

Network Working Group
 Request for Comments: 0001
 Category: Experimental

R-Type Project Team
 EPITECH
 November 2025

R-Type Game Protocol (RTGP)
 Version 1.1.0 (Reliability Update)

Abstract

This document specifies the binary application-layer protocol used for real-time communication between the R-Type Client and Server. It defines a lightweight datagram-oriented protocol prioritizing low latency and bandwidth efficiency, and specifies a selective reliability layer (RUDP) over UDP to ensure critical game events are delivered.

Status of This Memo

This document is a project specification intended for implementation and review. It is not an IETF RFC and does not constitute an Internet Standards Track specification.

Copyright Notice

Copyright (c) 2025 R-Type Project Team. All rights reserved.

Table of Contents

1. Introduction	2
2. Terminology & Conventions	2
3. Transport Layer	2
4. Packet Structure	3
4.1. Header Format	3
4.2. User ID Conventions	4
4.3. Reliability Mechanism (Flags)	4
5. Protocol Operations (OpCodes)	4
5.1. Session Management	4
5.2. Gameplay & Entity Management	5
5.3. Input & Reconciliation	6
6. Security Considerations	6
7. Future Extensions	7

1. Introduction

The R-Type Game Protocol (RTGP) is a lightweight, binary, datagram-oriented protocol designed to facilitate real-time multiplayer gameplay. It prioritizes low latency and bandwidth efficiency while providing a selective reliability mechanism (RUDP).

to ensure critical game events are delivered without the overhead of TCP.

2. Terminology & Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

2.1. Data Types

```
Byte:     8-bit unsigned integer.
uint16:   16-bit unsigned integer.
uint32:   32-bit unsigned integer.
float:    32-bit IEEE 754 floating point.
String:   NOT SUPPORTED in standard packets to avoid allocation
          overhead, unless specified in the payload.
```

2.2. Byte Order

All multi-byte numeric fields MUST be transmitted in Network Byte Order (Big-Endian). Implementations on Little-Endian architectures (x86/x64) MUST convert data before transmission (htons, htonl) and after reception (ntohs, ntohl).

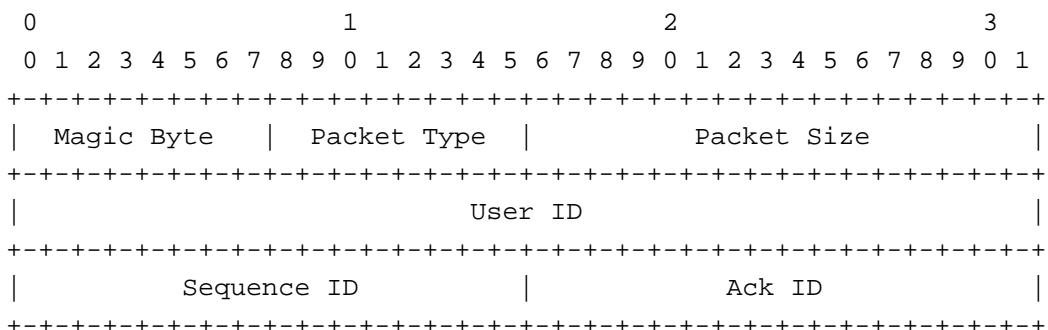
3. Transport Layer

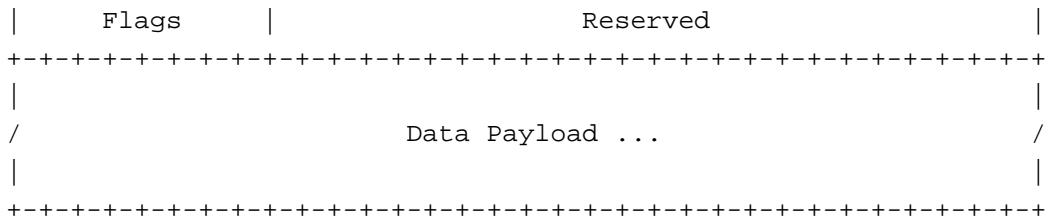
```
Protocol:      UDP (User Datagram Protocol).
Default Port: 4242.
MTU Safety:   The total packet size (Header + Payload) SHOULD NOT
               exceed 1400 bytes to avoid IP fragmentation on
               standard networks.
```

4. Packet Structure

Every RTGP packet consists of a fixed 16-byte Header followed by a variable-length Data Payload.

4.1. Header Format





Field Definitions:

Magic Byte: uint8. MUST be 0xA1. Used to filter spurious traffic.

Packet Type: uint8. The Operation Code (OpCode) defined in Sec 5.

Packet Size: uint16. Length of the Data Payload in bytes. Excludes Header size.

User ID: uint32. The sender's unique identifier.

Sequence ID: uint16. Incremental ID of the packet sent. Wraps at 65535.

Ack ID: uint16. The Sequence ID of the last packet successfully received.

Flags: uint8. Bitmask for packet attributes (see 4.3).

Reserved: 3 bytes. Padding for 16-byte alignment. MUST be 0.

4.2. User ID Conventions

Server Authority: 0xFFFFFFFF (-1). Only the Server uses this.

Unassigned Client: 0x00000000. Used during handshake.

Assigned Client: 0x00000001 to 0xFFFFFFFFFE.

4.3. Reliability Mechanism (Flags)

The Flags field is used to manage the reliability layer (RUDP).

Flag Bitmask Values:

0x01 (RELIABLE): The sender requests an acknowledgement for this packet. The receiver MUST acknowledge this packet (either via a dedicated ACK or piggybacking).

0x02 (IS_ACK): The Ack ID field in this header is valid and acknowledges a previously received packet.

Behavior:

1. Sequence ID: MUST be incremented by 1 for every new packet sent.
2. Ack ID: MUST always contain the Sequence ID of the last valid packet received from the remote peer.
3. Retransmission: If a packet marked RELIABLE is not acknowledged within a specific timeout (e.g., 200ms), the sender MUST retransmit it.

5. Protocol Operations (OpCodes)

5.1. Session Management

0x01 - C_CONNECT

Sender: Client
 Reliability: RELIABLE
 Description: Request to establish a connection.
 Payload: Empty.

0x02 - S_ACCEPT

Sender: Server
 Reliability: RELIABLE
 Description: Connection accepted. Assigns User ID.
 Payload: New User ID (uint32).

0x03 - DISCONNECT

Sender: Client OR Server
 Reliability: RELIABLE
 Description: Graceful termination of the session.
 Payload: Empty.

0x04 - C_GET_USERS

Sender: Client
 Reliability: RELIABLE
 Description: Request a list of currently connected players.
 Payload: Empty.

0x05 - R_GET_USERS

Sender: Server
 Reliability: RELIABLE
 Description: Response to C_GET_USERS.
 Payload: Count (uint8), UserIDs (uint32[]).

0x06 - S_UPDATE_STATE

Sender: Server
 Reliability: RELIABLE
 Description: Notifies clients of a global game state change.
 Payload: StateID (uint8): 0=Lobby, 1=Running, 2=Paused,
 3=GameOver.

5.2. Gameplay & Entity Management

0x10 - S_ENTITY_SPAWN

Sender: Server

Reliability: RELIABLE (Critical)

Description: Instructs clients to instantiate a game object.

Payload:

Entity ID (uint32)

Type (uint8): 0=Player, 1=Bydos, 2=Missile.

PosX (float)

PosY (float)

0x11 - S_ENTITY_MOVE

Sender: Server

Reliability: UNRELIABLE

Description: Regular state update.

Payload:

Entity ID (uint32)

PosX (float), PosY (float)

VelX (float), Vely (float)

0x12 - S_ENTITY_DESTROY

Sender: Server

Reliability: RELIABLE

Description: Instructs clients to remove an entity.

Payload: Entity ID (uint32).

5.3. Input & Reconciliation

0x20 - C_INPUT

Sender: Client

Reliability: UNRELIABLE

Description: Client input state.

Payload: Input Mask (uint8): 0x01=UP, 0x02=DOWN, 0x04=LEFT,
0x08=RIGHT, 0x10=SHOOT.

0x21 - S_UPDATE_POS (Reconciliation)

Sender: Server

Reliability: UNRELIABLE

Description: Correction of client position (anti-drift).

Payload: Authoritative X (float), Authoritative Y (float).

6. Security Considerations

1. Header Validation: Any packet where Header[0] != 0xA1 MUST be silently dropped.
2. Sequence Validation: Packets with a Sequence ID significantly older than the last received ID SHOULD be discarded to prevent replay attacks and processing old state.

3. Spoofing Protection: The Server MUST verify User ID against IP/Port.
4. Authority Check: Clients MUST ignore packets claiming to be 0xFFFFFFFF (Server) if they do not originate from the known Server IP endpoint.

7. Future Extensions

Ping/Pong: OpCodes 0xF0 (Ping) and 0xF1 (Pong) reserved for latency calculation.

Packet Fragmentation: Not currently supported. Payloads exceeding MTU must be handled at the application logic level or split into multiple entities.