RFC – EchoSync Protocol (ESP)

1. Introduction

The EchoSync Protocol (ESP) is a custom UDP-based synchronization protocol built for the real-time multiplayer game "Grid Clash".
It provides low-latency, partially reliable communication between clients and the central game server.
In Grid Clash, players compete on a shared 20×20 grid to claim cells by clicking them.
The server acts as the authoritative source of truth — it receives player actions, resolves conflicts, and continuously broadcasts snapshots representing the current grid state.
ESP avoids TCP because retransmission and congestion control cause latency spikes unsuitable for fast-paced games.
Instead, it builds a lightweight reliability layer on top of UDP, handling:

- **Fragmentation a**nd reassembly of large packets,
- **Sequence-based acknowledgment** and **retransmission**,
- **Periodic synchronization** of state through snapshots and incremental updates.

Assumptions & Constraints:

- Reliability Mechanism: Redundant updates (include last K updates per packet),
- Transport Layer: UDP
- Maximum packet size: ≤ 1200 bytes (fragments larger payloads automatically)
- Update rate: 20–60 Hz
- Expected packet loss: 2–5%
- Target latency: ≤ 50 ms

---

2. Protocol Architecture

The EchoSync Protocol follows a centralized client–server architecture, where the server manages the global game state, and clients synchronize based on snapshots and updates.

2.1 Entities

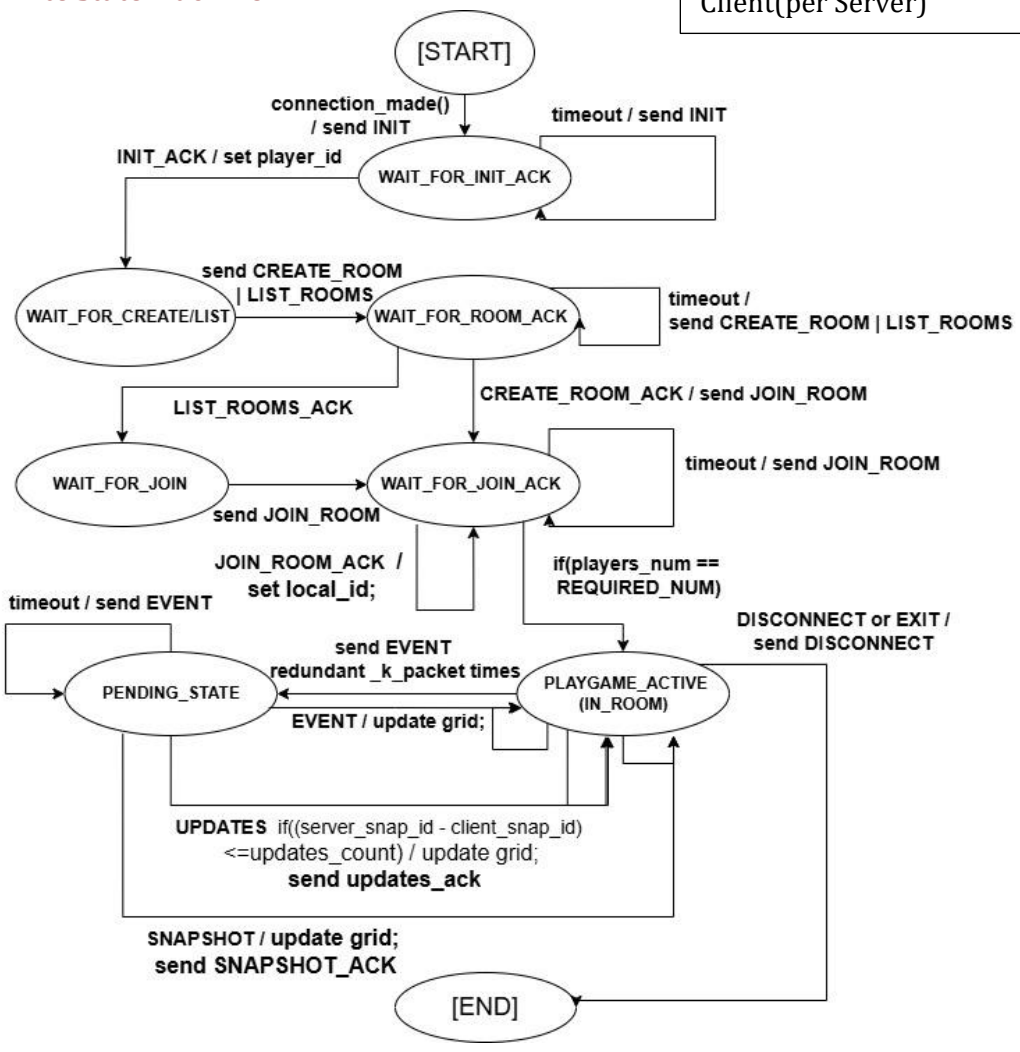
- **Server:**
  - Maintains the authoritative grid state and list of active rooms with up to 4 concurrent players.
  - Assigns each player a unique player_id and manages player registration.
  - Periodically sends UPDATES messages (Most Recent Update) to all players in each room.
  - If client is too late send Sapshot(Complete Grid State)

- Tracks acknowledgments (ACKs) to measure latency and retransmit lost packets if necessary.
- Cleans up disconnected clients and removes inactive rooms automatically.

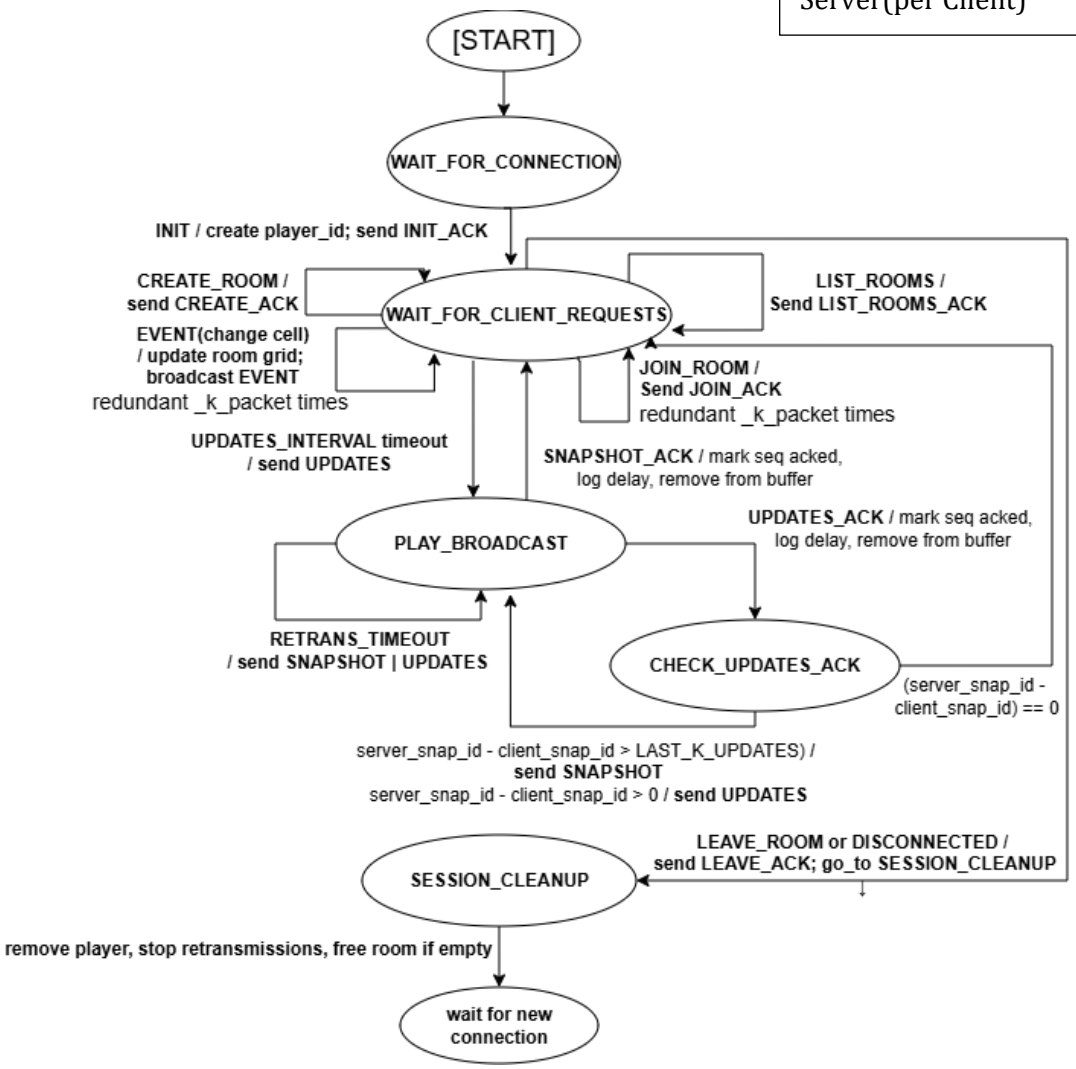
- **Clients:**
  - Connect to the server using an INIT → INIT_ACK handshake.
  - Send CREATE_ROOM or JOIN_ROOM requests to enter a game session.
  - Once inside a room, they render the grid and continuously receive SNAPSHOT and UPDATES packets.
  - On each cell click, they send EVENT messages (cell claim requests).
  - Respond to each SNAPSHOT or UPDATE with an ACK (to confirm receipt and assist in latency tracking).
  - Handle retransmission and fragment reassembly for large payloads.

2.2 Finite State Machine

Client(per Server)



Server(per Client)



3-Message Formats

Each ESP packet is composed of a fixed-length header (32 bytes) followed by a variable-length payload (0–1168 bytes), keeping the total packet size under 1200 bytes to avoid fragmentation. The header ensures correct packet identification, ordering, synchronization, and integrity validation across UDP transport.

3.1 Header Structure(ASCII lay-out)

Field	Size(Bytes)	Offset	Justification	Type(Struct Format)
Protocol_id	**4**	**0-3**	**Constant ASCII `"ESP1"` identifying the protocol version**	**4s**
Version	**1**	**4**	**Currently `1`**	**B**

Msg_type	**1**	**5**	**Message category (e.g., `0=INIT`, `1=INIT_ACK`, `2=CREATE_ROOM`)**	**B**
Snapshot_id	**4**	**6-9**	**Snapshot identifier, used to track game state updates**	**I**
Seq-num	**4**	**10-13**	**Sequence number for ordering and loss detection.**	**I**
Timestamp	**8**	**14 – 21**	**for synchronization and latency measurement**	**Q**
Payload_len	**2**	**22 – 23**	**length of the payload data following the header**	**H**
Packet_id	**4**	**24 – 27**	**unique packet identifier used for retransmission tracking and debugging**	**I**
Checksum	**4**	**28 – 31**	**CRC32 checksum for verifying packet integrity.**	**I**

3.2-Sample message:

This message is an INIT_ACK packet sent by the server to confirm a client's initialization request, acknowledging sequence 42 with integrity verified by a checksum.

Field	**Value**
Protocol_id	b'ESP1'
Version	1
Msg_type	1(INIT_ACK = 1)
Snapshot_id	0
Seq-num	42
Timestamp	173,066,111,234.568 (ms)
Payload_len	8
Packet_Id	1001
Checksum	0x4A8F12CD