1 Impossibility Results for Transparent RDMA Failover

1.1 Preliminary Definitions

Definition (Silent Receiver): A receiver R that does not send application-level acknowledgments; the sender S relies solely on transport-level signals (e.g., WQE completion status). Formally: no `AppAckEvent` appears in any valid trace.

Definition (Memory Reuse): After consuming data at address A, the application at R may immediately reuse A for new data without coordination with S.

Definition (Transparent Overlay): A failover mechanism that operates without modifying application semantics or allocating additional persistent state in remote memory. Decisions are based solely on `sender_view(T)`.

Definition (Exactly-Once Delivery): A transport guarantee ensuring each message is delivered exactly once, despite failures.

Definition (Sender View): The projection function $\pi_S : \text{Trace} \to \text{SenderObs}$ that extracts only what the sender can observe: send confirmations, completions, and timeouts. This is the "crown jewel" abstraction—the sender's decision must be a function of this projection alone.

Definition (Execution Count): For operation op in trace T: $\text{exec\_count}(T, \text{op}) = |\{e \in T : e = \text{EvExecute}(\text{op}, r)\}|$. Safety requires exec_count ≤ 1.

1.2 Theorem 1: Impossibility of Safe Retransmission for Pure-Write Protocols

1.2.1 Formal Specification

Assumptions:

- `SilentReceiver`: $\forall T, e.\ e \in T \Rightarrow \neg\ \text{AppAckEvent}(e)$
- `MemoryReuseAllowed`: $\forall V_1, V_{\text{new}}.\ \exists T.\ \ \text{EvAppConsume} \in T \wedge \text{EvAppReuse} \in T$
- `Transparent(overlay)`: $\pi_S(T_1) = \pi_S(T_2) \Rightarrow \text{decide}(T_1) = \text{decide}(T_2)$

Properties:

- `ProvidesSafety`: $\text{exec\_count}(T, \text{op}) \leq 1$ (at-most-once)
- `ProvidesLiveness`: $\neg\ \text{executed}(T, \text{op}) \wedge \text{timeout}(T, \text{op}) \Rightarrow \text{retransmit} = \text{true}$

Theorem: $\neg(\text{ProvidesSafety} \wedge \text{ProvidesLiveness})$

Theorem (Impossibility of Safe Retransmission): In a system with a Silent Receiver and Memory Reuse, no transparent Sender overlay can guarantee Safety (Linearizability) during failover if the transport does not provide Exactly-Once Delivery semantics.

Proof. We prove by constructing two execution traces, $\mathcal{T}_1$ and $\mathcal{T}_2$, that are indistinguishable to the Sender S but require mutually exclusive actions to maintain safety.

The Protocol (LL128 Abstracted):

The application requires strict ordering:
1. S sends Data: $W_D = \text{Write}(A_{\text{data}}, V_1)$
2. S sends Flag: $W_F = \text{Write}(A_{\text{flag}}, 1)$
3. R (App) polls A_{flag}. Upon seeing 1, it consumes V_1 at A_{data}
4. R (App) resets $A_{\text{flag}} \leftarrow 0$ and may reuse A_{data} for a new value V_{new}

Trace $\mathcal{T}_1$ (Packet Loss — Retransmission Required):
1. S posts W_D
2. Network Failure: W_D is lost before reaching R
3. S detects timeout/error
4. State at R: A_{data} is untouched
5. **Required Action:** S MUST retransmit W_D to ensure liveness

Trace $\mathcal{T}_2$ (ACK Loss — Retransmission Fatal):
1. S posts W_D. R receives W_D. $A_{\text{data}} \leftarrow V_1$
2. S posts W_F. R receives W_F. $A_{\text{flag}} \leftarrow 1$
3. Application Execution: R (App) sees 1, consumes V_1
4. Memory Reuse: R (App) modifies A_{data} (e.g., $A_{\text{data}} \leftarrow V_{\text{new}}$)
5. Network Failure: The ACK for W_D (or W_F) is lost
6. S detects timeout/error
7. **Observation at S:** S sees "Uncompleted WQE" — identical to $\mathcal{T}_1$

The Dilemma:

- If S retransmits W_D (as required in $\mathcal{T}_1$):
  - In $\mathcal{T}_2$: R receives stale data V_1 at A_{data}
  - R currently holds V_{new} at A_{data}
  - Result: **Data Corruption (Ghost Write)** — violates safety
- If S does not retransmit:
  - In $\mathcal{T}_1$: R never receives the data
  - Result: **Deadlock** — violates liveness

Since S cannot distinguish $\mathcal{T}_1$ from $\mathcal{T}_2$ using only transport-level signals, no correct action exists. □

1.3 Theorem 2: Violation of Linearizability for Retried Atomics

1.3.1 Formal Specification

Case A: FADD Non-Idempotency

```
exec_fadd(m, addr, delta) = (m', old_val)
  where m'[addr] = m[addr] + delta
        old_val = m[addr]
```

- After double execution: $v = v_0 + 2 \cdot \text{delta}$ instead of $v_0 + \text{delta}$
- Violates: `execution_count <= 1`

Case B: CAS with ABA Problem

```
Trace: S.CAS(0→1) → P3.CAS(1→0) → S.CAS(0→1)  [retry]
```

- Both S.CAS operations succeed (return `true, 0`)
- P3's successful modification is silently overwritten
- `execution_count(S.CAS) = 2` violates atomicity

Key Insight: "Retry is safe because duplicate CAS fails" is FALSE with concurrency.

Theorem (Non-Linearizability of Retried Atomics): In a system with concurrent access and no receiver-side deduplication, a transparent overlay cannot guarantee Linearizability for RDMA Atomics under network failure.

Proof. We assume there exists a correct transparent failover mechanism $\mathcal{M}$ that retransmits atomics upon timeout. We derive a contradiction.

Case A: Fetch-and-Add (State Corruption)

Let Op = FADD(Add = 1) with initial state $v = 0$.

1. S sends Op
2. R receives Op, updates $v \leftarrow v + 1$. New state: $v = 1$
3. Failure: The ACK from R to S is lost
4. S observes a timeout
5. Following $\mathcal{M}$, S retransmits Op
6. R executes Op again: $v \leftarrow v + 1$. New state: $v = 2$

Violation: The operation Op was issued once but executed twice. The final state $v = 2$ is invalid for a single FADD. This violates at-most-once semantics.

Case B: Compare-and-Swap (Return Value Inconsistency)

Let Op = CAS(Expect = 0, Swap = 1) with initial state $v = 0$.

1. S sends Op
2. R checks $v = 0$, sets $v \leftarrow 1$. Returns Success (OldVal = 0)
3. Failure: The Success ACK is lost
4. Concurrent modification: A third party P_3 executes CAS(Expect = 1, Swap = 0), resetting $v \leftarrow 0$
5. S retransmits Op
6. R checks $v = 0$, sets $v \leftarrow 1$. Returns Success (OldVal = 0)

Violation: S believes its single CAS succeeded once. The actual history is:

$$S.\text{CAS} \rightarrow P_3.\text{CAS} \rightarrow S.\text{CAS} \tag{1}$$

The linearization point of S's operation is ambiguous. More critically, S has overwritten P_3's modification without awareness, violating the Atomicity of the concurrent schedule. □

1.4 Theorem 3: Consensus Hierarchy Impossibility

1.4.1 Formal Specification: Failover as 2-Process Consensus

The key insight is that failover coordination IS the 2-process consensus problem.

The Two "Processes":

```
Process := Past | Future
```

- **Past**: Represents what actually happened to the original CAS
- **Future**: Represents the retry decision that must be made

The Decision Space:

```
FailoverDecision := Commit | Abort
  Commit: Original CAS executed; do NOT retry
  Abort:  Original CAS not executed; retry is SAFE
```

Knowledge Asymmetry:

```
PastKnowledge := PastExecuted | PastNotExecuted
FutureObservation := FutureSeesTimeout | FutureSeesCompletion
```

The Dilemma (Two Indistinguishable Scenarios):

Scenario	Past	Future Sees	Correct Decision
1. Packet lost	`NotExecuted`	`Timeout`	`Abort` (retry)
2. ACK lost	`Executed`	`Timeout`	`Commit` (no retry)

Core Theorem: No function f : FutureObservation $\rightarrow$ Decision is correct:

```
¬∃f. f(Timeout) = Abort ∧ f(Timeout) = Commit
```

Since `Abort` $\neq$ `Commit`, no such f exists.

1.4.2 Consensus Number Analysis

Herlihy's Consensus Hierarchy:

```
cn(Read/Write Register) = 1
cn(Test-and-Set) = 2
cn(FADD) = 2
cn(CAS) = ∞
```

The Barrier:

- Failover coordination requires 2-process consensus: CN ≥ 2
- Transparent overlay can only use reads: CN $= 1$
- $1 < 2 \implies$ **Impossible**

Why Backup RNIC Doesn't Help:

```
Backup CAN execute:  CAS(addr, expected, new)
Backup CANNOT solve: "Should I execute this CAS?"
```

The decision problem has CN ≥ 2, but verification uses only reads (CN $= 1$).

Theorem (Consensus Hierarchy Barrier): Transparent Failover for RDMA Atomics is impossible because it requires solving Consensus using only Registers.

Proof. **Premise:** The application relies on RDMA Atomics (e.g., CAS) to solve consensus problems (leader election, lock acquisition). Thus, the RDMA operation provided by the overlay must maintain Consensus Number ∞.

Failover Coordination: When a network fault occurs, the Client and the Backup RNIC must agree on the state of the previous operation ("Committed" or "Aborted") to preserve linearizability. This is equivalent to solving the Consensus Problem between the "Past Attempt" and the "Future Attempt."

Available Primitives: Under the transparency constraint, the overlay cannot allocate new persistent state (epoch counters, transaction logs) in remote memory. It can only inspect existing application data via the Backup RNIC.

The Observation Limit: Inspecting application data to infer completion is equivalent to a Read operation. In Herlihy's Hierarchy:

- Read/Write Registers have Consensus Number = 1
- CAS has Consensus Number = ∞

Contradiction: By Herlihy's universality result, it is impossible to implement a primitive with Consensus Number ∞ (Reliable CAS) using only primitives with Consensus Number 1 (Reads for verification) in a wait-free manner.

Conclusion: The existence of a Backup RNIC is irrelevant because, while it can execute CAS, it lacks the shared coordination state required to decide *whether* to execute it. Thus, transparent failover for Atomics is impossible. □

1.5 Summary

Theorem	Core Argument	Impossibility Type
1. Pure-Write	Indistinguishable traces	Safety vs. Liveness dilemma
2. Atomics	Non-idempotency	Linearizability violation
3. Consensus	Herlihy hierarchy	Consensus number barrier

1.6 Coq Formalization Correspondence

The following table maps specification elements to their Coq implementations:

Concept	Coq Module	Key Definitions
Trace semantics	`Core/Traces.v`	`Trace`, `Event`, `sender_view`
Memory model	`Core/Memory.v`	`Memory`, `mem_read`, `mem_write`
Operations	`Core/Operations.v`	`Op`, `OpResult`, `exec_cas`, `exec_fadd`
Safety properties	`Core/Properties.v`	`execution_count`, `AtMostOnce`, `op_executed`
Indistinguishability	`Theorem1/Indistinguishability.v`	`indistinguishable`, `sender_view_eq`
Impossibility Thm 1	`Theorem1/Impossibility.v`	`impossibility_safe_retransmission`
FADD violation	`Theorem2/FADD.v`	`fadd_double_increment`
CAS ABA problem	`Theorem2/CAS.v`	`cas_double_success`, `cas_retry_not_generally_safe`

Concept	Coq Module	Key Definitions
Consensus numbers	`Theorem3/ConsensusNumber.v`	`consensus_number`, `cn_lt`, `rdma_read_cn`
Failover = Consensus	`Theorem3/FailoverConsensus.v`	`no_correct_future_decision`, `failover_needs_cn_2`
Hierarchy barrier	`Theorem3/Hierarchy.v`	`transparent_cas_failover_impossible`

1.6.1 Key Coq Theorems

Theorem 1 (Indistinguishability):

```
Theorem impossibility_safe_retransmission :
  forall overlay : TransparentOverlay,
    Transparent overlay -> SilentReceiver ->
    MemoryReuseAllowed -> NoExactlyOnce ->
    ~ (ProvidesSafety overlay /\ ProvidesLiveness overlay).
```

Theorem 2 (FADD Non-Idempotency):

```
Theorem fadd_double_increment :
  result_1 = ResFADDResult 0 ->
  result_2 = ResFADDResult 1 ->
  mem_read state_2 target_addr = 2.
```

Theorem 3 (Failover = 2-Process Consensus):

```
Theorem no_correct_future_decision :
  ~ exists f : FutureObservation -> FailoverDecision,
      f scenario1_future = scenario1_correct /\
      f scenario2_future = scenario2_correct.
```

This theorem directly encodes: no deterministic function from observations to decisions can be correct for both the "packet lost" and "ACK lost" scenarios, since both produce `FutureSeesTimeout` but require opposite decisions.