Mini-RFC: GridClash Binary Protocol (GCBP v1)

Submitted to:

Dr. Karim Ahmed

Eng. Noha Wahdan

Submitted by :

Loaiy Mahmoud 23P0419

Omar Ayman23P0244

Ali Moataz 23P0280

Omar Osama 2300090

Jana Tamer 23P0173

Jana Mohamed 2301032

Mini-RFC: GridClash Binary Protocol (GCBP v1)

1. Introduction

The GridClash Binary Protocol (GCBP v1) is a custom UDP-based application-layer protocol designed to synchronize player actions and global state in a low-latency, loss-tolerant multiplayer game called Grid Clash. The game involves multiple players competing on a shared 2D grid (e.g., 20×20 cells). Players click cells to claim them; conflicts are resolved by the server based on message timestamps. The protocol ensures near real-time synchronization across clients while remaining resilient to packet loss and network jitter. Goals: - Achieve sub-50 ms latency for real-time state updates. - Maintain synchronization consistency across 4 clients. - Handle transient network loss (≤5%) gracefully using selective reliability and redundancy. Assumptions and Constraints: - Transport: UDP (connectionless) - Maximum payload size: ≤ 1200 bytes - Typical update frequency: 20 Hz - No per-packet retransmissions (loss-tolerant design) - Runs cross-platform (Linux/Windows, Python 3)

2. Protocol Architecture

Entities: - Server: Authoritative game state manager. Periodically broadcasts state snapshots and resolves conflicts. - Client: Represents an individual player. Sends actions (e.g., cell acquisition requests) and applies server updates. - Network: Unreliable medium with potential delay, jitter, and loss.

Data Flow Overview:

Client → Server: CONNECT_REQUEST → WELCOME Client → Server:
ACQUIRE_REQUEST(cell_id, timestamp) Server → Client:
ACQUIRE_RESPONSE(success/failure) Server → All: GAME_STATE(snapshot_id,
player_positions) Server → All: GAME_OVER(winner_id) Both sides: HEARTBEAT, ACK/NACK for selective reliability
+---------+ CONNECT_REQUEST +-----------+ | Client | ----------> | Server | | | <----- WELCOME ----------| | | | -----> ACQUIRE_REQUEST -----> | | | | <------ GAME_STATE ----------| | | |
<------ GAME_OVER -----------| | +---------+ +-----------+
Reliability Enhancement: The protocol incorporates selective reliability by sending ACK/NACK messages for critical actions (e.g., ACQUIRE_REQUEST, GAME_OVER), while non-critical updates (e.g., GAME_STATE) rely on redundancy and sequence tracking.

3. Message Formats

All messages are binary-encoded with fixed-size headers for minimal overhead. Messages begin with a 12-byte header, followed by an optional payload.

Field	Size (bytes)	Description
protocol_id	4	ASCII identifier, fixed to 'GRID'
version	1	Protocol version (1)
msg_type	1	Message type ID
snapshot_id	4	Unique snapshot identifier
seq_num	4	Sequential number for deduplication
server_timestamp	8	Milliseconds since epoch (server clock)
payload_len	2	Payload size in bytes
checksum	4 (optional)	CRC32 for message integrity

Message Types:

Type	Code	Direction	Purpose
CONNECT_REQUEST	0x01	Client→Server	Initiate handshake
WELCOME	0x02	Server→Client	Confirm connection, assign ID
ACQUIRE_REQUEST	0x03	Client→Server	Request to claim a cell
ACQUIRE_RESPONSE	0x04	Server→Client	Grant/deny cell acquisition
PLAYER_MOVE	0x05	Client→Server	Update player position
GAME_STATE	0x06	Server→All	Broadcast grid snapshot
GAME_OVER	0x07	Server→All	End-of-game notice and results
ACK	0x08	Both	Confirm receipt of critical packet
NACK	0x09	Both	Request retransmission of critical packet
HEARTBEAT	0x0A	Both	Liveness indicator

Example Header Encoding (Python struct format):

HEADER_FORMAT = '!4s B B I I Q H' # protocol_id, version, msg_type, snapshot_id, seq_num, timestamp, payload_len