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.

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 $\to$ 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 Operations

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

Theorem (Queue Sliding (Two-Sided Non-Idempotency))**.** *Retrying a SEND operation is not idempotent because it consumes an additional Receive WQE.*

Proof. Let the receiver queue $Q_R = [R_1, R_2, ...]$. Trace 1 (Success): Message M_1 consumes R_1. $Q_{R'} = [R_2, ...]$. ACK lost. Trace 2 (Retry): Message M_1 (retry) consumes R_2. $Q_{R''} = [R_3, ...]$. Result: M_1 is duplicated in buffers B_1 and B_2, and R_2 (intended for M_2) is lost. The streams are permanently misaligned. □

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	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. □

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

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

A.4.2 Reduction: Failover Solver ⇒ 2-Consensus

Definition (Failover Solver)**.** $F : \mathrm{Memory} \to \{\mathrm{Commit}, \mathrm{Abort}\}$ 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* $\mathrm{proposed}[i]$
2. *Both call $F(m)$ on the shared memory state*
3. *If $F(m) = \mathrm{Commit}$: decide* $\mathrm{proposed}[0]$
4. *If $F(m) = \mathrm{Abort}$: decide* $\mathrm{proposed}[1]$

Proof.

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

□

Theorem (Failover Solver Yields 2-Consensus)**.** *A correct failover solver yields a read-based protocol solving 2-consensus.*

Proof. Given F satisfying `solves_failover`, construct observation $\mathrm{obs}(e, i) := \text{if } F(m_0) \text{ then } 0 \text{ else } 1$ (constant, trivially satisfies `valid_rw_observation`) and decision $\mathrm{decide}(x) := \text{if } x = 0 \text{ then } 0 \text{ else } 1$.

- Solo P_0: `solves_failover` on `HistExecuted`(m_0) gives $F(m_0) =$ `true` → $\mathrm{obs} = 0$, $\mathrm{decide}(0) = 0$ ✓
- Solo P_1: `solves_failover` on `HistNotExecuted`(m_0) gives $F(m_0) =$ `false` → $\mathrm{obs} = 1$, $\mathrm{decide}(1) = 1$ ✓

□

Theorem (Failover Impossible by Register CN=1)**.** *No correct failover solver exists. The impossibility is formally derived from the Register CN=1 theorem.*

Proof. By mechanized reduction chain:

1. `failover_solver_yields_2consensus`: a correct solver yields `obs` (satisfying `valid_rw_observation`) and `decide` satisfying solo validity for both processes
2. `readwrite_2consensus_impossible_same_protocol`: no read-based `obs`/`decide` pair can satisfy both solo validities (Register CN=1)
3. `failover_impossible_by_read_cn`: combines (1) and (2) into contradiction

This is not a separate ABA argument—it is a formal *consequence* of $\mathrm{CN}(\mathrm{Register}) = 1$. □

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

Proof. The main theorem (`transparent_cas_failover_impossible`) lifts the CN-based impossibility to the `TransparentFailover` interface:

1. Any `provides_reliable_cas` witness yields a `VerificationMechanism` satisfying `solves_failover`
2. `failover_impossible_by_read_cn` derives contradiction via the Register CN=1 barrier

□

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`, `send_queue_sliding`, `cas_double_success`
Theorem 3	2200	`register_cn_1_verified`, `failover_impossible_by_read_cn`, `transparent_cas_failover_impossible`

Table 1: Rocq formalization statistics (total 3,900 lines)

All proofs compile with Rocq 9.0 without axioms beyond the standard library. The formalization models RDMA operations as state transformers (Memory → Memory × Result), traces as event sequences, and the sender view as a projection function. The failover impossibility (Theorem 3) is formally derived FROM the Register CN=1 theorem via `failover_impossible_by_read_cn`, mechanizing the connection to Herlihy's hierarchy.

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: Non-idempotency (Atomics)	Return error if atomic WR is in-flight during fault
Thm 2: Queue Sliding (Two-Sided)	Implement 3-way handshake to re-synchronize queue indices (breaking transparency to ensure correctness)
Thm 3: Consensus barrier	No attempt to verify execution status via memory reads; rely on protocol-level idempotency or handshake instead

SHIFT's approach—supporting NCCL Simple while rejecting atomics and synchronizing two-sided ops—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.