

LinkPass

Keeping AI training, Reinforcement Learning and Inference Running Through Network Link Failures

A failed network link should not force healthy GPUs to abandon useful work. LinkPass is Clockwork's network plugin for the NVIDIA Collective Communications Library (NCCL), keeping supported training, reinforcement-learning and multi-node inference workloads running through link failures. It detects stalled communication, selects healthy alternative NIC paths and safely completes interrupted transfers without restarting the workload. For covered failures with an available alternative path, the objective is a seamless path failover instead of a checkpoint rollback or a prolonged job hang.

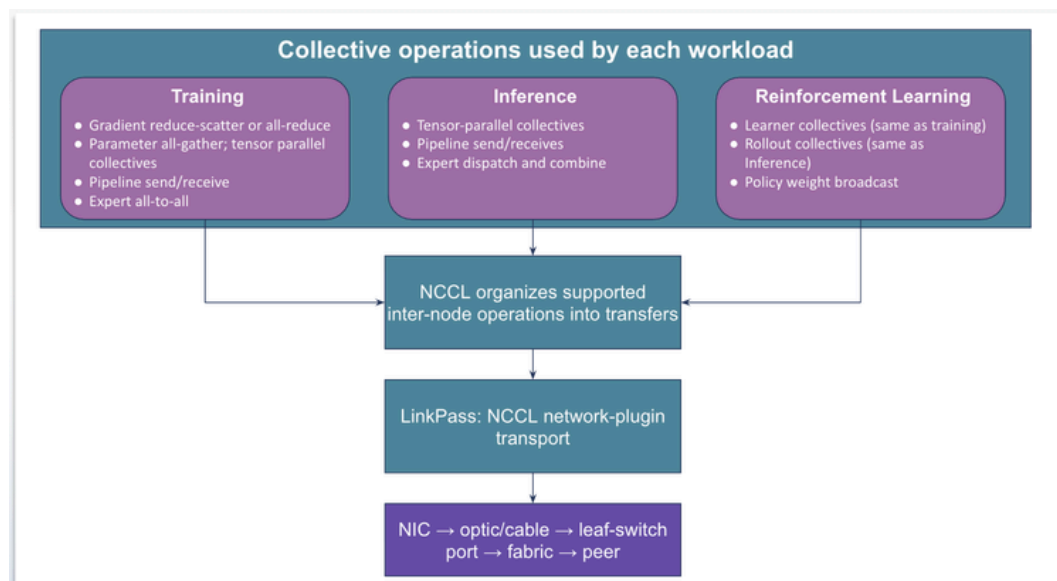
The impact of NIC-to-leaf link failures on training, RL and inference

A link flap occurs when a link transiently loses and regains connectivity. Common causes include thermal excursions, dust or contamination on optical connectors, marginal or aging transceivers and cabling faults. The flap's location matters: failures on internal fabric links (switch-to-switch) and on edge links (the segment from a host NIC to its top-of-rack (ToR) switch, also called a leaf switch) differ in blast radius and recoverability, and appear as distinct categories in production failure breakdowns [1]. Published fleet reports document network and optical-interconnect failures at scale, although they do not all measure the same failure domain. Google's TPUv4 supercomputers lose 0.005% of their optical inter-chip interconnect cables to failure each day, about 0.15% a month [2]. Alibaba's Aegis operators count 100 to 230 critical failures a week on a training cluster of about 10,000 GPUs, and across their production failures 9.1% are network interface card (NIC) errors and 6.7% are optic-module or fiber errors [3].

A flap on a leaf-switch port does not know whether the GPU behind it is training a model, serving inference requests or generating rollouts for reinforcement learning (RL). The shared problem is that an incomplete required transfer can delay the other workers that depend on it. In an all-reduce, every rank's result depends on contributions from the group. Ranks waiting for missing data cannot finish the affected operation; dependent work on their CUDA streams waits, and subsequent communication can propagate the delay through the group.

The effect of stalled communication differs by workload:

- **Training:** In data-parallel training, the optimizer update depends on completed gradient communication. In sharded training, a layer may wait for the all-gather that supplies its parameters. In pipeline-parallel training, an incomplete transfer of activations or gradients delays the next stage and can back up the pipeline.



- **Inference:** When tensor-parallel serving spans nodes, missing data in a layer's collective can delay the next token for requests being processed by that serving group. In expert-parallel serving across nodes over NCCL, an incomplete dispatch to remote experts or combine of their results delays the affected computation.
- **Reinforcement learning:** The learner depends on gradient communication in the same way a training job does. Rollout engines can also depend on inter-node communication within their serving groups and on NCCL broadcasts of updated policy weights. A stalled transfer can delay sample generation or policy updates; in an asynchronous design, it can increase the age of the policy in use.

Appendix A lists the NCCL operations and their stall and recovery behavior.

In training, inference and RL alike, the affected traffic is the same: inter-node transfers that NCCL sends through its network-plugin interface (ncclNet). When a required transfer on that path stops progressing, dependent GPU work stays blocked while the transport retries or the application waits for a timeout. LinkPass, Clockwork's NCCL network plugin, replaces NCCL's built-in InfiniBand/RoCE transport at this interface and protects all three workloads from link failures on that path.

How do operators address the pain today and what gaps remain?

Operators have several answers to a flapping edge link. Each is described below, including what it does well, and what it leaves open.

1. Extending transport timeouts

A longer retry window can absorb a short interruption when the existing transport connection remains recoverable. Instead of immediately reporting an error, the transport waits for communication to resume. The affected work still waits during that interval.

Retries do not create a new path around a persistent failure. When the connection cannot recover, a longer timeout just delays escalation to another recovery mechanism.

2. Fabric redundancy

The second answer is to give each NIC more than one path into the network. In a multi-planar fabric, a NIC's ports connect to separate switches, and each switch leads into its own plane of the network. Alibaba's High-Performance Network (HPN) is a published example: the two ports of each NIC connect to two different ToR switches, and both links carry traffic at once. In Alibaba's tests, a link flap caused negligible slowdown with two ToRs, compared with more than nine seconds of halted training with one [4]. This protection has two costs:

1. **Bandwidth drops while a link is down:** When one of a NIC's links fails, the NIC stops using that plane and carries its traffic on the planes that remain. That leaves the NIC with half its bandwidth in a two-plane design, or three-quarters in a four-plane design, and the rest of the job slows waiting on delayed communication. In HPN, a single failed link cost the job 6.25% of its throughput [4].
2. **It must be built into the cluster:** Each NIC needs its ports cabled to switches in separate planes. In HPN's design, the switches also had to present those links to the host as a single connection, so Alibaba built its own version of link aggregation (LACP) to make two independent ToRs look like one device. The extra wiring also caused many cabling mistakes during early build-out [4]. A cluster already built with one ToR per NIC cannot enable this through configuration.

3. NCCL's built-in port failover

NVIDIA provides software failover within NCCL's built-in InfiniBand/RoCE transport. Introduced in NCCL 2.29.7 and enabled with `NCCL_IB_RESILIENCY_PORT_FAILOVER=1`, it redirects affected communication to a surviving member of a merged NIC: ports or devices combined into one logical interface before communication begins. The group can be formed automatically from topology or explicitly through configuration. NCCL uses connections already established within that group and selectively retransmits missing data. NCCL 2.30.3 added separately enabled port recovery, through `NCCL_IB_RESILIENCY_PORT_RECOVERY=1`, so traffic can return to a recovered port [13, 14, 16].

The reviewed NCCL 2.32.3 implementation has three relevant limitations [15]. First, failover stays within the merged group formed before the job starts. By default, that group contains only the ports of a single physical NIC, so a single-port NIC has nothing to fail over to even when other NICs in the server are healthy. Second, if another port in the same group fails while the first is still down, NCCL stops with an error rather than using the healthy ports that remain. Third, failover starts only after the NIC reports a transport error, so the job stays stalled for as long as the NIC keeps retrying the failed link.

4. Fault-tolerant collective libraries

Another approach builds resilience into the collective library itself. VCCL adds a primary-backup queue-pair mechanism for network-port failures [5], and R2CCL extends NCCL with resilient collectives [6]. These approaches address communication failures directly, but they require running a modified or replacement build of NCCL, which adds its own versioning and validation burden. LinkPass instead integrates at NCCL's network-plugin boundary, changing the transport implementation rather than requiring a replacement build of the collective library.

5. Recovery based on checkpoint/restart

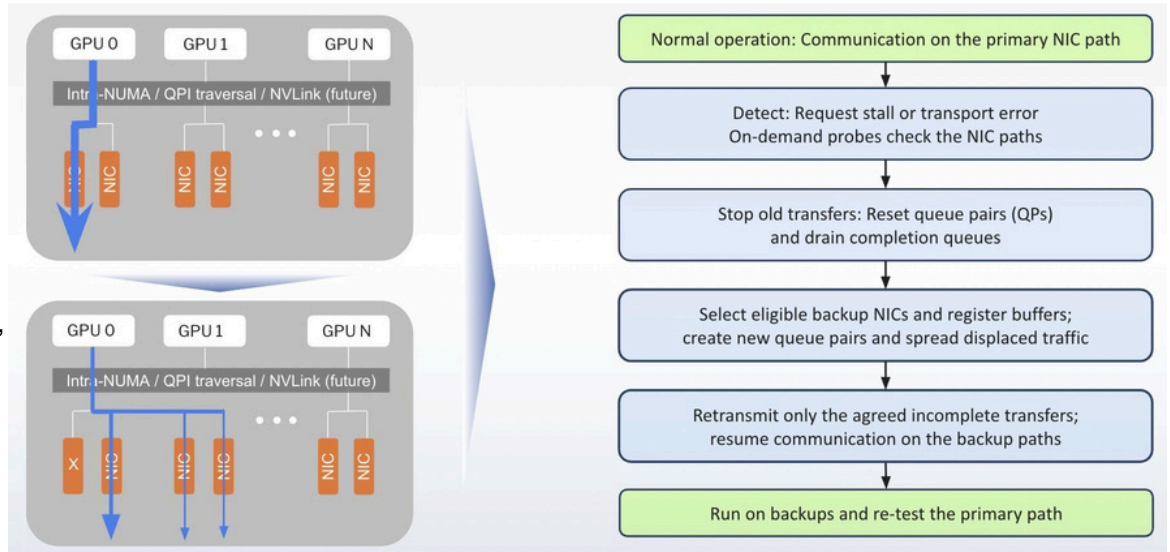
Checkpoint and restart remain necessary for failures that cannot be absorbed in place. More frequent checkpoints limit the amount of progress lost, and faster recovery reduces restart delay. Restoring execution and communication still incurs overhead [7–9]. LinkPass complements these mechanisms by keeping covered communication failures from requiring a workload restart. The objective is to preserve useful work in place while retaining checkpoint recovery for failures beyond the available transport protection.

LinkPass closes the gap between waiting for a failed path and restarting the workload. It combines progress-based detection, selection of eligible alternative NIC paths and coordinated recovery between the two endpoints. The following section explains how those mechanisms preserve the running communicator and complete interrupted transfers safely within the supported failure scope.

How LinkPass addresses the problem

LinkPass is a NCCL network plugin that implements the `ncclNet` interface using standard InfiniBand Verbs, the software interface used to operate the RDMA transport. RDMA, or remote direct memory access, moves data directly between registered memory buffers on different servers. NCCL continues to post transfers and poll for completions through its normal plugin interface. LinkPass supplies the protected transport without changes to application code or a replacement build of NCCL.

Detection. LinkPass tracks whether each connection's posted operations are making progress. A stall beyond a configurable completion threshold, or a transport error completion, raises a candidate fault. Separately, a background health monitor continuously probes over unreliable-datagram queue pairs to check alternative NIC paths and identify healthy backup pairs.



Recovery. After detecting a covered fault, the two endpoints coordinate a recovery transaction over a TCP control connection separate from the RDMA data transfer. This control connection must remain usable for the transaction to complete. The transaction preserves the running communicator and keeps interrupted work correct through the following rules:

- Before either endpoint uses a backup path, both reset the old queue pairs and drain their completion queues.
- The endpoints agree which transfers remain incomplete on both sides and retransmit only that set. A request already reported complete to NCCL by either endpoint is not resent, because its buffer may have been reused.
- Retransmissions preserve the ordering required by LinkPass's transport protocol. The receiver re-posts receives for that set on the backup path and tells the sender it is ready. The sender then retransmits the agreed incomplete transfers in the corresponding order.
- If another covered fault occurs during failover, the endpoints restart the recovery transaction rather than continuing from an inconsistent partial handoff. While recovery owns a connection, the plugin returns without reporting completion and NCCL continues polling. When recovery succeeds, communication resumes through the same distributed communicator without a workload restart. This requires a usable alternative path and continued control-channel connectivity.

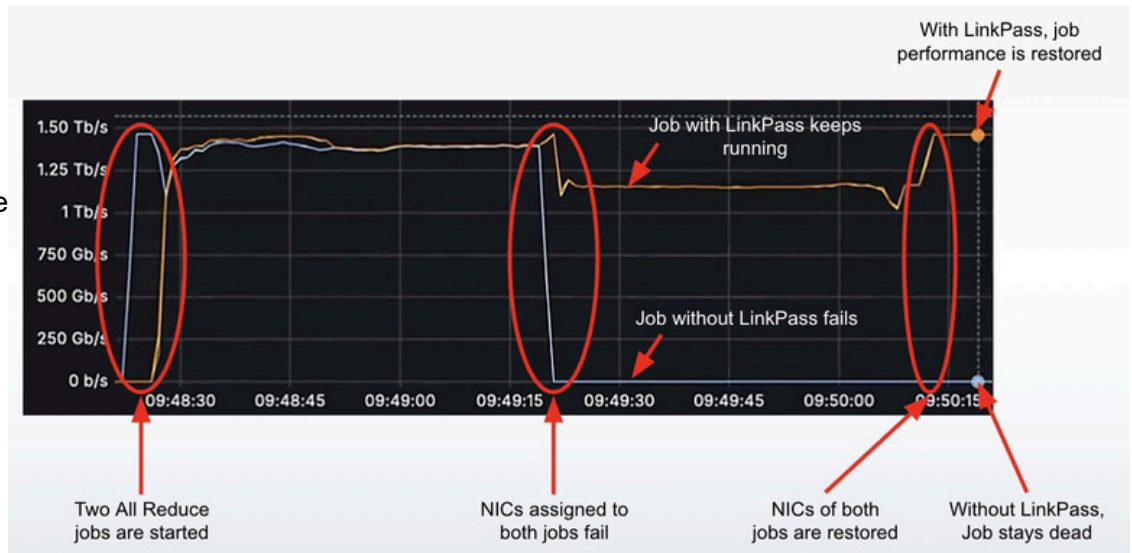
Backup paths. LinkPass creates backup queue pairs when a fault occurs rather than reserving idle backup connections at job start. It can spread displaced traffic across several eligible healthy NICs, preferring devices close in non-uniform memory access (NUMA) terms, meaning devices near the same CPU socket and memory. Before using a backup device, it registers the required buffers with that device so the new connections can address them correctly.

Failback. While communication runs on backup paths, LinkPass re-tests the primary pair at a configurable interval. Once the primary pair passes its recovery check, the endpoints use the same coordinated handoff procedure to move traffic back onto the original NICs.

Telemetry. The plugin reports whether a connection is running on a backup device. It reports each detection event and its cause. It reports health-check results, the devices and queue pairs it selected, and the traffic on each queue pair.

Deployment and coverage. LinkPass protects communication that uses its supported NCCL network-plugin path. Recovery requires viable worker processes, access to eligible alternative NICs and the relevant buffers, control-channel connectivity and a usable network path between endpoints. It complements recovery mechanisms for GPU, node and broader infrastructure failures. The hardware, software and transport configuration must be supported for the deployment.

This Clockwork demonstration compares two concurrent all-reduce jobs, one using LinkPass and one using the baseline NCCL configuration. After one NIC is taken down in each job, the LinkPass trace continues at a reduced rate (due to less aggregate bandwidth being available) while the baseline trace drops to zero. After the affected NICs are restored, the LinkPass trace returns to its pre-fault rate; the baseline trace remains at zero during the interval shown.



The graph illustrates the behavior of these configurations in this run. It is not a claim that every native NCCL configuration requires restart: optional native failover and port recovery are available in supported configurations [13–15].

Illustrative economic impact of network interruptions

Network interruptions create three kinds of economic exposure: useful computation lost when a job restarts, capacity consumed while a job waits without restarting, and engineering effort spent on each incident. The following lower and higher scenarios illustrate those costs for a 1,000-GPU training cluster. They are not measured customer results or a forecast of LinkPass savings. All inputs use explicit assumptions; the incident counts distinguish job-ending events from stalls that recover without a restart.

- **Annual cost of job-ending events** = job-ending incidents × affected GPUs per incident × (checkpoint interval ÷ 2 + detection and restart time) × assumed value per GPU-hour. All time quantities are expressed in hours. The half-interval term assumes failures occur at random points between usable checkpoints; detection and restart time excludes the recomputation already represented by that term.
- **Annual cost of non-job-ending stalls** = separate stall incidents × affected GPUs per stall × stall duration in hours × assumed value per GPU-hour. These stalls do not also count as job-ending events.
- **Annual engineering cost** = job-ending incidents × staff-hours per incident × loaded hourly staff cost. In these scenarios, incident labor includes diagnosis, repair coordination and revalidation; it is total incident effort, not an assumption that LinkPass eliminates all of that work.

Using the assumptions below, modeled annual loss exposure is approximately \$115,000 in the lower scenario and \$781,000 in the higher scenario for a 1,000-GPU training cluster. The figures combine the assumed value of lost GPU capacity with engineering cost; they are not necessarily cash savings for an owned cluster. Appendix B has the detailed assumptions.

Annual modeled cost	Lower scenario	Higher scenario
Lost computation and downtime from job-ending incidents	\$64,000	\$614,400
Separate non-job-ending stalls	\$1,422	\$17,067
Engineering effort for job-ending incidents	\$50,000	\$150,000
Total modeled loss exposure	\$115,422	\$781,467

Clockwork LinkPass keeps covered network failures from becoming broader workload interruptions. It rapidly detects stalled communication, selects eligible healthy NIC paths, coordinates safe retransmission and restores the primary path if it later becomes healthy. It does this through NCCL's network-plugin interface, without changing application code or replacing the collective library. For training, reinforcement learning and multi-node inference, the value of LinkPass is preserving useful work in place rather than falling back to a restart.

Clockwork has deployed LinkPass in enterprise production environments for more than two years. That operating experience supports the product's practical value; deployment-specific coverage and performance should be established for the customer's workload and topology.

Sources and References

1. A. Grattafiori et al., “The Llama 3 herd of models,” 2024, arXiv:2407.21783; see Sec. 3.3.4 (Reliability and Operational Challenges). [Online]. Available: <https://arxiv.org/abs/2407.21783>
2. Resiliency at Scale: Managing Google’s TPuv4 Machine Learning Supercomputer
<https://www.usenix.org/conference/nsdi24/presentation/zu>
3. Evolution of Aegis: Fault Diagnosis for AI Model Training Service in Production:
<https://www.usenix.org/conference/nsdi25/presentation/dong>
4. Qian et al., Alibaba HPN: <https://dl.acm.org/doi/10.1145/3651890.3672265>
5. M. Zhang et al., “An efficient, reliable and observable collective communication library in large-scale GPU training clusters,” 2025, arXiv preprint; code: github.com/sii-research/VCCL (fork of NVIDIA/ncccl). [Online]. Available: <https://arxiv.org/abs/2510.00991>
6. W. Wang, N. Yu, S. Xiong, and Z. Liu, “Reliable and resilient collective communication library for LLM training and serving,” 2025, arXiv preprint; code: github.com/r2cc-project/R-2CCL. [Online]. Available: <https://arxiv.org/abs/2512.25059>
7. J. Mohan, A. Phanishayee, and V. Chidambaram, “CheckFreq: Frequent, fine-grained DNN checkpointing,” in 19th USENIX Conference on File and Storage Technologies (FAST), 2021. [Online]. Available: <https://www.usenix.org/conference/fast21/presentation/mohan>
8. Z. Wang, Z. Jia, S. Zheng, Z. Zhang, X. Fu, T. S. E. Ng, and Y. Wang, “GEMINI: Fast failure recovery in distributed training with in-memory checkpoints,” in Proceedings of the 29th Symposium on Operating Systems Principles (SOSP), 2023.
9. T. Gupta, S. Krishnan, R. Kumar, A. Vijeev, B. Gulavani, N. Kwatra, R. Ramjee, and M. Sivathanu, “Just-in-time checkpointing: Low cost error recovery from deep learning training failures,” in Proceedings of the Nineteenth European Conference on Computer Systems (EuroSys), 2024.
10. Wan et al., “Robust LLM Training Infrastructure at ByteDance,” SOSP 2025;
<https://arxiv.org/abs/2509.16293>
11. Kokolis et al., “Revisiting Reliability in Large-Scale Machine Learning Research Clusters,” HPCA 2025 (Meta); <https://arxiv.org/abs/2410.21680>
12. He et al., “Unicron: Economizing Self-Healing LLM Training at Scale,”
<https://arxiv.org/html/2401.00134v1>
13. NVIDIA, “NCCL Release 2.29.7,” Key Features and Enhancements: Port Failover.
https://docs.nvidia.com/deeplearning/nccl/release-notes/rel_2-29-7.html
14. NVIDIA, “NCCL Release 2.30.3,” Key Features and Enhancements: Port Recovery.
https://docs.nvidia.com/deeplearning/nccl/release-notes/rel_2-30-3.html
15. NVIDIA NCCL source, tag v2.32.3-1, src/transport/net_ib/. See p2p_resiliency.cc (failure count, queue-pair replacement, replay and restoration), p2p_resiliency_recovery.cc (port recovery), common.cc (asynchronous events) and connect.cc (connection device membership).
https://github.com/NVIDIA/nccl/tree/v2.32.3-1/src/transport/net_ib
16. NVIDIA NCCL source, tag v2.29.7-1, src/graph/topo.cc. See ncclTopoForceMerge and ncclTopoAutoMerge for explicit and topology-based merged-NIC formation.
<https://github.com/NVIDIA/nccl/blob/v2.29.7-1/src/graph/topo.cc>

Appendix A: NCCL Operations

This appendix maps the inter-node NCCL operations discussed in the paper to their workload role and the dependency that a failed transfer can block. It applies only when the operation uses the supported network-plugin path. Recovery depends on the deployed NCCL version, enabled resiliency features and workload control plane; native port failover and recovery can complete supported cases in place [13–16]. The descriptions below do not prescribe one universal watchdog or restart timer.

Operation	Purpose and workloads	Stall and current recovery behavior
Data-parallel gradient reduction and parameter gathering (all-reduce; or reduce-scatter and all-gather under sharding)	Gradient communication combines contributions from data-parallel workers before dependent optimizer updates. Under sharded training, parameter all-gathers supply the layer with the parameters it needs. These operations occur in distributed training and RL learners using the corresponding parallelism.	A worker cannot complete work that depends on missing gradient or parameter data. Delay can propagate through the relevant communication group and subsequent training steps. Supported transport recovery can complete the interrupted exchange in place; otherwise the job relies on its configured error handling and recovery.
Tensor-parallel collectives across nodes (all-reduce; or all-gather and reduce-scatter with sequence parallelism)	These operations combine or exchange layer data across GPUs holding different parts of the model. They matter when tensor parallelism spans nodes in training, multi-node inference or RL rollout engines.	A missing contribution can delay completion of a layer and the next token for requests depending on that serving group. This exposure recurs as execution reaches further inter-node collectives. In-place recovery and failure escalation depend on the actual transport and runtime configuration, not solely on the serving engine's name.
Pipeline-parallel send and receive (point-to-point)	A pipeline transfer hands activations to the next stage and, during training, gradients to the previous stage. It crosses the network when adjacent stages are on different nodes.	An incomplete transfer first delays the dependent stage. Later microbatches can queue behind it, while stages without a dependency may continue temporarily. The pipeline can then back up. A supported transport recovery completes the transfer; a failed recovery requires the workload's configured higher-level response.
Expert-parallel dispatch and combine over NCCL (all-to-all; or all-gather and reduce-scatter)	Dispatch sends token data to the selected experts; combine returns and assembles their results. This row applies to MoE training, inference and RL configurations whose relevant inter-node expert communication uses NCCL's supported network-plugin path.	Incomplete dispatch or combine transfers delay the token computations that depend on them and can hold up the scheduled work. Other expert-communication backends are outside the coverage established in this paper. Supported in-place recovery can complete the exchange; otherwise the runtime must escalate or recover.
Policy-weight propagation (NCCL broadcast from the learner to the rollout engines)	The learner distributes an updated policy to rollout engines that generate new samples. This row applies to RL configurations that use NCCL broadcast for that transfer; it does not imply that every RL system uses this mechanism.	A synchronous workflow can wait for weight delivery before generating with the new policy. In an asynchronous workflow, incomplete delivery can increase policy age until a version limit or another dependency blocks progress. When transport recovery cannot complete the transfer, separate application-level recovery is required.

Appendix B: Economic impact of network interruptions - key inputs

Input	Lower scenario	Higher scenario	Notes informing scenario assumptions
Job-ending network events per 1,000 GPUs per year	50	100	Llama 3 logged 35 switch/cable and 7 NIC interruptions in 54 days on 16,384 GPUs, ~17 per 1,000 GPU-years [1]; Alibaba's Aegis counts 100–230 critical failures a week on a 10,000-GPU cluster, of which 9.1% are NIC errors and 6.7% optic or fiber errors, 80–190 per 1,000 GPU-years [3]
GPUs idled per job	256	512	Assumption used for both job-ending incidents and separate non-job-ending stalls.
Checkpoint interval	1	2	HPN: four production LLMs checkpoint every 2–4 hours, at about 30 GB per GPU and 100 s per checkpoint, with 5% overhead [4];
Detection and restart time (minutes)	30	60	ByteDance: A hang produces no logs “until timeout; explicit failures localized in 2–15 minutes; failover “often more than 10 minutes” [10]; Meta prices lost goodput per failure at 30 minutes of the job's GPUs [11]; Alibaba's Unicon example: 30-minute all-reduce hang, 9-minute resubmission wait, 14 minutes of setup, 15 minutes of recomputation, 68 minutes in all [12].
Flaps that stall a running job per year	200 at 20s per stall	1000 at 20s per stall	One flap under single-ToR “halts the training for more than nine seconds” [4]; NCCL's InfiniBand retries run 4.3 s per attempt and up to 30 s before an error; Fleet flap counts: HPN: 5,000–60,000 a day fleet-wide [4]; Aegis handles “dozens of link-repairing tickets every week” on 10,000 GPUs, 125–310 per 1,000 GPU-years [3].
Cost per GPU-hour (\$), B200	\$5	\$6	
Staff-hours per job-ending event (\$125 per staff-hour)	8	12	Covers detection and triage, root-cause analysis, the fix and revalidation before the job is resubmitted, across infrastructure engineers, network engineers and job owners.