state rather than the collective's own timing, and can name a slow rank without naming the degraded link or the size of the slowdown.

Fabric telemetry. Reading the network directly, through in-band telemetry (P4.org Applications Working Group, 2020) or switch-side mirroring, reconstructs a run's timeline accurately. LLMPrism does this to within a fraction of a

Table 1. **Where Aspan sits.** Whether each approach localizes the faulty component, recovers per-component execution state, can run always-on, and needs no instrumentation of the job or fabric. Reconstruct means per-component compute and delay, not a timeline. Localization differs in grain: per-rank agents resolve a machine, while fabric telemetry and Aspan resolve the link or stage. Aspan's reconstruction (†) is a single-endpoint magnitude on hardware and a full per-link map in simulation. Its always-on capture (‡) adds a per-step host cost that scales with step time and that a data-processing-unit (DPU) offload would remove but that we have not measured off-host.

Approach	Localize	Reconstruct	Always-on	No instr.
In-job profilers	✓	✓	✗	✗
Per-rank agents	✓	✗	✓	✓
Fabric telemetry	✓	✗	✓	✗
Coarse counters	✗	✗	✓	✓
Aspan	✓	✓†	✓‡	✓

percent (Jiang et al., 2025). The trade-off is access and vantage: it needs cooperation from the network operator, which the job's owner usually does not have, and even then it sees the traffic on the wire, not the compute behind it, and reports that a transfer was slow without saying why. At scale it also samples what it mirrors, and misses what it does not.

Coarse always-on counters. Aggregate step time and transport retransmit counts run everywhere at no cost, and every deployment watches them. The trade-off is resolution: they are too coarse to catch a subtle fault, and they attribute none. Step time crosses a threshold only when a slowdown exceeds its normal run-to-run variance, leaving a subtle degradation unseen, and the retransmit counter stays flat whenever congestion control paces a link rather than dropping packets. Even when a slowdown is large enough to register, they report that a run is slow, not what slowed it.

Each is a sensible choice for what it optimizes, and each is widely deployed. None does all of what diagnosis needs at once: localize the fault, reconstruct the execution behind it, the per-rank compute and per-link delay that show where a job's time went, and run always-on, all from the endpoint and without instrumenting the job or the fabric (Table 1).

2.3 The opportunity in the schedule

The signal that names the cause is already in the traffic. A collective's schedule is fixed and repeats every step, and it sets when each packet should arrive. Read against that schedule, the arrival timing is a measurement of the execution (Patarasuk & Yuan, 2009). The rest of the paper turns that observation into a working system: §3 makes the map from schedule to timing precise and asks what it can recover, and §4 shows how to read the signal passively.

3 THE FORWARD MODEL

A collective repeats a fixed schedule that makes a packet's arrival time a known function of two hidden vectors: the per-rank compute c and the per-link delay d. In a ring all-reduce of N ranks, a chunk that reaches rank R after k steps has crossed k links and been reduced at $k-1$ ranks. Its arrival time is

$$t(R,k) = T_0 + \sum_{i=0}^{k-1} d_{(R-k+i) \bmod N} + \sum_{i=1}^{k-1} c_{(R-k+i) \bmod N} \tag{1}$$

where T_0 is the schedule-fixed start of the step. Each arrival is one linear equation in the delays and compute times along the path the chunk took. The method reads this map two ways.

Read forward, a fault perturbs the arrival pattern in a way that is characteristic of its type. A compute straggler starts its burst late but keeps the burst's shape: it delays the whole step without distorting the packet spacing. A congested link instead spreads the arrivals out, as the link paces packets and stretches the gaps between them. The within-window shape therefore separates a congested link from healthy traffic and from a straggler; the straggler shows instead in the cadence between bursts, not the shape within them (§5).

Read inverted, the arrival equations across all endpoints and steps stack into a linear system $t = A\,(d, c) + \varepsilon$, where the schedule fixes the matrix A and ε is the timestamp noise. For a ring, A has about $N(N-1)$ rows and $2N$ unknowns, overdetermined once $N \geq 4$, and least squares recovers (d, c) from the timestamps. The hidden compute and delay become recoverable, not merely the presence of a fault (§6). What one endpoint recovers is limited by coverage. A single endpoint fixes only aggregate magnitudes around the ring, a severity rather than which rank is responsible. Full coverage recovers the whole state, and the error falls as more ranks are observed, a trend that §6 confirms in simulation.

4 DESIGN

Aspan has three parts and one deployment parameter: a passive capture path, a per-window reduction to a few statistics, a read-out that detects, classifies, and inverts, and a receive-queue depth that capture depends on.

Capture. A receiver at each endpoint reads packets through the adapter's user-space interface and takes the hardware timestamp on arrival (Tursunbayev, 2026). It never enters the kernel path and never reads the payload, and observes timing without touching the job or the transport. The capture takes every packet the adapter receives, including traffic of other flows. At one endpoint a healthy Llama-1B run delivers on the order of 90M packets over a ∼180-second

trial, about 0.5 Mpps on average, arriving in dense per-step bursts with idle gaps between them.

Statistics. We reduce the stream in 5-second windows binned at 1 μs, dropping any window below 0.1 Mpps as an idle gap between bursts. Five seconds is long enough to hold a full step's burst and enough packets for a stable inter-arrival distribution. Each window is summarized by a few numbers from that distribution. Detection uses two: the entropy H (how spread out or bursty the arrivals are) and the p99.9 tail (the longest gaps). These capture how a fault distorts timing relative to the collective's healthy profile. A congested link spreads the arrivals and raises the entropy. The exception is all_gather, where that same congestion paces its startup burst into a steady stream and shrinks the tail instead. A stall or pipeline bubble opens a rare long gap that lifts the tail. Which way a number moves is set by the collective and the fault. The direction itself helps name the fault. Classification adds two more, the packet rate and the burst fraction, the fraction of inter-arrival intervals shorter than one microsecond. A handful of numbers per window keeps the observable small enough to run continuously.

Read-out. A healthy range, calibrated per collective, raises an alarm when a number moves outside it, above or below, a classifier names the fault class, and inverting the schedule reconstructs the execution state. We calibrate the detector on the first half of a healthy run, setting its threshold at 1.1 times the healthy maximum, and verify that it raises no alarm on the held-out second half. An alarm then means a real deviation from the schedule.

Receive-queue sizing. One parameter decides whether passive capture works at all: the receive-queue depth. Because the traffic is bursty, a queue sized for the average rate cannot keep up when a per-step burst arrives, and the receiver falls behind and stops capturing mid-run: at 512 work-queue entries capture stalls on real training, while 4096 absorbs the bursts and captures the full run. This is the single tuning the deployment requires.

5 Evaluation

The evaluation answers five questions. Can Aspan (i) separate a compute fault from a network fault at matched throughput (§5.1), (ii) classify and localize the congested link (§5.2), (iii) stay specific to faults rather than to load (§5.3), (iv) hold up against the tools an operator already runs (§5.4), and (v) trace one fault end to end as an operator would (§5.5)? Every result is scoped to what the four-node testbed shows. Because a fault on a link or rank perturbs the arrival pattern at the endpoints whose traffic crosses it, the detection mechanism is present at any number of nodes. Node count affects only the accuracy of the inverse (§6.3). The packet-level simulation reproduces the congestion signature at the testbed's scale before it is applied at larger N (§7).

Setup. The testbed is four nodes on the FABRIC testbed (Baldin et al., 2019), each with one NVIDIA T4 GPU and one ConnectX-5 adapter, joined by a single 25 GbE switch running RoCEv2 with DCQCN (Data Center Quantized Congestion Notification) (Zhu et al., 2015). The workloads are Llama-3.2-1B distributed data-parallel (DDP) training and a synthetic all-reduce loop. We inject four faults: congestion, incast, a compute straggler (induced by both a sleep stall and a power cap), and packet loss, whose injection the fabric's counters did not confirm. A sequence-length sweep serves as a benign control. Results pool 914 trials, each captured at every endpoint. We evaluate every detector under cross-validation grouped by injected condition, and calibrate its alarm band on healthy traffic separate from the trials it is scored on. The baselines are a step-time monitor debounced over three consecutive windows, the adapter's transport and congestion counters, and a per-rank GPU agent, the signals an operator already has.

5.1 Separating compute from network faults

Endpoint timing tells a compute straggler from a congested link even when the two produce the same throughput drop. At a matched drop of about a quarter, the straggler leaves the timing shape flat, its entropy within the healthy band, while congestion lifts the entropy outside that band (Figure 2). A second injector, a power cap that throttles the GPU clock instead of sleeping, produces the same flat-entropy signature. The separation follows the fault mechanism, not one particular way of inducing it. The timing carries what an aggregate metric conflates, and a throughput number alone cannot. Table 1 places this capability against the instrumented tools, and §9 notes that we compare those by what each can observe and the tools our testbed can run by a measured head-to-head (§5.4).

5.2 Classifying and localizing

Detection and attribution run as two steps. The zero-false-alarm band of §4 fires first, and only a flagged window reaches the classifier, a random forest over the window statistics that names the fault. Scored standalone on all windows, the harder three-way problem of healthy, congestion, and straggler before that gate, the forest reaches 0.872 accuracy and 0.944 area under the ROC curve (AUC) for detecting a fault against healthy under group-blocked five-fold cross-validation, with per-class recall 1.00 for congestion, 0.89 for the straggler, and 0.75 for healthy. The 0.75 is the price of three-way attribution, not a deployed false-alarm rate: the detector removes the healthy windows, and the classifier only separates the fault types. The straggler is the harder class: its entropy sits inside the healthy band (§5.1),

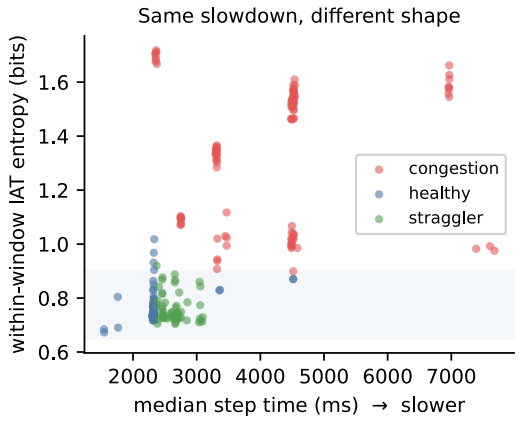


Figure 2. **At a matched throughput drop, timing shape separates the faults.** Entropy against step time for healthy, straggler, and congestion trials. The straggler overlaps healthy in entropy while running as slow as congestion; congestion sits above both.

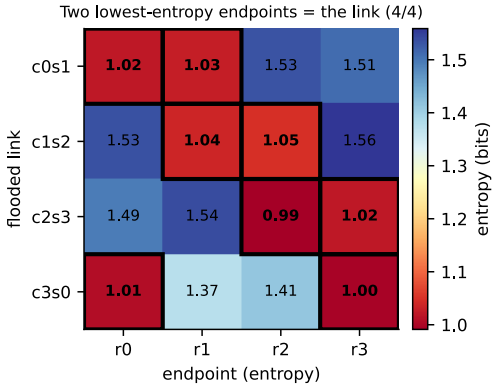


Figure 3. **The congested link's endpoints report the lowest entropy.** Rows are the four flooded links, columns are the endpoints. In each row, the two darkest cells are the link's own endpoints, which localizes the link without switch access.

and the classifier leans on packet rate and burst structure to catch it. The entropy and rate of the window carry most of the signal.

Localization then follows from how the signal is distributed across endpoints. Both endpoints of a congested link report the lowest entropy. Under a flood every endpoint's entropy rises above the healthy band, and the flooded link's own endpoints rise least: the steady flood fills their inter-arrival gaps, while the other endpoints see only the disrupted collective. That pattern names the congested link in all four single-link floods and the victim in all four incasts (Figure 3). On this single-switch testbed a link resolves to its endpoint pair, and a multi-hop fabric is covered only in simulation (§7). Packet loss is the one fault class the timing does not carry. Under leak-free, group-blocked cross-validation, loss reads as congestion most of the time, and we report it as a limitation rather than claim it.

5.3 Specificity

A detector is useful only if it fires on faults and not on ordinary changes in the workload. Raising the sequence length from 128 to 1024, a benign change that alters the traffic volume, moves the entropy from 0.69 to 0.84 bits, against 1.27 under congestion, and at a matched configuration benign and healthy agree to within 0.01 bit (0.760 against 0.759). This signal is specific to faults, not to load, the false-positive control the alarm depends on. The control exercises the entropy and tail; the rate and burst features that catch the straggler are not tested against benign load.

5.4 Comparison with existing tools

We measure how well the tools an operator can run on our testbed detect faults. We score step time, the adapter's transport and congestion counters, and the NVIDIA Data Center GPU Manager (DCGM) per-rank agent (NVIDIA, 2026a) through the same detector on the same faults as Aspan. Pooled over the three classes of §5.2, healthy, congestion, and straggler, step time detects at 0.91 AUC, Aspan and the counters at 0.82, and the agent at 0.69.

The tools fall behind on a congestion flood that costs less throughput than a healthy run's step time naturally varies by, p99 17.3 s on a 2.3 s median, and that DCQCN absorbs without dropping packets (Figure 4). A debounced step-time monitor never trips, the explicit-congestion-notification (ECN) mark counter sits at chance per window (AUC 0.50), and the congestion-notification and retransmit counters rise 6 to 20 times in aggregate but overlap healthy traffic too much to threshold (AUC at most 0.58). The entropy and tail separate the same flood at 0.94 and flag it within the first 5-second window after onset with zero false alarm before it.

5.5 Case study: diagnosing a slowdown end to end

This is a walkthrough, not a new experiment. The separation of §5.1 and the localization of §5.2 were measured as separate controlled studies. Here both read-outs come from a single passive capture of one fault, with no extra instrumentation and no reconfiguration between the two answers. On the four-node testbed, a competing flow floods the link between rank 1 and rank 2 while a collective runs. The symptom is generic: the job slows and step time drifts up within its normal band, and the always-on counters reveal nothing about the cause. The endpoint read first tells apart the two things a slowdown could be: the entropy sits outside the healthy band rather than flat, marking a network fault rather than a compute straggler (§5.1, Figure 2). It then localizes: the entropy splits across endpoints, rank 1 and rank 2 rising only to about 1.0 bits while rank 0 and rank 3 climb to about 1.5, and the two lowest-entropy endpoints are

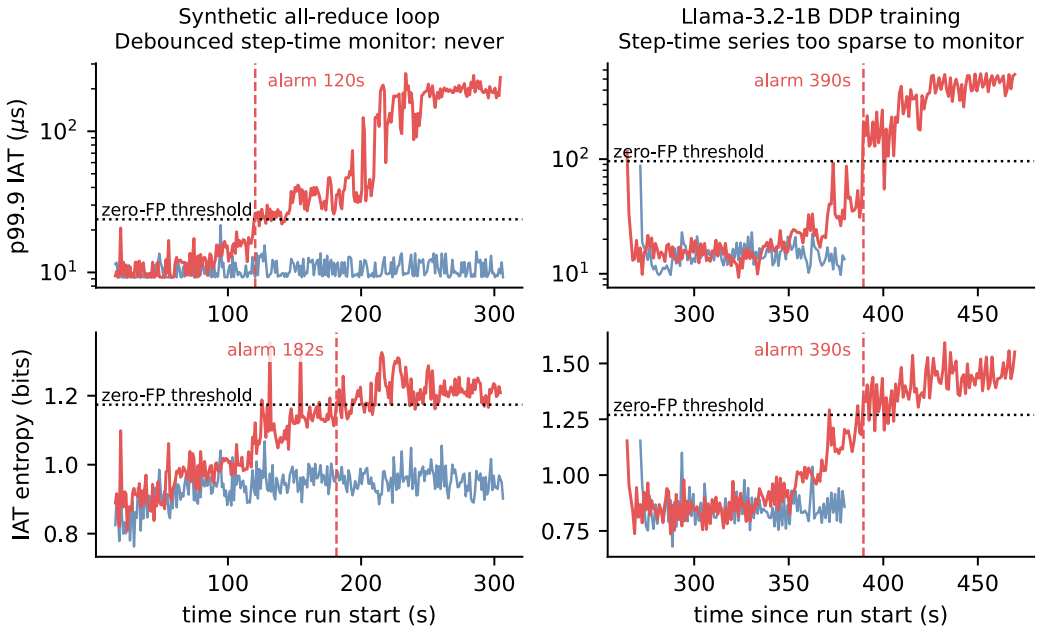


Figure 4. **The endpoint statistics alarm where the production monitors are silent.** Entropy and the p99.9 tail over time, healthy against a matched flooded run. Left: a synthetic all-reduce loop, where a debounced step-time monitor never trips. Right: real Llama-3.2-1B training, where the two statistics detect the flood with zero false alarm before onset while step time stays inside its healthy variance.

the ends of one link (Figure 3). That names rank 1–rank 2 as congested, without a switch counter or any fabric access, and the operator can throttle or reroute the competing flow. One passive signal has separated the fault type and named the link in a single pass.

6 Reconstructing the Execution

The same timing recovers state that no endpoint sees directly: the hidden execution numbers, not only an alarm. On hardware it reaches a compute magnitude and the slow pipeline stage, and we show the full per-link and per-rank map in simulation.

6.1 Single-endpoint magnitude on hardware

From one passive endpoint, Aspan measures how much extra compute a rank added, the severity of a compute fault that no endpoint tool reports today. Because the collective repeats every step, the arrival stream at an endpoint is periodic with the step period, and autocorrelating the timestamps recovers that period. Extra compute at a rank sits on the step's critical path and lengthens it, and the period grows by the added time; its increase over a healthy baseline is the recovered magnitude. This reads one term of Eq. 1 directly by autocorrelation; it is not the least-squares inversion of the full system (§6.3), which is why it yields the magnitude and not which rank. We inject a compute sleep of a

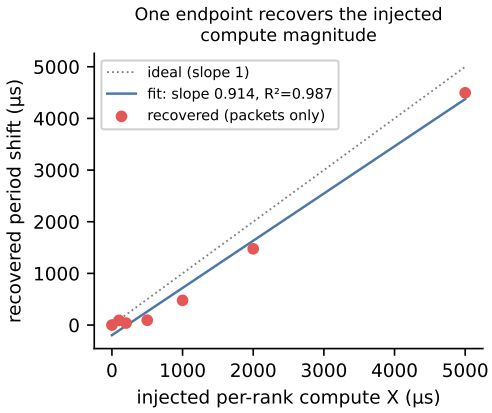


Figure 5. **One endpoint recovers the compute magnitude.** Recovered against injected extra compute on real hardware, with the ideal line. The fit has slope 0.914 and $R^2 = 0.987$.

known duration at one rank and capture at another: the recovered magnitude tracks the injected one with slope 0.914 and $R^2 = 0.987$ (Figure 5). As the model predicts, a single endpoint recovers the magnitude of the extra compute but not which rank added it; naming the rank needs coverage, which the next results provide in aggregate and simulation.

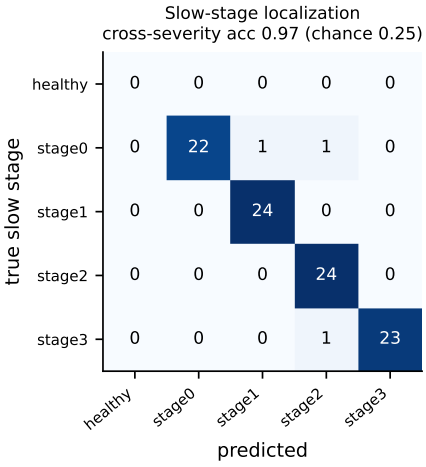


Figure 6. **Endpoint timing names the slow stage on unseen severities.** Confusion matrix over the four pipeline stages, trained on one set of severities and tested on another; the healthy class has no severity and is absent from the test. Accuracy is 0.97 against a chance rate of 0.25.

6.2 Localizing the slow pipeline stage on hardware

With every rank captured, endpoint timing names the slow stage of a pipeline. We run a four-stage pipeline across the four nodes over real RoCE, with the compute at each stage emulated by a timed delay under a blocking schedule, and inject a slow stage. Trained on some injected severities and tested on severities it never saw, the read-out identifies the slow stage at 0.97 accuracy against a chance rate of 0.25 across the four stages (Figure 6); the classifier is trained with a healthy class, which has no severity and so does not appear in the cross-severity test. Generalizing across unseen severities shows that the timing reads which stage is slow from the pipeline schedule, not a memorized fingerprint of one setting. The compute here is emulated rather than a real model's forward and backward pass, and the schedule is a blocking one. We mark both as limitations.

6.3 Per-link and per-rank recovery at scale

Four nodes cannot show how recovery behaves as the job grows. To test this, we invert Eq. 1 in a Monte-Carlo simulator at the ConnectX-5 timestamp noise of about 50 ns. This inversion names the degraded link in every trial and recovers both per-link delay and per-rank compute: the delay to 1.7% mean error at $N = 4$ and 0.18% at $N = 256$, and the coarser per-rank compute from 7.0% to 0.55% over the same range, as the number of equations grows as $N(N-1)$ against $2N$ unknowns (Figure 7). Both errors fall as power laws in N, about $N^{-0.53}$ for the delay and $N^{-0.60}$ for the compute. This is a ceiling, not an achieved hardware number. It models only the adapter's 50 ns timestamp noise, and recovery across nodes needs cross-node clock synchronization at that scale, which the current Precision Time Protocol (PTP) does not provide. Closing that gap is out of scope here. This simulation tests the conditioning of the inverse on its own model, and the packet-level results of §7 independently confirm the forward-model timing signatures the detector relies on.

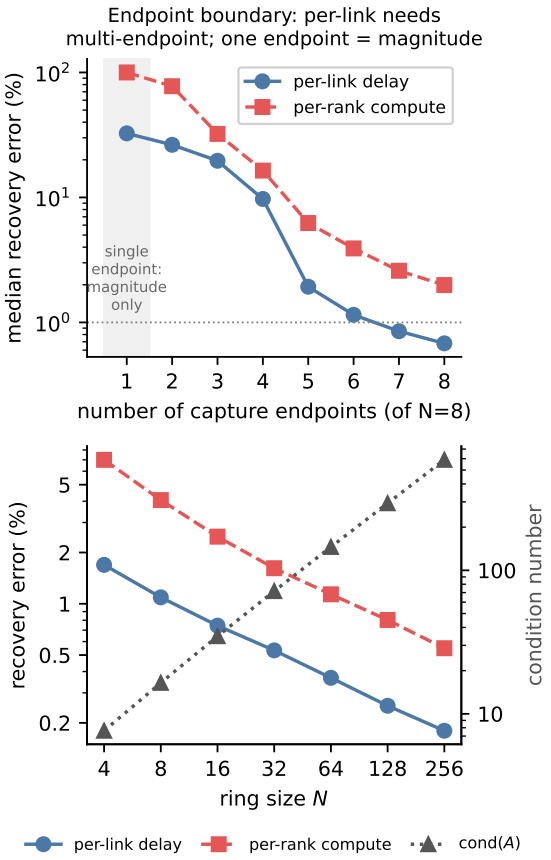


Figure 7. **Per-link and per-rank recovery needs coverage and sharpens with scale.** Top: recovery error against the number of observed endpoints; one endpoint gives only a magnitude. Bottom: error falls with ring size N as the equations outnumber the unknowns by a growing margin, even as the condition number rises.

7 Generality

The signature is not specific to one collective, algorithm, or scale. It survives the switch of algorithm on hardware, holds for all four collective types in packet-level simulation, and persists as the job grows.

7.1 Collective types

Throttling one uplink on a ring of $N = 64$ separates a healthy run from a congested one for all_gather, all_reduce, reduce_scatter, and all_to_all (Table 2), using ASTRA-sim over ns-3 (Won et al., 2026). For three of the four the

Table 2. **The mean moves for one collective, the shape for all four.** Healthy to congested, packet-level simulation at $N = 64$ with one throttled uplink. Each collective runs at a size where it saturates that link: 1 MB for all_gather, 16 MB for all_reduce and reduce_scatter, 4 MB for all_to_all. A tail marked *splits* goes from uniform across ranks to bimodal, collapsing at the throttled endpoint and rising at the receivers.

Collective	Mean inter-arrival	Entropy	p99.9 tail
all_gather	65→518 ns	0.70–1.47→0.60–2.15	15.2→0.4 μs
all_reduce	197→221 ns	0.23→0.46–2.46	20.1 μs, splits
reduce_scatter	197→206 ns	0.23→0.46–2.39	20.2 μs, splits
all_to_all	702→706 ns	0.65→1.02–2.10	20–39→30–55 μs

mean inter-arrival barely moves, by 13% or less, too small for a throughput monitor to threshold on. Entropy and the tail call every one correctly: the shape carries what the mean misses. all_gather is the exception where the mean does move, and its p99.9 tail collapses from 15 μs to below 1 μs as throttling paces the startup burst into a steady stream. This signature appears once the collective is bandwidth-bound at the throttled link. Below the sizes listed, a collective finishes before the throttle binds and shows no separation.

7.2 Algorithm and scale

Algorithm. Under the same single-link flood, the tree algorithm's tail at rank 0 rises from 61 μs to 25.7 ms, with entropy rising alongside. The tree has its own healthy per-rank profile, which a deployment learns before it runs detection.

Scale. Detection and reconstruction both improve as the job grows, by different mechanisms. Detection separation holds and even widens with scale, while reconstruction needs more coverage and its error falls as a power law in job size (§6.3). On a ring the entropy gap between a healthy and a throttled run persists from $N = 4$ to 16 (Figure 8). A throttle, unlike the competing flood of §5.2, leaves no steady stream in the capture, so the affected hosts read highest rather than lowest. On a fat-tree it widens: for all_gather the healthy mean inter-arrival stays near 65 ns from $N = 16$ to 96, while the throttled-uplink mean grows from 238 to 548 ns.

7.3 Cost and model independence

Because the receiver runs on the host as a per-step cost, its share of throughput depends on step time: under Llama-3.2-1B DDP training over the NVIDIA Collective Communications Library (NCCL) it adds 4% at a 2.3 s step, a rise from 2323 to 2416 ms, and 1.5% at an 18 s step. It is a host tax, not a differentiator. A DPU offload would move the capture off the host, the deployment path we intend but have not yet measured.

Because the read depends on the collective's schedule and not the model's weights, the same capture applies across

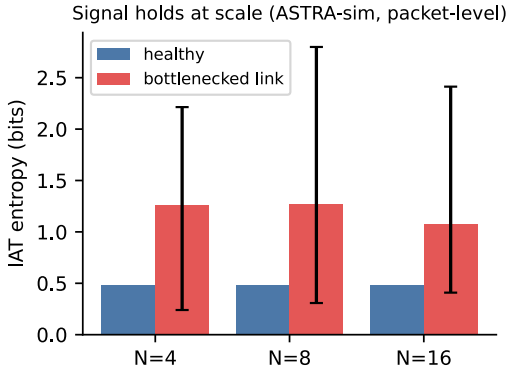


Figure 8. **The bottleneck signature holds at every ring size.** Packet-level simulation of all_gather at 1 MB, healthy against a throttled host uplink at $N = 4$, 8, and 16, read at four hosts. Healthy entropy is 0.48 bits at every host and every N; the bottleneck raises the affected hosts to 1.0 to 2.8 bits while the unaffected hosts stay at the healthy value. The bar, a mean over the four hosts, sits between the two groups, and the whisker spans them.

model families: we read clean traces from GPT-2 and Pythia on an earlier TCP ring and from Llama, a synthetic mixture-of-experts all-to-all, and a synthetic tensor-parallel all-reduce on the RoCE rig. Because the last two are the communication patterns of expert- and tensor-parallel inference, the same read spans training and inference collectives. Under a congestion flood, endpoint timing detects the fault on the tensor-parallel all-reduce at 0.85 to 0.87 AUC across three campaigns, against 0.64 to 0.75 for the adapter counters. Step time and the per-rank agent are not captured on the inference path. On the mixture-of-experts all-to-all the same detector is unstable across campaigns, from 0.34 to 0.88, because expert routing makes that schedule data-dependent.

8 Discussion

Every result in this paper reads one signal against the schedule. Read once, it detects and localizes; read continuously, it is a feedback input, and it supplies the trigger and the fault class that a throughput number cannot. Its value is that a single passive observable answers several questions that today need several tools.

A congestion event wastes GPU time that becomes recoverable once it is attributed to a component. We do not claim a self-optimizing loop. A closed-loop remediation was negative on four nodes behind one switch, where both collective algorithms cross the same link and there is nothing to reroute to. The read needs only a fixed collective schedule, which both training and inference provide. Production serving, with its continuous batching and irregular request arrivals, breaks that assumption and is future work.

9 Limitations

Our scale claims rest on simulation, since the hardware testbed is four nodes, and per-link recovery across nodes needs cross-node clock synchronization that current PTP does not provide (§6.3). Detection and single-endpoint recovery need no clock synchronization; only the multi-endpoint inversion does. The within-window shape does not flag a compute straggler on its own: a straggler shifts the cadence between bursts, not the shape within them, and the classifier catches it (0.89 recall) only with the rate and burst features. A single endpoint cannot name which rank is slow, only the magnitude. Timing does not separate packet loss from congestion, and its detection on the mixture-of-experts all-to-all is unstable across campaigns. On hardware the endpoints of the localized link also receive the competing traffic; congestion on a link that terminates at no captured endpoint is covered only in simulation. The capture path reads every packet on the host at 25 GbE; its cost at 400 Gb/s and its coexistence with GPUDirect RDMA are untested. A step that interleaves several collectives in one window needs a profile per mix, which we have not built. The pipeline-stage result uses emulated compute under a blocking schedule, not a real model's forward and backward pass. We measure Aspan head to head against the tools our testbed can run (§5.4). In-job profilers, cross-rank agents, and switch telemetry need instrumentation or fabric access we lack. We position those by capability. Every result injects a single fault at a time, and a few results rest on thin sweeps. The forward model assumes a stable path between endpoints. A route change during a step would need the model to be recalibrated. Finally, faults that do not perturb arrival timing stay invisible: silent data corruption changes results without changing when packets arrive, and hangs or control-plane failures raise errors instead.

10 Related work

Diagnosing distributed training. §2 contrasts the signals an operator has today. The closest research systems read the execution from three vantages. In-job profilers and library plugins expose per-kernel and per-collective detail (Zhou et al., 2026; Deng et al., 2025a; NVIDIA, 2026c;b; Hu et al., 2022; Wu et al., 2025). Per-rank agents correlate signals across ranks to find the faulty machine (Deng et al., 2025b; Si et al., 2025; Yao et al., 2025; NVIDIA, 2026a; Liu et al., 2026). Switch-side and in-band measurement read the run from the fabric (Jiang et al., 2025; P4.org Applications Working Group, 2020; Chen et al., 2025; Liu et al., 2025). FlowPulse (Krebs et al., 2025) starts from the same observation as Aspan, that collective traffic is predictable, but reads it at the switch. Two systems read RoCE from the endpoint, as Aspan does: R-Pingmesh (Liu et al., 2024) probes the fabric with its own RDMA pings, and C4 (Dong et al., 2025) instruments the collective library to compare worker timing at each synchronization point. Each needs probe traffic, instrumentation, or access to the fabric that Aspan does not. Step time, the adapter counters, and a per-rank agent are our measured baselines (§5.4).

Network tomography and inverse problems. Aspan's reconstruction inherits the tomography tradition of inferring internal link state from edge measurements (Vardi, 1996; Coates et al., 2002; Castro et al., 2004) and the regularized inversion that tradition relies on (Tikhonov, 1963; Hansen, 1998). That line injects probes or assumes exogenous traffic on a general topology. Here the collective's own schedule is the forward model and its own traffic the probe, which turns diagnosis into the overdetermined linear system of §3. Where prior work reports what slowed and when, Aspan reconstructs the execution behind it, the compute on each rank and the delay on each link, from the endpoint alone.

11 Conclusion

A distributed job's communication schedule turns its own packet timing into a measurement of its execution. Read forward, that measurement detects and localizes faults that a throughput monitor conflates; read inverted, it reconstructs the compute and delay behind them. One passive signal, read two ways, answers questions that today need several instrumented tools. Recovering hidden state from a known, structured signal is a general principle, and a running collective already produces the signal.

References

Baldin, I., Nikolich, A., Griffioen, J., et al. FABRIC: A national-scale programmable experimental network infrastructure. *IEEE Internet Computing*, 23(6), 2019. doi: 10.1109/MIC.2019.2958545.

Castro, R., Coates, M., Liang, G., Nowak, R., and Yu, B. Network tomography: Recent developments. *Statistical Science*, 19(3):499–517, 2004.

Chen, Y., Zhang, M., Li, X., et al. Vedrfolnir: RDMA network performance anomalies diagnosis in collective communications. In *Proceedings of the ACM SIG-*

COMM Posters and Demos, 2025. doi: 10.1145/3744969.3748396.

Coates, M., Hero, A. O., Nowak, R., and Yu, B. Internet tomography. *IEEE Signal Processing Magazine*, 19(3): 47–65, 2002.

Deng, Y., Zhang, L., Wang, Q., et al. Mycroft: Tracing dependencies in collective communication towards reliable LLM training. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, 2025a. doi: 10.1145/3731569.3764848.

Deng, Y. et al. Minder: Faulty machine detection for large-scale distributed model training. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2025b. arXiv:2411.01791.

Dong, J., Luo, B., Zhang, J., et al. Enhancing large-scale AI training efficiency: The C4 solution for real-time anomaly detection and communication optimization. In *Proceedings of the IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 1246–1258, 2025.

Gangidi, A. et al. RDMA over Ethernet for distributed training at Meta scale. In *Proceedings of the ACM SIGCOMM Conference*, 2024.

Grattafiori, A. et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Gunawi, H. S. et al. Fail-slow at scale: Evidence of hardware performance faults in large production systems. In *USENIX Conference on File and Storage Technologies (FAST)*, 2018.

Hansen, P. C. *Rank-Deficient and Discrete Ill-Posed Problems*. SIAM, 1998.

Hu, H., Jiang, C., Zhong, Y., et al. dPRO: A generic performance diagnosis and optimization toolkit for expediting distributed DNN training. In *Proceedings of the Conference on Machine Learning and Systems (MLSys)*, 2022.

Jiang, Z., Lin, H., Zhong, Y., et al. MegaScale: Scaling large language model training to more than 10,000 GPUs. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2024. arXiv:2402.15627.

Jiang, Z. et al. LLMPrism: Black-box performance diagnosis for production LLM training platforms. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2025. arXiv:2505.00342.

Krebs, J., Gavrilenko, D., Amir, D., et al. FlowPulse: Catching network failures in ML clusters. In *Proceedings of the ACM Workshop on Hot Topics in Networks (HotNets)*, 2025. doi: 10.1145/3772356.3772384.

Lin, J. et al. Understanding stragglers in large model training using what-if analysis. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2025. arXiv:2505.05713.

Liu, G., Patni, A., Lin, C., et al. Guard: Scalable straggler detection and node health management for large-scale training. In *Proceedings of the Conference on Machine Learning and Systems (MLSys)*, 2026. arXiv:2605.17879.

Liu, K., Jiang, Z., Zhang, J., et al. R-Pingmesh: A service-aware RoCE network monitoring and diagnostic system. In *Proceedings of the ACM SIGCOMM Conference (SIGCOMM)*, pp. 554–567, 2024.

Liu, W., Qian, K., Li, Z., et al. SkeletonHunter: Diagnosing and localizing network failures in containerized large model training. In *Proceedings of the ACM SIGCOMM Conference*, 2025. doi: 10.1145/3718958.3750513.

NVIDIA. Data center GPU manager (DCGM). https://developer.nvidia.com/dcgm, 2026a. Accessed 2026.

NVIDIA. NCCL inspector: Profiler plugin for NCCL. https://github.com/NVIDIA/nccl/tree/master/plugins/profiler/inspector, 2026b. Accessed May 2026.

NVIDIA. Nsight systems. https://developer.nvidia.com/nsight-systems, 2026c. Accessed May 2026.

P4.org Applications Working Group. In-band network telemetry (INT) dataplane specification. https://github.com/p4lang/p4-applications/blob/master/docs/INT_v2_1.pdf, 2020. Version 2.1.

Patarasuk, P. and Yuan, X. Bandwidth optimal all-reduce algorithms for clusters of workstations. *Journal of Parallel and Distributed Computing*, 69(2):117–124, 2009.

Si, M. et al. Collective communication for 100k+ GPUs. *arXiv preprint arXiv:2510.20171*, 2025.

Tikhonov, A. N. Solution of incorrectly formulated problems and the regularization method. *Soviet Mathematics Doklady*, 4:1035–1038, 1963.

Tursunbayev, B. Taking the pulse: Diagnosing GPU collectives from endpoint timestamps alone. In *Proceedings of the ACM SIGCOMM Posters and Demos*, pp. 2227–2228, 2026. doi: 10.1145/3789240.3830290.

Vardi, Y. Network tomography: Estimating source-destination traffic intensities from link data. *Journal of the American Statistical Association*, 91(433):365–377, 1996.

Won, W. et al. ASTRA-sim 3.0: Next-level distributed machine learning simulations via high-fidelity GPU and infrastructure modeling. *arXiv preprint arXiv:2606.10440*, 2026.

Wu, T., Wang, W., Yu, Y., et al. GREYHOUND: Hunting fail-slows in hybrid-parallel training at scale. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, 2025.

Yao, Z., Hu, P., Miao, C., et al. Holmes: Localizing irregularities in LLM training with mega-scale GPU clusters. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2025. URL https://www.usenix.org/conference/nsdi25/presentation/yao.

Zhou, J. et al. ARGUS: Production-scale tracing and performance diagnosis for over 10,000-GPU clusters. *arXiv preprint arXiv:2606.20374*, 2026.

Zhu, Y., Eran, H., Firestone, D., et al. Congestion control for large-scale RDMA deployments. In *Proceedings of the ACM SIGCOMM Conference*, 2015.