0.1 Mathematical Foundations

Adapted from sections of ./AE-Specifications/chapters/@GPT-reversibility.tex

In low-latency, high-throughput Layer 2 environments (e.g., Ethernet links), it's useful to model transactions as mathematical operations that can be precisely undone. This enables rollback, audit, and error recovery without heavyweight protocols.

1. **Model data as vectors.**
   Each Ethernet frame is viewed as a vector in $\mathbf{GF}(2)^n$, treating bits not as opaque payload but as elements in a vector space over a finite field.
2. **Transactions as invertible operations.**
   The sender and receiver maintain a shared state $S \in \mathbf{GF}(2)^n$. A transaction is an invertible linear transformation T applied to that state: $S' = T(S)$. Because T is invertible, the original state can always be recovered via T^{-1}.
3. **Reversibility via state updates.**
   To reverse a transaction, one sends a message (or derivable signal) allowing the application of T^{-1}. This guarantees deterministic rollback.

We consider a chain of $N+1$ nodes labeled $A_0 \to A_1 \to \cdots \to A_N$, where each node A_i maintains a local state vector $S_i \in \mathbf{GF}(2)^n$, typically initialized to the all-zero vector 0^n or some other agreed-upon state. Each link $(A_i \to A_{i+1})$ between adjacent nodes is associated with an invertible linear transformation $T_{i,i+1}$, which governs how state updates propagate along the chain.

0.1.1 Forward Execution

To execute a transaction spanning all links:

1. At each hop i, node A_i applies $T_{i,i+1}$ to its state S_i and transmits the transformation to A_{i+1}.
2. Node A_{i+1} applies the same $T_{i,i+1}$ to its own state S_{i+1}, maintaining link-local consistency.

The result is a chained sequence of transformations:

$$S_i' = T_{i,i+1} \cdot S_i \quad \text{for } i = 0, 1, \ldots, N-1, \quad S_N' = T_{N-1,N} \cdot S_N.$$

0.1.2 Rollback (Reverse Direction)

Reversibility is achieved by applying the inverse transformations in reverse order:

1. Node A_N applies $T_{N-1,N}^{-1}$ to revert S_N' to S_N.
2. It signals node A_{N-1}, which applies $T_{N-1,N}^{-1}$ and then $T_{N-2,N-1}^{-1}$, and so on.
3. This continues up the chain until A_0 applies $T_{0,1}^{-1}$, restoring the original S_0.

0.1.3 Example: XOR-Based Masks

If each $T_{i,i+1}$ is a simple XOR with mask $\Delta_{i,i+1}$, then:

$$S_i \mapsto S_i \oplus \Delta_{i,i+1}, \quad S_{i+1} \mapsto S_{i+1} \oplus \Delta_{i,i+1}.$$

Reversing just involves reapplying the same mask due to $\Delta \oplus \Delta = 0$.

0.1.4 Notes on Synchronization

- **Acknowledgments:** Each node should confirm that the next node has applied its transformation before committing its own.
- **Composite View:** The full transaction across N links is a composition:
$$T_{\text{total}} = T_{N-1,N} \circ T_{N-2,N-1} \circ \cdots \circ T_{0,1}.$$
- **Error Handling:** Any failure in transmission or transformation must be detected early, as desynchronization across nodes can compound. Redundant encodings, checksums, or commit/abort protocols may be used.

0.2 Atomic Transactions on Æ-Link

0.2.1 One-Phase Commit

0.2.2 Two-Phase Commit

0.2.3 Four-Phase Commit

0.3 Flow Control and Backpressure

0.4 Transactions on Trees