Exploring the Boundary of Transparent RDMA Failover

Formal Verification of SHIFT's Optimality

Three Impossibility Theorems for Cross-NIC Fault Tolerance

SIGCOMM 2025 | Formal Proofs in Rocq 9.0

Motivation

LLM Training at Scale

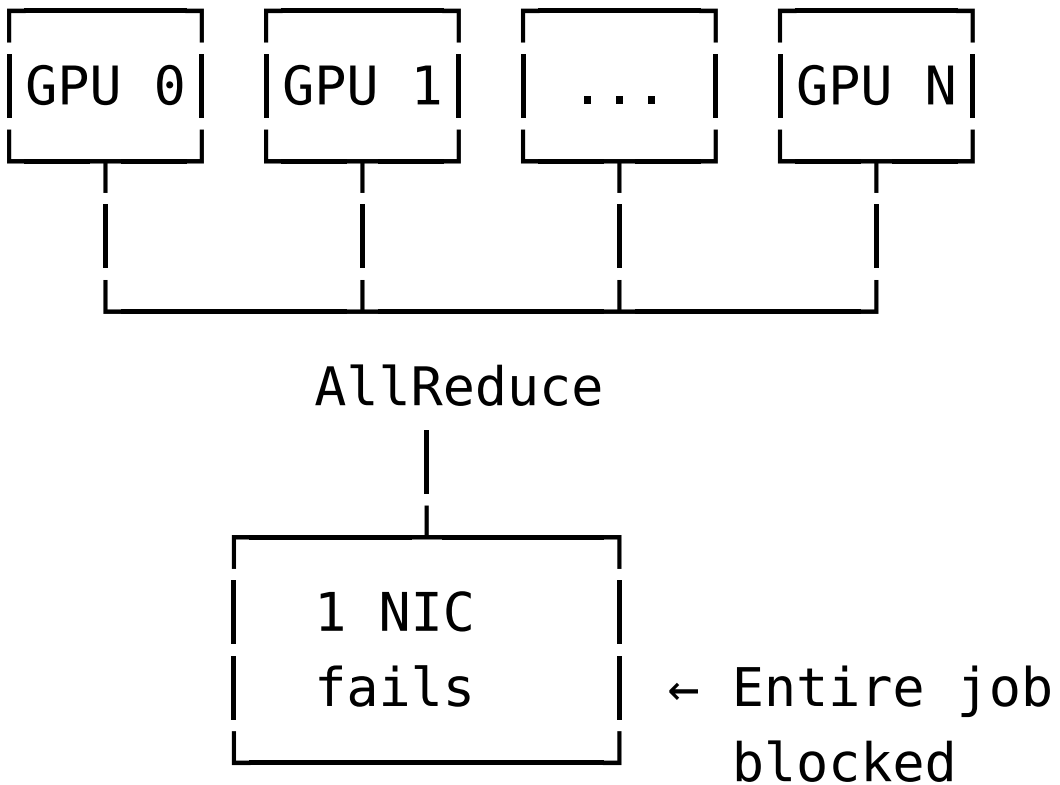
GPU Cluster Sizes

- Llama 3: 16,384 H100 GPUs
- GPT-4: Estimated 10,000+ GPUs
- Training duration: weeks to months
- Cost: millions of dollars per run

LLM Training at Scale

GPU Cluster Sizes

- Llama 3: 16,384 H100 GPUs
- GPT-4: Estimated 10,000+ GPUs
- Training duration: weeks to months
- Cost: millions of dollars per run



The Gang Scheduling Problem

Distributed training requires **all workers synchronized**.

A single failure → entire job stalls or crashes.

Network Failures Are Common in Production

Operator	Failure Rate	Breakdown
Alibaba[1]	15.8%	9.1% NICs + 6.7% optics
Azure[2]	8.3%	InfiniBand cluster failures
Tencent[3]	15% + 30%	NIC errors + network anomalies

[1] Dong et al., NSDI'25

[2] Xiong et al., 2024

[3] Meng et al., SIGCOMM'25

Network Failures Are Common in Production

Operator	Failure Rate	Breakdown
Alibaba[1]	15.8%	9.1% NICs + 6.7% optics
Azure[2]	8.3%	InfiniBand cluster failures
Tencent[3]	15% + 30%	NIC errors + network anomalies

[1] Dong et al., NSDI'25

[2] Xiong et al., 2024

[3] Meng et al., SIGCOMM'25

Rail-Optimized Topologies exacerbate this:

- Optical interconnects have higher failure rates than copper
- Multiple NICs per node $\rightarrow$ multiplicative failure probability
- Scale increases absolute frequency even with constant per-device rates

Current Approaches Fall Short

Checkpoint-Restart

- Save model state periodically
- On failure: restart from last checkpoint
- **Progress Loss**: All work since checkpoint is discarded

Runtime Resilience (Bamboo, Oobleck)

- Dynamically reconfigure pipeline
- Replace failed workers
- **Limitation**: Requires app modifications, not transparent

Current Approaches Fall Short

Checkpoint-Restart

- Save model state periodically
- On failure: restart from last checkpoint
- **Progress Loss**: All work since checkpoint is discarded

Runtime Resilience (Bamboo, Oobleck)

What We Want: SHIFT

- **Transparent** to applications (no code changes)
- Works with unmodified PyTorch + NCCL
- Continue training until next checkpoint
- Zero or minimal overhead in normal operation

- Dynamically reconfigure pipeline
- Replace failed workers
- **Limitation**: Requires app modifications, not transparent

Can we achieve this? What are the fundamental limits?

The Boundary

The Fundamental Question

What is the theoretical limit of transparent cross-NIC failover?

The Fundamental Question

What is the theoretical limit of transparent cross-NIC failover?

The Key Insight

WR-level retransmission (not packet-level) is the feasible granularity:

- Packet-level states are hardware-managed

SHIFT's Claim

We achieve the **theoretical maximum**:

- Everything that CAN be transparently retried → retried
- Everything that CANNOT → explicit error or handshake

- Sequence numbers, retry counters: inaccessible during failure
- RNIC offloads reliability → software cannot intercept
- Formally verified in Rocq (Coq)

What Does “Transparent” Mean?

Definition: Transparent Overlay

A failover mechanism that:

1. Makes decisions based **only** on sender-observable events
2. Does **not** modify application semantics
3. Does **not** require application or receiver code changes

Why is this critical?

One-Sided RDMA (WRITE/READ): The receiver CPU is **silent** (bypassed).

- We **cannot** run logic on the receiver to track state or send ACKs.
- Failover must be handled entirely by the sender's NIC/Software.

(Note: For Two-Sided RDMA, the receiver CPU is active, allowing us to mitigate this via the Handshake—see Theorem 2b.)

What Does “Transparent” Mean?

Definition: Transparent Overlay

A failover mechanism that:

1. Makes decisions based **only** on sender-observable events
2. Does **not** modify application semantics
3. Does **not** require application or receiver code changes

Why is this critical?

One-Sided RDMA (WRITE/READ): The receiver CPU is **silent** (bypassed).

- We **cannot** run logic on the receiver to track state or send ACKs.
- Failover must be handled entirely by the sender's NIC/Software.

(Note: For Two-Sided RDMA, the receiver CPU is active, allowing us to mitigate this via the Handshake—see Theorem 2b.)

The Constraint: The overlay sees ONLY:

$$\texttt{Send} \rightarrow \texttt{Timeout} \rightarrow \texttt{Completion}$$

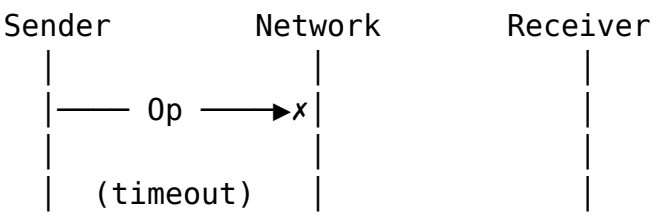
It CANNOT see: Network events | Receiver execution | Memory state

Therefore: Any retransmission decision is fundamentally a guess.

Theorem 1: Indistinguishability

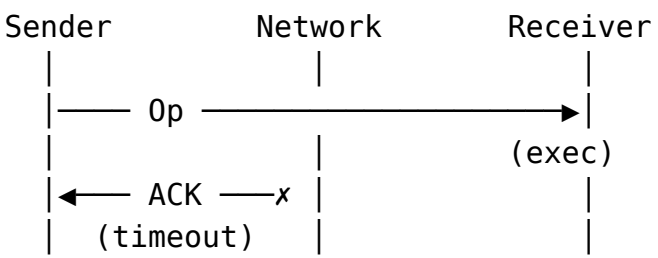
The Intuition: The Sender's Dilemma

Scenario 1: Packet Lost (Liveness requires Retry)



Intuition: "Nothing happened. I must try again or the system hangs."

Scenario 2: ACK Lost (Safety forbids Retry)



Intuition: "It finished, but I don't know it. Retrying might corrupt state."

The Formalization: Indistinguishability

In **both** scenarios, the sender observes exactly the same:

```
[ObsSent(op), ObsTimeout(op)]
```

Theorem 1: Indistinguishability (Verified in Rocq)

For any transparent overlay, there exist traces T_1 (packet lost) and T_2 (ACK lost) such that:

$$\text{SenderView}(T_1) = \text{SenderView}(T_2)$$

Yet T_1 requires retry for liveness, while T_2 forbids it for safety.

Impossibility: A transparent decision is a function of the View. Since views are equal, the decision must be equal. It **must** be wrong for either T_1 or T_2.

Theorem 2: Non-Idempotency of Atomics

Atomic Operations Cannot Be Transparently Retried

FADD: State Corruption

```
Initial:     Memory[a] = 0

1st FADD:    Memory[a]=0+δ=δ
             Returns: 0

2nd FADD:    Memory[a]=δ+δ=2δ
(retry)      Returns: δ
```

Application expects counter = δ.

Double execution gives $2\delta \rightarrow$ **Corruption**

CAS: The ABA Problem

```
Memory[a] = 0
S: CAS(0→1) succeeds → a = 1
P: CAS(1→0) succeeds → a = 0
S: Retry CAS(0→1)    → a = 1✗
```

S's single CAS executed **twice**.

P's modification silently overwritten.

$\rightarrow$ **Linearizability violated**

Atomic Operations Cannot Be Transparently Retried

Atomic Operations Cannot Be Transparently Retried

SHIFT's Response (Optimal by Theorem 2)

Return `IBV_WC_RETRY_EXC_ERR` if atomic WR is in-flight during fault.

This is the **only** correct behavior under transparency constraints.

Queue Sliding: Two-Sided Operations

The Hidden Non-Idempotency of SEND/RECV

Step	Receive Queue	Buffers
Initial	$[R_1, R_2, R_3]$	B_1, B_2, B_3 empty
M_1 arrives	$[R_2, R_3]$	$B_1 \leftarrow M_1$
ACK lost, retry M_1	$[R_3]$	$B_2 \leftarrow M_1$ (duplicate!)
M_2 arrives	$[\emptyset]$	$B_3 \leftarrow M_2$ (shifted!)

Queue Sliding: Two-Sided Operations

The Hidden Non-Idempotency of SEND/RECV

Step	Receive Queue	Buffers
Initial	$[R_1, R_2, R_3]$	B_1, B_2, B_3 empty
M_1 arrives	$[R_2, R_3]$	$B_1 \leftarrow M_1$
ACK lost, retry M_1	$[R_3]$	$B_2 \leftarrow M_1$ (duplicate!)
M_2 arrives	$[\emptyset]$	$B_3 \leftarrow M_2$ (shifted!)

Queue Sliding: Messages and buffers become permanently misaligned.

SHIFT's Solution: CQ event-based 2-way handshake re-synchronizes queue indices.

(Breaks pure transparency but hides coordination from application)

Communication Library Analysis

Which NCCL Protocols Can SHIFT Support?

Protocol	Data Transfer	Notification	SHIFT Compatible?
Simple	RDMA Write	RDMA Write_Imm (flag)	Fully supported
LL	RDMA Write	Inline flag (same WR)	Vulnerable
LL128	RDMA Write	Inline flag (8B)	Vulnerable

Which NCCL Protocols Can SHIFT Support?

Protocol	Data Transfer	Notification	SHIFT Compatible?
Simple	RDMA Write	RDMA Write_Imm (flag)	Fully supported
LL	RDMA Write	Inline flag (same WR)	Vulnerable
LL128	RDMA Write	Inline flag (8B)	Vulnerable

Why Simple Protocol Works:

- Receiver reads data buffer **only after** receiving flag via Write_Imm
- Flag WR comes **after** data WR completes on sender side
- If data WR times out → flag never sent → receiver never reads stale data
- Theorem 1′s T_2 scenario (memory reuse before retry) cannot occur!

Extended Compatibility Analysis

Library	Notification Method	SHIFT Support
NCCL Simple	RDMA Write_Imm	Full (< 0.02% CTS overhead)
NVSHMEM	RDMA Atomic (signal)	Partial (error on atomics)
MSCCL++	RDMA Atomic (signal)	Partial (error on atomics)
UCX (tag matching)	Two-sided SEND/RECV	Via handshake

Extended Compatibility Analysis

Library	Notification Method	SHIFT Support
NCCL Simple	RDMA Write_Imm	Full (< 0.02% CTS overhead)
NVSHMEM	RDMA Atomic (signal)	Partial (error on atomics)
MSCCL++	RDMA Atomic (signal)	Partial (error on atomics)
UCX (tag matching)	Two-sided SEND/RECV	Via handshake

The Partition (Formal Result)

All RDMA communication patterns fall into three classes:

1. **Idempotent Writes with Ordered Notification**: Transparent retry ✓
2. **Atomics (FADD, CAS)**: Must return error (Theorem 2)
3. **Two-Sided (SEND/RECV)**: Requires handshake (Queue Sliding)

SHIFT handles all three optimally within transparency constraints.

Theorem 3: Consensus Hierarchy Barrier

Why Read-Only Verification Fails

Herlihy's Consensus Hierarchy

Primitive	Consensus Number
Read/Write	1
FADD, Queue, Stack	2
Compare-and-Swap	∞

The Failover Problem IS 2-Consensus

Two "processes" must agree:

- **P0 (Old NIC)**: "I executed the op"
- **P1 (Backup)**: "I should take over"

CN = max processes that can solve consensus using only that primitive

They must decide: **Commit** or **Abort**?

Why Read-Only Verification Fails

Herlihy's Consensus Hierarchy

Primitive	Consensus Number
Read/Write	1
FADD, Queue, Stack	2
Compare-and-Swap	∞

The Failover Problem IS 2-Consensus

Two "processes" must agree:

- **P0 (Old NIC)**: "I executed the op"
- **P1 (Backup)**: "I should take over"

CN = max processes that can solve consensus using only that primitive

They must decide: **Commit** or **Abort**?

But: Transparent verification = Read-only

$$\mathrm{CN}(\mathrm{Read}) = 1 < 2$$

Reads alone cannot solve 2-consensus!

The ABA Attack on Any Verification

History H_1: CAS Executed	History H_0: CAS Not Executed
Memory[a] = 0 CAS(0→1) succeeds ← executed Third party: CAS(1→0) Final: Memory[a] = 0	Memory[a] = 0 (packet lost, never arrived) Final: Memory[a] = 0

The ABA Attack on Any Verification

History H_1: CAS Executed	History H_0: CAS Not Executed
Memory[a] = 0 CAS(0→1) succeeds ← executed Third party: CAS(1→0) Final: Memory[a] = 0	Memory[a] = 0 (packet lost, never arrived) Final: Memory[a] = 0

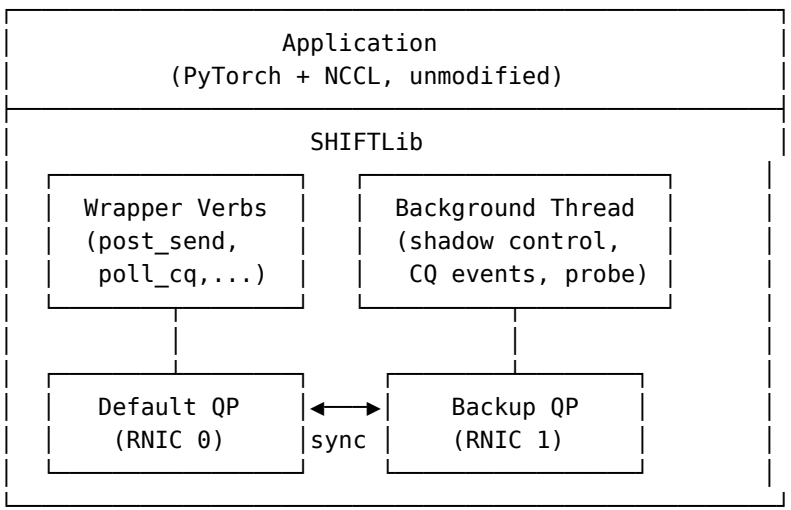
Same final memory state → Any read-based verification returns identical result

Cannot distinguish H_1 from $H_0 \rightarrow$ No correct failover solver exists

Rocq: `Theorem transparent_cas_failover_impossible`

SHIFT Design Overview

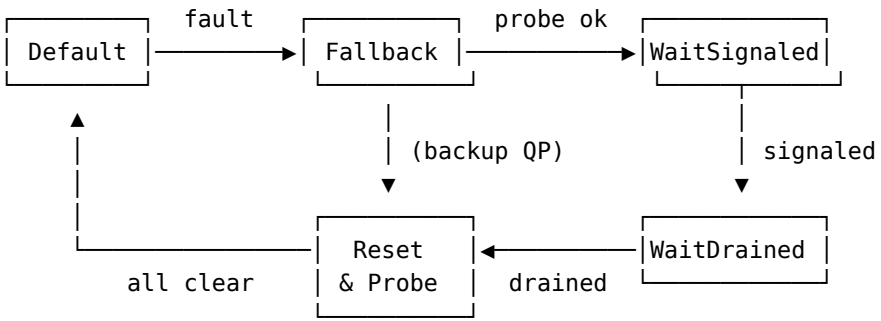
Architecture: SHIFTLib



Key Mechanisms

- Shadow control verbs
- CQ event-based handshake
- WR execution fence

SHIFT State Machine



State Transitions

1. **Default → Fallback**: Error WC
2. **Fallback → WaitSignaled**: Probe
3. **WaitSignaled → WaitDrained**: Fence
4. **WaitDrained → Default**: Resume

Latency: 500µs (WRITE), 900µs (SEND)

Sub-ms recovery!

Evaluation Highlights

Performance: Negligible Overhead

Baseline Latency (ib_write_lat)

Message	Standard	SHIFT
1 B	2.68 µs	2.69 µs
4 B	2.70 µs	2.69 µs
16 B	2.69 µs	2.68 µs

Overhead: **< 0.4%**

Zero-copy data path preserved

Memory Overhead

- Backup QP context: 138 KB each
- Backup CQ: 33 KB each
- 1000 backup QPs: 171 MB total
- **Negligible** for modern servers

Control Verb Overhead

- Shadow verbs run in background thread
- Total setup time: 35ms (non-blocking)
- Does not affect application startup

Real-World Impact: PyTorch Training

Without SHIFT:

Training starts → 9.5 minutes work → NIC fails →

Job crashes → Restart from checkpoint → **9.5 minutes LOST**

Real-World Impact: PyTorch Training

Without SHIFT:

Training starts → 9.5 minutes work → NIC fails →

Job crashes → Restart from checkpoint → **9.5 minutes LOST**

With SHIFT:

Training starts → 9.5 minutes work → NIC fails →

SHIFT failover (< 1ms) → Training continues → Checkpoint saves →

0 minutes lost (progress preserved until checkpoint)

Real-World Impact: PyTorch Training

Without SHIFT:

Training starts → 9.5 minutes work → NIC fails →

Job crashes → Restart from checkpoint → **9.5 minutes LOST**

With SHIFT:

Training starts → 9.5 minutes work → NIC fails →

SHIFT failover (< 1ms) → Training continues → Checkpoint saves →

0 minutes lost (progress preserved until checkpoint)

GPT-3 13B on 8× H100: Training continues seamlessly

Conclusion

SHIFT Achieves the Theoretical Maximum

Theorem	What It Proves	SHIFT's Response
1	Packet/ACK loss indistinguishable to sender	Retry writes (liveness)
2a	FADD/CAS are non-idempotent	Return error on atomics
2b	Two-sided ops have queue sliding	2-way handshake sync
3	CN=1 primitives cannot solve 2-consensus	No read-based verification

SHIFT Achieves the Theoretical Maximum

Theorem	What It Proves	SHIFT's Response
1	Packet/ACK loss indistinguishable to sender	Retry writes (liveness)
2a	FADD/CAS are non-idempotent	Return error on atomics
2b	Two-sided ops have queue sliding	2-way handshake sync
3	CN=1 primitives cannot solve 2-consensus	No read-based verification

Optimality Claim (Verified in Rocq)

Any system claiming broader transparent failover coverage MUST either:

- **Sacrifice transparency**: Require application/receiver modifications, OR
- **Violate correctness**: Contradict these impossibility theorems

SHIFT operates exactly at this proven boundary.

Summary: The Boundary of Transparent Failover

SHIFT operates at the proven boundary:

CAN Transparently Retry:

- RDMA Writes (idempotent)
- NCCL Simple protocol
- Ordered notification patterns

CANNOT (by theorem):

- FADD/CAS (non-idempotent)
- Uncoordinated two-sided
- Any read-based verification

Result: Zero progress loss for supported workloads (NCCL Simple).

2,100 lines of Rocq proofs verify this boundary.

Thank You

Questions?

SHIFT: Exploring the Boundary of RDMA Network Fault Tolerance

Key Theorems (Rocq)

- `impossibility_safe_retransmission`
- `fadd_not_idempotent`

Key Files

- `Indistinguishability.v`
- `FADD.v`, `CAS.v`

- transparent_cas_failover_impossible
- Hierarchy.v

Verified with Rocq 9.0 | ShiftVerification namespace