A Formal Verification of Failover Impossibility

We formalize and mechanically verify three impossibility theorems for transparent RDMA failover. All proofs are verified in Rocq 9.0 and available at https://github.com/taooceros/shift-verification.

A.1 Model and Definitions

Definition (Execution Trace)**.** A trace $\mathcal{T}$ is a sequence of events including: operation sends (`EvSend`), packet/ACK losses, executions at the receiver (`EvExecute`), completions, and timeouts.

Definition (Sender View)**.** The *sender view* $\sigma(\mathcal{T})$ projects a trace to only sender-observable events: sends, completions, and timeouts. Crucially, the sender cannot observe `EvExecute`, `EvPacketLost`, or `EvAckLost` directly.

Definition (Transparent Overlay)**.** A failover mechanism is *transparent* if its retransmission decision $D : \sigma(\mathcal{T}) \times \mathrm{Op} \to \{\mathrm{retransmit}, \mathrm{skip}\}$ depends only on the sender view—no persistent metadata, no receiver-side modifications, no application protocol changes.

Definition (Idempotency)**.** Operation op is *idempotent* on memory m if $\mathrm{exec}(\mathrm{exec}(m, \mathrm{op}), \mathrm{op}) = \mathrm{exec}(m, \mathrm{op})$ (same final state and return value).

A.2 Theorem 1: Indistinguishability of Packet Loss and ACK Loss

Theorem (Impossibility of Safe Retransmission)**.** *For any transparent overlay D, there exist traces $\mathcal{T}_1$ and $\mathcal{T}_2$ such that $\sigma(\mathcal{T}_1) = \sigma(\mathcal{T}_2)$, but safety requires $D(\sigma(\mathcal{T}_1)) = \mathrm{retransmit}$ while $D(\sigma(\mathcal{T}_2)) = \mathrm{skip}$.*

Proof. We construct two traces for a Write operation to address A_{data} with value V_1:

$\mathcal{T}_1$ (packet lost—retransmission required for liveness):

$$[\mathrm{EvSend}(W), \mathrm{EvPacketLost}(W), \mathrm{EvTimeout}(W)]$$

$\mathcal{T}_2$ (ACK lost, memory reused—retransmission corrupts data):

$$[\mathrm{EvSend}(W), \mathrm{EvReceive}(W), \mathrm{EvExecute}(W), \mathrm{EvAppConsume}, \mathrm{EvAppReuse}(V'), \mathrm{EvAckLost}(W), \mathrm{EvTimeout}(W)]$$

Both traces yield sender view $\sigma(\mathcal{T}_1) = \sigma(\mathcal{T}_2) = [\mathrm{ObsSent}(W), \mathrm{ObsTimeout}(W)]$.

- In $\mathcal{T}_1$: operation never executed → liveness requires retransmission
- In $\mathcal{T}_2$: operation executed, receiver consumed V_1 and wrote new value $V' \to$ retransmission would overwrite V' with stale V_1

Since D is a function and $\sigma(\mathcal{T}_1) = \sigma(\mathcal{T}_2)$, we have $D(\sigma(\mathcal{T}_1)) = D(\sigma(\mathcal{T}_2))$. But the traces require opposite decisions. Contradiction. □

Implication for SHIFT: This theorem explains why SHIFT cannot guarantee correctness for *all* traffic. However, for protocols where the receiver does not access data until a subsequent signal (e.g., NCCL Simple's flag mechanism), the "ACK lost + memory reused" scenario cannot occur before SHIFT completes retransmission.

A.3 Theorem 2: Non-Idempotency of Atomic Operations

Theorem (FADD Non-Idempotency)**.** *For any $\delta > 0$ and memory m, FADD is not idempotent:* $\mathrm{exec}_{\mathrm{FADD}\left(\mathrm{exec}_{\mathrm{FADD}(m,a,\delta)},a,\delta\right)} \neq \mathrm{exec}_{\mathrm{FADD}(m,a,\delta)}$.

Proof. Let $m[a] = v$. After one FADD: $m'[a] = v + \delta$, returns v. After retry: $m''[a] = v + 2\delta$, returns $v + \delta$. Since $\delta > 0$: $v + \delta \neq v + 2\delta$ and return values differ. □

Theorem (CAS Double Success)**.** *Under concurrent modification, a CAS retry can succeed even after the original succeeded.*

Proof. Consider sender S with $\mathrm{CAS}(a, 0, 1)$ and concurrent process P with $\mathrm{CAS}(a, 1, 0)$:

Step	**Actor**	**Operation**	$m[a]$
0	—	Initial	0
1	S	$\mathrm{CAS}(0 \to 1)$	1 (success)

2	S	Fault before ACK	—
3	P	CAS($1 \to 0$)	0 (success)
4	S	Retry CAS($0 \to 1$)	1 (success!)

S's single logical CAS executed twice, and P's modification was silently overwritten. □

The Fallacy: "CAS retry is safe because duplicates fail" assumes no concurrent modification—violated in real distributed systems.

A.4 Theorem 3: Consensus Hierarchy Barrier

We prove that correct failover requires solving 2-process consensus, which read-only verification cannot achieve.

A.4.1 Background: Herlihy's Consensus Hierarchy

Definition (Consensus Number)**.** $\text{CN}(X) = n$ means primitive X can solve wait-free n-process consensus but not $(n+1)$-consensus. Key results: $\text{CN}(\text{Register}) = 1$, $\text{CN}(\text{FADD}) = 2$, $\text{CN}(\text{CAS}) = \infty$.

The hierarchy is *strict*: primitives with $\text{CN} = k$ cannot implement primitives with $\text{CN} > k$.

A.4.2 Reduction: Failover Solver ⇒ 2-Consensus

Definition (Failover Solver)**.** $F : \text{Memory} \to \{\text{Commit}, \text{Abort}\}$ returns the correct decision: Commit if the original CAS executed, Abort otherwise.

Theorem (2-Consensus from Failover Solver)**.** *A correct failover solver F yields a 2-consensus protocol:*

1. *Each process P_i writes its input to* proposed[i]
2. *Both call $F(m)$ on the shared memory state*
3. *If $F(m) =$* Commit*: decide* proposed[0]
4. *If $F(m) =$* Abort*: decide* proposed[1]

Proof.
- **Wait-free**: No loops; finite steps.
- **Agreement**: Both call same F on same $m \to$ same result $\to$ same decision.
- **Validity**: If CAS executed (P_0 "won"), $F(m) =$ Commit, decision = proposed[0]. Similarly for P_1.

□

Theorem (No Correct Failover Solver Exists)**.** *The ABA problem makes F impossible.*

Proof. Consider two histories with the same final memory state m:
- H_0: CAS never executed $\to$ correct decision is Abort
- H_1: CAS executed, then concurrent process reset value (ABA) $\to$ correct decision is Commit

Both have $m[a] = 0$ (the original expected value). $F(m)$ must return both Commit and Abort. Contradiction. □

Theorem (Main Impossibility)**.** *Transparent failover for atomic operations is impossible.*

Proof.
1. Failover requires solving 2-process consensus (distinguishing H_0 from H_1)
2. Transparency limits verification to read-only operations
3. $\text{CN}(\text{Read}) = 1 < 2$
4. By Herlihy's impossibility theorem, $\text{CN} = 1$ primitives cannot solve 2-consensus

Therefore, no transparent mechanism can provide correct failover for atomics. □

A.5 Mechanization Summary

Component	Lines	Key Theorems
Core (Memory, Ops, Traces)	400	`mem_read_write_same`, `exec_op`
Theorem 1	300	`sender_views_equal`, `impossibility_safe_retransmission`
Theorem 2	250	`fadd_not_idempotent`, `cas_double_success`
Theorem 3	1150	`register_cn_1_verified`, `no_correct_failover_solver`, `transparent_cas_failover_impossible`

Table 1: Rocq formalization statistics (total 2,100 lines)

All proofs compile with Rocq 9.0 without axioms beyond the standard library. The formalization models RDMA operations as state transformers (Memory $\to$ Memory $\times$ Result), traces as event sequences, and the sender view as a projection function.

A.6 Connection to SHIFT Design

These impossibility results directly inform SHIFT's design decisions:

Theorem	SHIFT Design Choice
Thm 1: Indistinguishability	Best-effort WR-level retransmission; error propagation when safety cannot be guaranteed
Thm 2: Atomic non-idempotency	Return error if atomic WR is in-flight during fault
Thm 3: Consensus barrier	No attempt to verify execution status via memory reads; rely on protocol-level idempotency instead

SHIFT's approach—supporting NCCL Simple while rejecting atomics—is not a limitation of implementation but a necessary consequence of these fundamental impossibility results. The boundary identified in Table 1 is precisely the boundary between what can and cannot be transparently failed over.