Master Thesis Proposal
Algebraic Server Reconciliation for Online Games

Philipp Hinz
Reviewed by Dr.-Ing. Guido Rößling

26. March 2025

1 Introduction

Modern online games often involve multiple players interacting within a shared virtual world. This requires synchronizing game elements, referred to here as „entities," across all participants' systems. To combat cheating, a common approach is server-authoritative architecture, where the central server dictates the game's logic and state. Clients transmit their inputs to the server and receive state updates in return. However, network latency, due to physical limitations, introduces a delay in receiving these updates, hindering real-time responsiveness, especially in fast-paced games.

Client-side prediction helps mask this latency. Each client simulates the game's progression based on its inputs, providing immediate feedback. However, this can lead to discrepancies if the client's simulation deviates from the server's authoritative state. Simply overwriting the client's state with the server's is problematic because the server's update reflects a past state. Techniques to reconcile these past state updates with the present client state are known as „server reconciliation."

2 Related Work

The prevalent server reconciliation method involves rolling back the client's simulation to the past state provided by the server and then reapplying the user's inputs that have occurred since that time. This rollback can be achieved in two primary ways: either the server transmits the complete game state with each update (increasing network load and potential synchronization problems) or, if the game physics are deterministic, the client stores past states and replays other players user inputs.

Both approaches are computationally demanding, as they require the game simulation to run at least twice per frame. Optimizations that avoid full simulation replays often necessitate custom solutions, increasing development complexity and the risk of synchronization errors. Furthermore, the deterministic approach requires each client to possess complete knowledge of the game state, which is not only resource-intensive for large worlds but also increases vulnerability to cheating.

Alternatives to server-authoritative models exist, such as client-authoritative architectures. In this approach, each player has direct control over their character or objects, sending state changes rather than inputs to the server. While easier to implement, this method is highly susceptible to cheating, as players can transmit manipulated state information. Server reconciliation also provides benefit in simulating other player inputs, allowing low-latency feedback.

3 Problem Statement

This thesis introduces **Algebraic Server Reconciliation**, a novel approach to applying past state changes to the current game state. This method aims to reduce computational overhead compared to traditional rollback methods while maintaining flexibility and avoiding the need for deterministic physics. It offers a more general framework than popular existing solutions like GGPO (Good Game Peace Out, a rollback-based networking library).

4 Approach

Algebraic Server Reconciliation leverages the concept of an abelian group to model the game state. This requires defining an associative and commutative addition operation + for the game state, along with a zero element and an inverse (negative) for each state element. The server transmits state **changes** to the client; if no change has occurred, a zero element is sent. The client keeps track of state changes since the last server update.

Upon receiving a server update, the client identifies the corresponding past state change it made at the equivalent time. It then calculates the difference between the server's change and its own past change. This difference represents the divergence between the client's prediction and the server's authoritative state. This difference is then added to the client's current state, effectively correcting the discrepancy. Over time, this process leads to convergence between the client and server states.

More formally:

Let $s^c_{t_0}$ and $s^s_{t_0}$ be the client and server states at time t_0, respectively. Assume the client has simulated two steps forward, reaching state $s^c_{t_2} = s^c_{t_0} + x_{t_0} + x_{t_1}$, where x_{t_0} and x_{t_1} are the client's state changes at times t_0 and t_1. The server then sends a state change y_{t_0}. The client updates its state by adding the difference $\left(y_{t_0} - x_{t_0}\right)$:

$$\begin{aligned} & s^c_{t_2} + \left(y_{t_0} - x_{t_0}\right) \\ &= \left(s^c_{t_0} + x_{t_0} + x_{t_1}\right) + \left(y_{t_0} - x_{t_0}\right) \\ &= \left(s^c_{t_0} + y_{t_0}\right) + x_{t_1} + \left(x_{t_0} - x_{t_0}\right) \\ &= s^s_{t_1} + x_{t_1} \end{aligned} \tag{4.1}$$

Assuming $s^c_{t_0} = s^s_{t_0}$

This demonstrates how applying the difference re-integrates the server's past state change without requiring a full rollback and re-simulation.

5 Planned Steps

This thesis will investigate the feasibility and limitations of Algebraic Server Reconciliation, with a particular focus on identifying effective methods for representing game states as abelian groups. The research will explore both the benefits and potential drawbacks of this approach. Two development tracks are planned:

1. Proof-of-Concept Simulation: A small-scale simulation will be developed, implementing both Algebraic Server Reconciliation and a conventional rollback-based method. This will enable a direct performance comparison and provide an initial demonstration of the proposed technique's viability.
2. Full-Fledged Game Prototype: A more comprehensive game prototype will be constructed to explore the constraints and opportunities presented by a more realistic and complex game environment. This will facilitate the identification of potential limitations and allow for refinement of the approach for practical implementation.

6 Additional Ideas

The core concept of Algebraic Server Reconciliation lends itself to several potential optimizations and refinements:

1. Entity Removal Handling: When a client removes an entity (represented as a state change $-a$) before receiving server confirmation, the client must retain a copy of the original entity state (a). This is necessary because the reconciliation process requires calculating $(y - x)$, where x might be $-a$. Therefore, the client needs to be able to compute $-(-a) = a$, effectively „undoing“ the removal locally. Crucially, this full entity data is only required for client-initiated removals. Server-initiated removals, represented by y, only need to contain the entity's identifier, as the client never needs to compute the inverse of y. This asymmetry helps minimize the data the server needs to transmit.
2. Efficient Change Comparison with Merkle Trees: To determine whether client and server state changes for a given entity and its components are identical, Merkle trees can be employed. Instead of comparing the raw data of each change, the client and server can compare the Merkle roots (hashes) of their respective change sets. A mismatch in the roots indicates a discrepancy, while matching roots provide high confidence that the changes are identical, significantly reducing the computational cost of comparison. This is especially beneficial for complex entities with numerous components.