RFC Project explanation

Here in this page you will find the explanations of our rfc in our R-Type game. You can also download it in pdf format here

Multiplayer Game Synchronization Protocol (PSJM)

Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

This document is a product of EPITECH Network Programming course. It represents information that the author believes is valuable to share with the community.

Abstract

This document specifies the Multiplayer Game Synchronization Protocol (PSJM), a simple UDP-based protocol for real-time multiplayer games. It facilitates player connections/disconnections, position synchronization, and game state updates. The protocol has been extended to support user authentication, leaderboards, user profiles, level progression, and compressed game state transmission.

Table of Contents

1. Introduction

The Multiplayer Game Synchronization Protocol (PSJM) is a simple UDP-based protocol for real-time multiplayer games. It supports:

- Player connections/disconnections
- Position synchronization
- Game state updates
- User authentication (login/register)
- Leaderboard system
- User profiles and statistics
- Level progression

- Compressed data transmission

2. Packet Format

All packets have a fixed 11-byte header (HEADER_SIZE):

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Magic      | Client ID     |      Sequence Number          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       (Sequence Number cont'd)                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Packet Type  |           Payload Length                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         (Payload Length cont'd)           |    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
|                  Payload (variable)                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Fields: - Magic Number (1 byte): 0x93 - Packet validation identifier - Client ID (1 byte): Unique identifier assigned by server - Sequence Number (4 bytes): Packet ordering for tracking/lost detection - Packet Type (1 byte): Type identifier (see section 3) - Payload Length (4 bytes): Length of payload in bytes - Payload (variable): Packet-specific data

Header (11 bytes total): - Magic Number (1 byte) - Client ID (1 byte) - Sequence Number (4 bytes) - Packet Type (1 byte) - Payload Length (4 bytes)

Body: - Payload (variable length, specified in header)

3. Packet Types Client and Server packet types (complete list used by the codebase):

Value	Name	Description
0x00	NO_OP_PACKET	No operation / keep-alive
0x01	CONNECTION_CLIENT_PACKET	Client connection request
0x02	ACCEPTATION_PACKET	Server acceptance / assign client ID
0x03	DISCONNECTION_PACKET	Client disconnection
0x04	EVENT_PACKET	Client input/event
0x05	END_GAME_PACKET	Server notifies end of game
0x06	CAN_START_PACKET	Server tells clients they can start
0x07	CLIENT_READY_PACKET	Client signals ready state
0x08	SPAWN_PLAYER_PACKET	Server spawns a player/entity
0x09	DEATH_PLAYER_PACKET	Server notifies a player/entity death
0x0A	WHOAMI_PACKET	Optional identification/resync packet
0x0B	SERVER_STATUS_PACKET	Server sends lobby status information
0x0C	REQUEST_LOBBY_PACKET	Client send request to create lobby
0x0D	SEND_LOBBY_CODE_PACKET	Server sends the code to whom requested

0x0E	CONNECT_TO_LOBBY	Client connect to an existing lobby
0x0F	LOBBY_MASTER_REQUEST_START	Client that created lobby starts the game
0x10	LOBBY_CONNECT_VALUE	Return success or failure of connection
0x11	LEVEL_COMPLETE_PACKET	Server notifies level completion
0x12	NEXT_LEVEL_PACKET	Server notifies next level transition
0x13	REGISTER_PACKET	Client registration request
0x14	CONNECT_USER_PACKET	Server confirms user authentication
0x15	LOGIN_PACKET	Client login request
0x16	GAME_STATE_BATCH_PACKET	Server batched game state update
0x17	GAME_STATE_BATCH_COMPRESSED_PKT	Compressed batched game state
0x18	GAME_STATE_COMPRESSED_PACKET	Compressed game state update
0x19	REQUEST_LEADERBOARD_PACKET	Client requests leaderboard data
0x1A	LEADERBOARD_PACKET	Server sends leaderboard information
0x1B	REGISTER_FAIL_PACKET	Server notifies registration failure
0x1C	REQUEST_PROFILE_PACKET	Client requests user profile data
0x1D	PROFILE_PACKET	Server sends user profile information
0x1E	GAME_RULES_PACKET	Server sends current game rules to client
0x1F	REQUEST_GAME_RULES_UPDATE_PACKET	Client requests an update for game rules
0x20	NEW_CHAT_PACKET	Client sends a chat message
0x21	BROADCASTED_CHAT_PACKET	Server broadcasts chat message to all
0x22	FORCE_LEAVE_PACKET	Server forces client to leave lobby
0x23	LEAVE_LOBBY_PACKET	Client leaves lobby
0x24	ACK_LEAVE_LOBBY	Server acknowledges lobby leave

4. Packet Details

4.1 Client Details

4.1.1 CONNECTION_CLIENT_PACKET (0x01) – Sent from client to server

- Empty payload (no player name sent at connection)
- Fixed length: `LENGTH_CONNECTION_PACKET` (0 bytes)

4.1.2 DISCONNECTION_PACKET (0x03) – Client requests to disconnect

- Player ID (1 byte)
- Fixed length: `LENGTH_DISCONNECTION_PACKET` (1 byte)

4.1.3 EVENT_PACKET (0x04) – Client notifies input

- Event type (1 byte, e.g., Up, Down, Left, Right, Space)
- Additional event data (e.g., movement depth as double, 8 bytes)
- Fixed length: `LENGTH_EVENT_PACKET` (9 bytes)

4.1.4 CLIENT_READY_PACKET (0x07) – Client signals it is ready

- Used by client to indicate readiness prior to start
- Empty payload

4.1.5 REQUEST_LOBBY_PACKET (0x0C) - Client request a game code

- Empty payload
- Fixed length: `LENGTH_REQUEST_LOBBY_PACKET` (0 bytes)

4.1.6 CONNECT_TO_LOBBY (0x0E) - Client send a request to connect to a lobby

- Payload contains the lobby code
- Fixed length: `LENGTH_CONNECT_TO_LOBBY_PACKET` (1 byte)

4.1.7 LOBBY_MASTER_REQUEST_START (0x0F) - Client that created the lobby can start the game

- Payload contains the lobby code (8 bytes string)
- Variable length payload

4.1.8 REGISTER_PACKET (0x13) – Client registration request

- Username (8 bytes) and password (8 bytes) for account creation
- Passwords are encrypted using XOR encryption with base64 encoding
- Fixed length: `LENGTH_REGISTER_PACKET` (16 bytes)

4.1.9 LOGIN_PACKET (0x15) – Client login request

- Username (8 bytes) and password (8 bytes) for authentication
- Fixed length: `LENGTH_LOGIN_PACKET` (16 bytes)

4.1.10 REQUEST_LEADERBOARD_PACKET (0x19) – Client requests leaderboard data

- Empty payload
- Fixed length: `LENGTH_REQUEST_LEADERBOARD_PACKET` (0 bytes)

4.1.11 REQUEST_PROFILE_PACKET (0x1C) – Client requests user profile data

- Empty payload
- Fixed length: `LENGTH_REQUEST_PROFILE_PACKET` (0 bytes)

4.1.12 REQUEST_GAME_RULES_UPDATE_PACKET (0x1F) – Client requests an update for game rules

- Rule type (1 byte): 0=gamemode, 1=difficulty, 2=crossfire
- Value (1 byte): cycles through available options
- Fixed length: `LENGTH_REQUEST_GAME_RULES_UPDATE_PACKET` (2 bytes)

4.1.13 NEW_CHAT_PACKET (0x20) – Client sends a chat message

- Message content (variable length string)
- Variable length payload

4.1.14 LEAVE_LOBBY_PACKET (0x23) – Client leaves lobby

- Empty payload
- Fixed length: 0 bytes

4.2 Server Details

4.2.1 ACCEPTATION_PACKET (0x02) – Sent from Server to Client (connection accept)

- Player ID assigned by server (1 byte)
- Fixed length: `LENGTH_ACCEPTATION_PACKET` (1 byte)

4.2.2 END_GAME_PACKET (0x05) – Server notifies end of game

- Empty payload
- Fixed length: `LENGTH_END_GAME_PACKET` (0 bytes)

4.2.3 CAN_START_PACKET (0x06) – Server tells clients the game can start

- Empty payload
- Server broadcasts to all ready clients

4.2.4 SPAWN_PLAYER_PACKET (0x08) – Server spawns a player/entity

- Payload includes entity data required for client to instantiate the entity
- Variable length depending on entity type and components

4.2.5 DEATH_PLAYER_PACKET (0x09) – Server notifies a player/entity death

- Entity ID (8 bytes, uint64_t) identifying the dead entity
- Fixed length: `LENGTH_DEATH_PACKET` (8 bytes)

4.2.6 WHOAMI_PACKET (0x0A) – Optional identification / resynchronization packet

- May be used to request/confirm identification or small resync actions
- Fixed length: `LENGTH_WHOAMI_PACKET` (0 bytes)

4.2.7 SERVER_STATUS_PACKET (0x0B) – Server sends lobby status information

- Connected clients count (8 bytes, uint64_t)
- Ready clients count (8 bytes, uint64_t)
- Client ID (8 bytes, uint64_t)
- Client ready status (8 bytes, uint64_t, 0=not ready, 1=ready)
- Fixed length: `LENGTH_SERVER_STATUS_PACKET` (32 bytes)
- Sent periodically to keep clients updated on lobby state

4.2.8 SEND_LOBBY_CODE_PACKET (0x0D) Server sends the lobby code to the ‘master’ of the game

- Payload contains the lobby code (8 bytes string)
- Fixed length: `LENGTH_LOBBY_CODE_PACKET` (8 bytes)

4.2.9 LOBBY_CONNECT_VALUE (0x10) Server says to the client if the connection to the lobby was successful or not

- Payload contains char: ‘t’ for success, ‘f’ for failure

- Fixed length: `LENGTH_CONNECT_TO_LOBBY_PACKET` (1 byte)

4.2.10 LEVEL_COMPLETE_PACKET (0x11) – Server notifies level completion

- Indicates that the current level has been completed
- Empty payload

4.2.11 NEXT_LEVEL_PACKET (0x12) – Server notifies next level transition

- Indicates transition to the next level
- Empty payload

4.2.12 CONNECT_USER_PACKET (0x14) – Server confirms user authentication

- Username (8 bytes)
- Fixed length: `LENGTH_CONNECT_USER_PACKET` (8 bytes)

4.2.13 GAME_STATE_BATCH_PACKET (0x16) – Server sends batched game state update

- Contains multiple entity states in a single packet
- Variable length depending on number of entities and components

4.2.14 GAME_STATE_BATCH_COMPRESSED_PACKET (0x17) – Compressed batched game state

- LZ4 compressed version of batched game state
- Significantly reduces bandwidth usage
- Variable length

4.2.15 GAME_STATE_COMPRESSED_PACKET (0x18) – Compressed game state update

- LZ4 compressed single game state update
- Variable length

4.2.16 LEADERBOARD_PACKET (0x1A) – Server sends leaderboard information

- Contains top 10 player rankings with usernames and high scores
- Fixed length: `LENGTH_LEADERBOARD_PACKET` (160 bytes)
- Format: 10 entries × (8 bytes username + 8 bytes score)

4.2.17 REGISTER_FAIL_PACKET (0x1B) – Server notifies registration failure

- Sent when username already exists
- Fixed length: `LENGTH_FAIL_REGISTER_PACKET` (0 bytes)

4.2.18 PROFILE_PACKET (0x1D) – Server sends user profile information

- Contains user statistics:
  - Username (8 bytes)
  - Wins (8 bytes, uint64_t)
  - High score (8 bytes, uint64_t)

    - Games played (8 bytes, uint64_t)
    - Time spent (8 bytes, uint64_t)
- Fixed length: `LENGTH_PROFILE_PACKET` (40 bytes)

4.2.19 GAME_RULES_PACKET (0x1E) – Server sends current game rules to clients

- Payload contains serialized game rules:
    - Gamemode (1 byte): 0=Classic, 1=Infinite
    - Difficulty (1 byte): 0=Easy, 1=Normal, 2=Hard
    - Crossfire enabled (1 byte): 0=No, 1=Yes
- Fixed length: `LENGTH_GAME_RULES_PACKET` (3 bytes)

4.2.20 BROADCASTED_CHAT_PACKET (0x21) – Server broadcasts chat message to all clients

- Contains sender username (8 bytes) and message content (variable)
- Variable length payload

4.2.21 FORCE_LEAVE_PACKET (0x22) – Server forces client to leave

- Reason code (1 byte): 0=Closed, 1=Kicked, 2=Banned
- Fixed length: 1 byte

4.2.22 ACK_LEAVE_LOBBY (0x24) – Server acknowledges lobby leave

- Empty payload
- Fixed length: 0 bytes

Notes: - The canonical constant names and packet lengths are defined in `common/interfaces/IPacketManager.hpp`. - The header size is fixed at 11 bytes (HEADER_SIZE constant). - Magic number for packet validation is 0x93 (MAGIC_NUMBER constant). - Passwords are encrypted using XOR-based encryption with base64 encoding for secure storage. - Chat system supports real-time messaging between authenticated clients. - Game rules can be modified by lobby master during waiting phase.

5. Communication Example

```
+------------------+                    +------------------+
|                  |                    |                  |
|      Client      |                    |      Server      |
|                  |                    |                  |
+------------------+                    +------------------+
        |                                        |
        |           CONNECTION_CLIENT            |
        |--------------------------------------->|
        |                                        |
        |             ACCEPTATION                |
        |<---------------------------------------|
        |                                        |
        |               CLIENT_READY               |
        |----------------------------------------->|
        |                                          |
        |               SERVER_STATUS              |
        |<-----------------------------------------|
        |                                          |
        |                CAN_START                 |
        |<-----------------------------------------|
        |                                          |
        |============= Game Loop Begins ===========|
        |                                          |
        |                  EVENT                   |
        |----------------------------------------->|
        |                                          |
        |               GAME_STATE                 |
        |<-----------------------------------------|
        |                                          |
        |                END_GAME                  |
        |<-----------------------------------------|
        |                                          |
        |              DISCONNECTION               |
        |----------------------------------------->|
```

5.1 Ready System Logic

After connection establishment, clients must signal readiness before the game begins. This ensures all players start simultaneously.

Flow: 1. Client connects and receives ACCEPTATION 2. Client loads necessary resources and displays ready interface 3. Client sends CLIENT_READY when player indicates readiness 4. Server tracks readiness status for each client 5. Server periodically sends SERVER_STATUS to update lobby information 6. When all connected clients are ready, server broadcasts CAN_START 7. Game loop begins with synchronized start

Benefits: - Synchronized game starts across all clients - Prevents clients from starting prematurely - Provides lobby status updates

6. Packet lost consideration

6.1 Tracking :

```
To avoid and track easier what was lost, each packet is numbered
and assigned to a user so that the server can now when a package
was lost.
```

6.2 Rollback :

```
The client will have an interpalation logic, so that if needed he
can predict and advance until the server (the absolute truth),
```

`sends a new packet or starts reponding again.`

7. Technical Considerations

- Encoding: UTF-8 text
- Number format: Network order (big-endian)
- Compression: LZ4 compression for game state packets to reduce bandwidth usage

8. Map Format Protocol

8.1. File Formatting

- File type : File containing the map must be a .json

8.2. Map Format

- Element type ?

8.3. Map Rendering

- One elem = ?x? pixel square

9. References

[RFC7322] Flanagan, H. and S. Ginoza, "RFC Style Guide", RFC 7322, DOI 10.17487/RFC7322, September 2014, rfc.

10. Author's Address

Matisse Marsac EPITECH Email: matisse.marsac@epitech.eu

Albane Merian EPITECH Email: albane.merian@epitech.eu

Marin Lamy EPITECH Email: marin.lamy@epitech.eu

Eliott Tesnier EPITECH Email: eliott.tesnier@epitech.eu

Alban Roussee EPITECH Email: alban.roussee@epitech.eu