Fairly Ordered Atomic Multicast: Preventing Cross-Shard Front-Running

Yunus Mert Aliskan
Thesis Committee: Jérémie Decouchant, Alexios Voulimeneas
Delft University of Technology

1. Background & Motivation

Decentralized Finance has grown into a massive ecosystem, but adversaries frequently exploit transaction ordering protocols to extract **Maximal Extractable Value** via front-running.

Fair-ordering protocols enforce consensus rules to strictly order transactions and mitigate these exploitations.

Atomic Multicast improves scalability by partitioning the network into independent shards. Transactions are processed only by affected shards. However, coordinating fairness across independent partitions is highly complex.

Haechi [1] is a leading atomic multicast protocol proposed to support fair-ordering. It utilizes consensus protocols in each shard and builds a total order for order-sensitive transactions.

2. The Architectural Blind Spot

- Haechi treats the underlying Byzantine fault-tolerant (BFT) protocol as a black box.
- Global ordering relies on **block creation timestamps**, ignoring node-based arrival times.
- This design overlooks the **mempool** (local buffers where transactions wait before BFT ledger extraction).
- Shard congestion causes unpredictable **mempool delays**.
- **Conclusion:** Haechi's global sequencing completely fails to account for localized mempool latencies.

3. Research Questions

To address these challenges, we investigate the following primary research questions:

1. Under what precise mathematical conditions does Haechi's reliance on independent local ledger timestamps fail, allowing an adversary to manipulate the processing order and violate finalization fairness?
2. How can the protocol be formally redesigned to eliminate cross-shard mempool front-running, and what are the theoretical trade-offs of this mitigation regarding network synchrony bounds, message complexity, and formal liveness guarantees?

4. Transaction Ordering & Fairness

To evaluate the fairness of the cross-shard ordering, Haechi relies on the following relationships for Order-Sensitive Contracts (OSCs):

- **Processing Order ($<_P$):** The globally deterministic sequence defined by the coordinator, utilizing the timestamp $\Psi(M)$ of the first shard ledger where a transaction appears.
- **Execution Order ($<_E$):** The actual sequence in which transactions execute and modify an OSC's state.
- **Finalization Fairness:** The fundamental security property requiring that processing order strictly dictates execution order:

$$M_1 <_P M_2 \implies M_1 <_E M_2$$

- **Conclusion:** An adversary exploiting localized mempool delays secures an earlier processing timestamp ($\Psi(M_a) < \Psi(M_v)$), granting execution priority and violating fairness.

5. Cross-Shard Mempool Front-Running Attack

We propose a novel cross-shard mempool front-running attack vector. An adversary with practical capabilities can execute this attack without violating the consensus safety or liveness of individual shards.

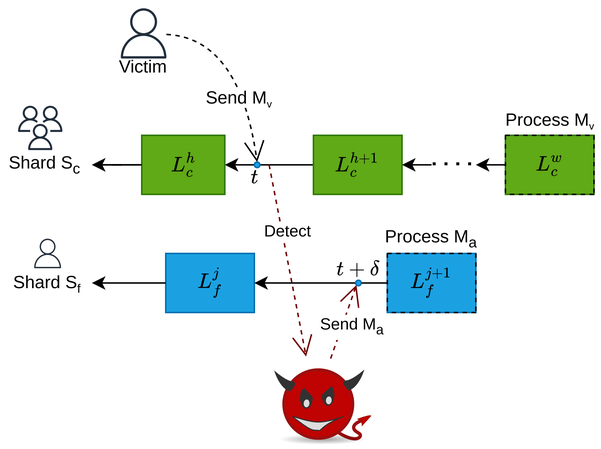

Figure 1. Process of the cross-shard routing architecture and injection timing. **By bypassing the congested shard, the attacker successfully front-runs the victim despite entering the network later.**

Execution Sequence

Consider a sharded network with a congested shard S_c and an uncongested shard S_f (as depicted in Figure 1). A victim sends a transaction, M_v, that enters the network at time t. Let L_x^y denote the ledger produced by shard x at sequence index y.

1. **Detection:** The attacker monitors the mempool of S_c and detects a victim transaction M_v.
2. **Cross-Shard Injection:** After a network routing delay δ, the attacker submits a competing transaction M_a directly to the uncongested shard S_f at time $t + \delta$.
3. **Alternative Finalization:** The transaction M_a experiences a minimal mempool delay Q_f in S_f and is processed in ledger L_f^{j+1}, whereas the victim transaction M_v is forced to wait in S_c for an extended delay Q_c and is processed later in ledger L_c^w.

Conditions for Success

The global coordinator assigns processing timestamps based on shard finalization:

$$\Psi(M_v) = \mathcal{T}(M_v, S_c) \approx t + Q_c$$
$$\Psi(M_a) = \mathcal{T}(M_a, S_f) \approx t + \delta + Q_f$$

The attacker successfully gains priority in the processing order ($M_a <_P M_v$) when $\mathcal{T}(M_a, S_f) < \mathcal{T}(M_v, S_c)$. This simplifies to the boundary condition:

$$\delta < Q_c - Q_f$$

The adversarial routing delay (δ) is tightly constrained to tens of milliseconds using optimized block distribution networks and strategic routing [2, 3]. Thus, a multi-second delay difference ($Q_c - Q_f$) reliably fulfills this condition, rendering the attack highly feasible in practice.

6. Proposed Mitigation

We propose replacing block-based timestamping with **per-transaction entry timestamping**.

- **Median Entry Time:** Validators log reception times. A deterministic median establishes an immutable consensus timestamp (τ_i).
- **Quorum Certificate:** Validators cryptographically sign to prevent unilateral leader manipulation.
- **Processing Order:** Coordinator assigns the minimum entry timestamp ($\Psi(M) = \min\{\tau_i\}$) from destination ledgers.

Minimum-Barrier Sorting

- **Problem:** Eager execution in the global layer still violates fairness due to varying shard speeds.
- **Solution:** Establish a global baseline minimum synchrony barrier (B_{min}).
- **Execution:** Coordinator only extracts/sorts transactions satisfying $\Psi(M) \leq B_{min}$.

tta

7. Complexity & Trade-offs

Messaging Complexity

Our mitigation preserves the baseline efficiency of atomic multicast over global broadcast. Given $N = S \cdot k_{max}$, where k represents the number of destination shards, S the validators per shard, and N the total network validators:

$$\mathcal{O}(k \cdot S^2) \leq \mathcal{O}(k_{max} \cdot S^2) \ll \mathcal{O}((S \cdot k_{max})^2) = \mathcal{O}(N^2)$$

Liveness and Synchrony Bounds

Minimum-Barrier Sorting anchors global ordering to the slowest shard. To preserve strict fairness, shards must reject transactions delayed in their local mempools beyond the established barrier.

- **System Liveness:** Preserved. The global coordinator continuously advances without deadlocking.
- **Transaction Liveness:** Reduced. Transactions must be proposed before the advancing barrier invalidates their timestamp.

To prevent a Byzantine shard from permanently halting the system, a synchronization timeout (Δ_m) is enforced. If a shard exceeds Δ_m, a view-change excludes it, reducing the network model to **partial synchrony**.

Timestamp Rebasing (Alternative)

To maintain standard liveness, delayed transactions could be rebased to the consensus time they are processed. However, this fatally reintroduces mempool delays to the timestamp, and creates the same front-running vulnerability.

References

[1] J. Zhang, W. Chen, S. Luot, T. Gong, Z. Hong, and A. Kate. "Front-running Attack in Sharded Blockchains and Fair Cross-shard Consensus". In: *Network and Distributed System Security (NDSS) Symposium 2024*. Feb. 2024.

[2] U. Klarman, S. Basu, A. Kuzmanovic, and E. G. Sirer. *bloXroute: A Scalable Trustless Blockchain Distribution Network*. Whitepaper. bloXroute Labs, 2018.

[3] W. Tang, L. Kiffer, G. Fanti, and A. Juels. "Strategic Latency Reduction in Blockchain Peer-to-Peer Networks". In: *ACM SIGMETRICS Performance Evaluation Review* 51.1