Request for Comments: P2P Pokémon Battle Protocol (PokeProtocol) over UDP

Document Type: Protocol Specification
Status: Draft

Abstract

This document specifies the Peer-to-Peer Pokémon Battle Protocol (PokeProtocol) using UDP as its transport layer, with an added feature for peer-to-peer text chat that now includes an option for sending stickers. This protocol is designed for conducting a turn-based Pokémon battle between two independent peers over a network. It defines a structured messaging format and a state machine, incorporating a custom reliability layer to ensure a synchronized view of the battle and reliable delivery of chat messages despite UDP's connectionless nature. This revision introduces a granular, four-step handshake for each turn, requiring explicit agreement on attack, defense, and calculation to prevent discrepancies. It also introduces three distinct communication roles: a **Peer-to-Peer** mode for direct one-to-one communication, a **Broadcast** mode for announcing battle availability on a local network, and a new **Spectator** role. Finally, it refines the damage calculation formula to include separate **Special Attack** and **Special Defense** stats. Players can use a limited number of special attack and special defense boosts during the battle, which are defined during the setup phase.

1. Introduction

The objective of this PokeProtocol revision is to provide a low-overhead, platform-agnostic, and unambiguous method for simulating Pokémon battles in two applications. By leveraging UDP, this protocol minimizes latency by forgoing TCP's built-in reliability and flow control. Instead, it addresses the challenges of game state synchronization in a decentralized environment by implementing a custom reliability and ordering mechanism at the application layer. The addition of a chat feature, which now supports stickers, enhances the interactive experience without impacting the core battle mechanics.

2. Terminology

- **Peer:** An application instance participating in a battle.
- **Host Peer:** The peer that initiates the connection and game. It is responsible for listening for datagrams from the Joiner Peer.
- **Joiner Peer:** The peer that sends a connection request to the Host Peer to start the battle.
- **Spectator Peer:** A peer that joins an active battle to observe. Spectators can send and receive chat messages but cannot influence the battle state.
- **Battle State:** The collective information representing the current state of the battle, including Pokémon health, status effects, and turn order.

3. Protocol Architecture

The PokeProtocol over UDP is built on a peer-to-peer architecture using UDP datagrams.

- **Transport Layer: UDP** (User Data Protocol) MUST be used.
- **Communication Modes:** This protocol supports three primary communication modes:
  - **Peer-to-Peer Mode:** Messages are sent directly from one peer to another using a known IP address and port. This is the standard mode for all battle-related messages once the connection is established.
  - **Broadcast Mode:** Messages are sent to a broadcast address (e.g., 255.255.255.255) to be received by all peers on a local network. This mode can be used for initial peer discovery or for announcing a game's presence.
  - **Spectator Mode:** A peer can join an existing battle as a spectator. They receive all battle and chat messages, but are restricted from sending battle-related commands. They can, however, send CHAT_MESSAGEs.
- **Connection Process:** Since UDP is connectionless, a logical connection must be established.
  1. The Host Peer creates a UDP socket and begins listening on a designated port.
  2. The Joiner Peer creates a UDP socket and sends a HANDSHAKE_REQUEST message to the Host Peer's IP address and port. A Spectator Peer would send a SPECTATOR_REQUEST instead.
  3. The Host Peer, upon receiving a request, responds with a HANDSHAKE_RESPONSE message.
  4. All peers are now considered logically connected and ready for their respective roles.

4. Message Format and Types

All communication between peers MUST be in a plain text format, with a single message per UDP datagram. Messages are composed of a series of newline-separated key: value pairs. The message_type field is mandatory for all messages.

4.1. HANDSHAKE_REQUEST

This message is sent by the Joiner Peer to initiate a logical connection as a player.

Format:

```
Shell
message_type: HANDSHAKE_REQUEST
```

4.2. HANDSHAKE_RESPONSE

This message is sent by the Host Peer to acknowledge a connection request.

- message_type: "HANDSHAKE_RESPONSE"
- seed: A random integer that will be used to seed the random number generator on both peers to ensure synchronized damage calculation.

Format:

```
Shell
message_type: HANDSHAKE_RESPONSE
seed: 12345
```

4.3. SPECTATOR_REQUEST

This message is sent by a peer to initiate a logical connection as a spectator.

Format:

```
Shell
message_type: SPECTATOR_REQUEST
```

4.4. BATTLE_SETUP

This message is sent by both playing peers after the handshake to exchange initial Pokémon data and define the communication mode for the battle.

- message_type: "BATTLE_SETUP"
- communication_mode: A string indicating the mode of communication for the battle. Valid values are "P2P" and "BROADCAST".
- pokemon_name: The name of the peer's Pokémon.
- stat_boosts: An object containing the player's allocation of a limited number of special attack and special defense uses.
- pokemon: An object containing the peer's chosen Pokémon data.

Format:

```
Shell
message_type: BATTLE_SETUP
communication_mode: P2P
pokemon_name: Pikachu
stat_boosts: { "special_attack_uses": 5, "special_defense_uses": 5 }
```

4.5. ATTACK_ANNOUNCE

This message is sent by the peer whose turn it is to announce their move choice.

- message_type: "ATTACK_ANNOUNCE"
- move_name: The name of the chosen move.
- sequence_number: A unique, monotonically increasing integer for this message.

Format:

```
Shell
message_type: ATTACK_ANNOUNCE
move_name: Thunderbolt
sequence_number: 5
```

4.6. DEFENSE_ANNOUNCE

This message is sent by the defending peer to acknowledge the opponent's ATTACK_ANNOUNCE and signal readiness to process the turn.

- message_type: "DEFENSE_ANNOUNCE"
- sequence_number: A unique, monotonically increasing integer for this message.

Format:

```shell
message_type: DEFENSE_ANNOUNCE
sequence_number: 6
```

4.7. CALCULATION_REPORT

This message is sent by both players to report the results of their independent damage calculation for the turn.

- message_type: "CALCULATION_REPORT"
- attacker: The name of the Pokémon that attacked.
- move_used: The name of the move that was used.
- remaining_health: The remaining health of the attacking Pokémon.
- damage_dealt: The amount of damage inflicted.
- defender_hp_remaining: The defender's remaining HP after the attack.
- status_message: A descriptive string of the turn's events (e.g., "Pikachu used Thunderbolt! It was super effective!").
- sequence_number: A unique, monotonically increasing integer for this message.

Format:

```shell
message_type: CALCULATION_REPORT
attacker: Pikachu
move_used: Thunderbolt
remaining_health: 90
damage_dealt: 80
defender_hp_remaining: 20
status_message: Pikachu used Thunderbolt! It was super effective!
sequence_number: 7
```

4.8. CALCULATION_CONFIRM

This message is sent by a player to confirm that their opponent's CALCULATION_REPORT matches their own.

- message_type: "CALCULATION_CONFIRM"
- sequence_number: A unique, monotonically increasing integer for this message.

Format:

```shell
message_type: CALCULATION_CONFIRM
sequence_number: 8
```

4.9. RESOLUTION_REQUEST

This message is sent by a player when a calculation discrepancy is detected. It contains the sender's calculated values for the turn.

- message_type: "RESOLUTION_REQUEST"
- attacker: The name of the Pokémon that attacked.
- move_used: The name of the move that was used.
- damage_dealt: The amount of damage inflicted as calculated by the sending peer.
- defender_hp_remaining: The defender's remaining HP as calculated by the sending peer.
- sequence_number: A unique, monotonically increasing integer for this message.

Format:

```shell
message_type: RESOLUTION_REQUEST
attacker: Pikachu
move_used: Thunderbolt
damage_dealt: 80
defender_hp_remaining: 20
sequence_number: 9
```

4.10. GAME_OVER

This message is sent by a player when their opponent's Pokémon has fainted.

- message_type: "GAME_OVER"
- winner: The name of the Pokémon that won.
- loser: The name of the Pokémon that fainted.
- sequence_number: A unique, monotonically increasing integer for this message.

```
Shell
message_over: GAME_OVER
winner: Pikachu
loser: Charmander
sequence_number: 10
```

4.11. CHAT_MESSAGE

This message is used for sending a plain text chat message or a sticker between peers. It can be sent by any peer (Host, Joiner, or Spectator) at any time, independently of the turn-based battle logic.

- message_type: "CHAT_MESSAGE"
- sender_name: The name of the peer sending the message.
- content_type: A string that specifies the content type of the message. Valid values are "TEXT" or "STICKER".
- message_text: (Optional) The content of the chat message, used when content_type is "TEXT".
- sticker_data: (Optional) A Base64 encoded string of the sticker data. The sticker dimensions MUST be 320px x 320px and the file size should be less than 10MB. Implementations should be aware that sending large messages (over 1.5 KB) over UDP may require fragmentation at the IP layer, which can affect network performance and reliability.
- sequence_number: A unique, monotonically increasing integer for this message.

Format:

// Example of a text message

```
Shell
message_type: CHAT_MESSAGE
sender_name: Player1
content_type: TEXT
message_text: Good luck, have fun!
sequence_number: 11
```

```
// Example of a sticker message (abbreviated Base64 data)
message_type: CHAT_MESSAGE
sender_name: Player2
content_type: STICKER
sticker_data: iVBORw0KGgoAAAANSUhEUgAAB...
sequence_number: 12
```

5. State Management and Game Flow

The battle follows a simple state machine, enhanced with a reliability layer. Chat messages are handled as an asynchronous overlay to this state machine.

5.1. Reliability Layer

To handle UDP's unreliable nature, the following is required:

- **Sequence Numbers:** Every non-ACK message MUST include a sequence_number.
- **Acknowledgements:** Upon receiving a message with a sequence number, the peer MUST send an ACK message with the corresponding ack_number.
- **Retransmission:** If a peer sends a message but does not receive an ACK within a predefined timeout, it MUST retransmit the message. A retransmission counter should be maintained for each message. If the maximum number of retries is reached without receiving an ACK, the peer SHOULD assume the connection has been lost and terminate the battle. The recommended timeout is **500 milliseconds**, and the recommended maximum number of retries is **3**.

5.2. Game Flow

1. **Initial State (SETUP):**
    - Host and Joiner Peers connect via the handshake process. The Host sends a HANDSHAKE_RESPONSE with a seed. Both peers use this seed for their random number generators.
    - Both playing Peers send a BATTLE_SETUP message with their chosen Pokémon data and the desired communication_mode.
    - Upon successfully exchanging BATTLE_SETUP messages, each peer transitions to the WAITING_FOR_MOVE state. Spectator Peers, upon connecting, receive all battle messages but do not take turns.
2. **Turn-Based State (WAITING_FOR_MOVE):**
    - The Host Peer is designated to go first.
    - The acting peer sends an ATTACK_ANNOUNCE message with a new sequence number.

- The defending peer must wait for a valid ATTACK_ANNOUNCE message. Upon receiving it, the defender sends a DEFENSE_ANNOUNCE message, confirming receipt and signaling readiness for the next phase.
- Upon receiving the DEFENSE_ANNOUNCE message, both players independently apply the damage and transition to the PROCESSING_TURN state.

3. **Turn Processing State (PROCESSING_TURN):**
   - Each player independently performs the damage calculation, using the appropriate attack and defense stats based on the move's damage_category.
   - Each player then sends a CALCULATION_REPORT message with a new sequence number to the other player. This message acts as a checksum.
   - **Discrepancy Resolution:**
     - If the received CALCULATION_REPORT **matches** the peer's local calculation, the peer sends a CALCULATION_CONFIRM message, and the turn order reverses, returning to the WAITING_FOR_MOVE state. The new field remaining_health can be used to perform additional integrity checks on the battle state.
     - If the received CALCULATION_REPORT **does not match** the peer's local calculation, the peer MUST send a RESOLUTION_REQUEST message containing its own calculated values. It should then wait for a response.
     - The other peer, upon receiving a RESOLUTION_REQUEST, must re-evaluate the turn based on the received data. If it agrees with the RESOLUTION_REQUEST's values, it sends an ACK and updates its state to match. If it still disagrees, this indicates a fundamental error, and the battle SHOULD terminate.
4. **Chat Functionality:**
   - A peer can send a CHAT_MESSAGE at any time, regardless of the current battle state.
   - Chat messages are handled separately from the turn-based logic.
   - Upon receiving a CHAT_MESSAGE, the peer should display the message to the user and send an ACK to confirm receipt.
5. **Final State (GAME_OVER):**
   - When a Pokémon's HP drops to zero or below, the peer whose opponent has fainted sends a GAME_OVER message.
   - This message also requires a sequence number and acknowledgment.
   - Upon receiving this message, the battle ends.

6. Damage Calculation

To ensure synchronization, both peers MUST use the exact same damage calculation formula. The formula uses the appropriate attack and defense stats depending on the move's damage_category.

$$Damage = \frac{BasePower \times AttackerStat \times Type1Effectiveness \times Type2Effectiveness}{DefenderStat}$$

- **AttackerStat**: The attacking Pokémon's **Attack** stat for physical moves, or **SpecialAttack** stat for special moves.
- **DefenderStat**: The defending Pokémon's **PhysicalDefense** stat for physical moves, or **SpecialDefense** stat for special moves.
- **BasePower**: The power attribute of the move.
- **Type1Effectiveness**: A multiplier based on the attacking move's type against the defending Pokémon's primary type.
- **Type2Effectiveness**: A multiplier based on the attacking move's type against the defending Pokémon's secondary type. If the Pokémon only has one type, this value is 1.0.
- **PhysicalDefense**: The physical defense stat of the defending Pokémon.
- **SpecialAttack**: The special attack stat of the attacking Pokémon.
- **SpecialDefense**: The special defense stat of the defending Pokémon.
- **stat_boosts**: An object containing the player's allocation of a limited number of special attack and special defense uses. These are a consumable resource for the player during the battle.

The given Pokémon stats can be found in the given CSV file.

Example: If a Fire-type move attacks a dual-type Pokémon that is Grass/Water:

- The move is super-effective against the Grass type (2.0).
- The move is not very effective against the Water type (0.5).
- The final TypeEffectiveness multiplier would be 2.0x0.5=1.0.