Network Project - Phase 2

GSync v2 Protocol Mini-RFC

Submitted By

Rawan Hany Shaker	23P0393
Hania Soliman	23P0033
Peter Maged	23P0192
John George	23P0266
Mohamed Tarek	2300535

GSync v2 Protocol Mini-RFC

Protocol Name: GSync v2 (Grid Synchronization Protocol Version 2) **Status:** Phase 2 Implementation **Date:** December 2025

Repo link: https://github.com/mo7amedmengasu/Grid-Clash-networking_project-.git

1. Introduction and Design Philosophy

GSync v2 is a **lightweight, UDP-based application-layer protocol** designed for **low-latency synchronization of game state** in multiplayer environments. It prioritizes real-time consistency and efficient bandwidth utilization.

The protocol addresses the unacceptable latency of existing TCP-based or heavyweight frameworks like ENet for fast-paced games.

1.1 Key Motivations & Constraints

The design goals reject reliability for **responsiveness**.

Feature	Specification/Target	Source
Transport	UDP only (connectionless, unreliable)	
Update Rate	20-60 Hz (Phase 2 uses 20 Hz)	
Latency Target	≤50 ms average end-to-end	
Header Size	Compact 28-byte header	
Loss Tolerance	Graceful degradation at 2−5% loss (Redundancy-based)	

Export to Sheets

1.2 Core Design Principles

- **No Retransmission** and **No Acknowledgments** (reduces round-trip overhead).

  Timestamp-based Reordering (clients handle out-of-order packets).

- **Redundancy over Windowing** (simpler, effective for short timescales).

2. Protocol Architecture

2.1 Entities and Roles

- **Server (Authoritative):** Maintains ground truth grid state, broadcasts snapshots at fixed intervals, and resolves conflicts.

- **Clients (Followers):** Receive and apply snapshot updates, send input events (cell acquisitions), and render local game state.

2.2 Communication Flow

The communication flow shows the connection setup, continuous synchronization (20 Hz), and termination. > **Key Steps:** Client sends **INIT** → Server registers client → Server **Broadcasts SNAPSHOT+REDUNDANCY** (20 Hz) → Clients apply updates → Server sends **GAME_OVER** (x2) on end condition.

3. Message Formats

3.1 Header Structure (28 bytes)

The header is a fixed 28-byte structure. The payload length is limited to 1200 bytes per packet.

Field	Offset	Size (bytes)	Type	Description
Protocol ID	0	4	ASCII	"GSYN" (0x4753594E)
Version	4	1	uint8	Protocol version = 1
Message Type	5	1	uint8	See section 3.2
Snapshot ID	6	4	uint32	Incremental counter

Sequence Number	10	4	uint32	Packet sequence for gap detection
Server Timestamp	14	8	uint64	ms since epoch (for latency calculation)
Payload Length	22	2	uint16	Bytes in payload (0-1200)
CRC32 Checksum	24	4	uint32	CRC32(header_zero + payload)

3.2 Message Types

Type	Value	Direction	Payload	Purpose
INIT	3	C → S	`player_id` (1 byte)	Register client
SNAPSHOT	0	S → C	Grid state (~303 bytes)	Broadcast state
EVENT	1	C → S	Acquire request (12 bytes)	Player action
GAME_OVER	4	S → C	Winner + scores	End game

4. Reliability and Performance Features

The protocol leverages redundancy instead of traditional retransmission to maintain low latency.

4.1 Redundancy-Based Reliability (K=3)

The SNAPSHOT payload includes **K=3 redundancy**, meaning the current snapshot is sent along with the two most recent previous snapshots (150ms history at 20 Hz).

- The **Client extracts the first (newest) snapshot only**.

  Redundancy provides insurance against burst loss and reordering, leading to better user experience than late corrections.

4.2 Critical Event Reliability (Double-Send)

For critical but rare events, like the cell acquisition **EVENT** , the client performs a **double-send** with a 1ms spacing. This is effective simple redundancy to dodge the same loss pattern.

4.3 Checksum and Detection

- **CRC32 Checksum Validation:** The 4-byte CRC32 in the header detects bit-flip errors and is industry standard. Packets failing validation are discarded.

  Duplicate Detection & Suppression: Clients track the `last_seq_num` to skip duplicates and outdated snapshots.

  Sequence Gap Detection: Clients detect lost packets by checking if seq_num>last_seq_num+1.

5. Experimental Evaluation

The protocol's performance is tested across four scenarios, including a **Baseline**, 2% and 5% **Packet Loss**, and **100ms Latency**.

5.1 Acceptance Criteria (Expected Results)

Scenario	Latency	Jitter	Loss Tolerance
Baseline	≤50ms	<10ms	99% delivery
Loss 5%	≤75ms	<20ms	95% delivery
Delay 100ms	100−150ms	<25ms	95% delivery

5.2 Metrics Collected

Metrics include per-packet measurements logged to CSV files, such as **latency** (recv_time−server_timestamp), **jitter**, **duplicate flag**, and **gap count**. Aggregate metrics include mean, median, and 95th percentile for latency and jitter, duplicate rate, and server-side CPU utilization.

6. Evaluation

Scenarios: Baseline, 2-5% loss, 100ms delay.
Metrics: Latency (50-75ms target), jitter, duplicates, delivery (95-99%).
Logging: CSV for snapshots/diagnostics; pcap captures.

7. Example Walkthrough

Two-player game: INIT → SNAPSHOTs → EVENTs → GAMEOVER with scores.

8. Limitations & Future

- Fixed 20 Hz; no encryption.
- Future: Adaptive rates, encryption, more players, use ack.