R-Type Group J. Aveline

Request for Comments: 4242 R-Tacos, Inc.

Category: Video Games October 2016

R-Type Protocol (RTP) Transport over UDP and TCP

Abstract

This document describes how the R-Type Protocol (RTP), works using and   
 UDP and TCP connection.

This protocol requires use of a R-Type server and at least one R-Type   
 client.

This protocol is using binary communication with the simplest client   
 being any socket program capable of connecting to the server.

The RTP was developed during the R-Type student project.

Status of This Memo

This document is the official specification of the R-Type Protocol,

and defines an Experimental Protocol for the R-Type community.

Discussion and suggestions for improvement are requested.

Distribution of this memo is unlimited.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the

document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal

Provisions Relating to IETF Documents in effect on the date of

publication of this document (http://trustee.ietf.org/license-info).

Please review these documents carefully, as they describe your rights

and restrictions with respect to this document.

Table of Contents

1. Introduction...........................................................

1.1 Conventions Used in This Document...............................

2. Session Management ....................................................  
3. Message Exchange ......................................................

4. Network Protocol ......................................................

4.1 Types ..........................................................

4.2 Packet .........................................................

5. References ............................................................

4. Author's Address ......................................................

[1](https://tools.ietf.org/html/rfc1459#section-1). Introduction

The R-Type protocol has been developed on systems using the TCP   
 network protocol and UDP network protocol, although there is no   
 requirement that this remain the only sphere in which it operates.

IP is described in [RFC0791]. TCP is described in [RFC0793].

UDP is described in [RFC0768].

1.1. Conventions Used in This Document

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 [RFC2119].

2. Session Management

The RTP session management facilities onto the UDP and TCP service is  
 straightforward. For TCP, the RTP session first requires a creation of  
 a TCP connection between two peers, one that initiates the connection   
 request and one that responds to the connection request. The initiating   
 peer is called the "client", and the responding peer is called the   
 "server". An RTP server MUST listen for UDP or TCP connection requests   
 on a standard TCP or UDP port.

An RTP session is normally ended by the client issuing and RTP   
 Disconnected packet. A server receiving and RTP Disconnected Packet  
 MUST end the RTP session, close the TCP connection if the socket was in  
 TCP mode, and broadcast this disconnexion on all client in the party.  
 In TCP mode, the client MAY end the RTP session by closing the socket.

3. Message Exchange

In UDP, the user is connected in a party when the RTP server returns 0   
 in the Error packet.

The user MUST be connected using the Connect packet to take part in   
 the parties.  
  
 After this, both client and server can initiate a RTP binary command.  
 In some commands, the server SHOULD NOT return a response for avoid   
 flooding the clients. For furthing details, go to the 4. Section.

The RTP packet’s command MUST be struct-packed for better transfert.

4. Network Protocol

4.1 Types

**CommandType:** uint8\_t  
 Ping = 0  
 Error = 1  
 Disconnected = 2  
 Connect = 5  
 CreateParty = 6  
 Spawn = 10  
 Move = 11  
 Destroyed = 12  
 Fire = 13  
 Collision = 14

**ObjectType:** uint8\_t  
 Ship = 0,  
 Missile = 1,  
 PowerUPs = 2,  
 Obstacle = 3

**ShipType:** uint8\_t  
 Standard = 0

**MissileType:** uint8\_t  
 Standard = 0

**PowerUPsType:** uint8\_t  
 Standard = 0

**ObstacleType:** uint8\_t  
 Standard = 0

**GameStatus:** uint8\_t  
 Waiting = 0,  
 GameOver = 1,  
 Win = 2

**Level:** uint8\_t  
 One = 1,  
 Two = 2,  
 Three = 3

**CollisionType:** uint8\_t  
 None = 0,  
 Destruction = 1,  
 PowerUP = 2

4.2 Packet

**Ping:**  
 CommandType cmdType;  
 long time;

This command can be invoqued either by the client or by the server.

The "cmdType" MUST be CommandType::Ping.  
 The "time" MUST be the Unix Time.

**Error:**  
 CommandType cmdType;  
 uint8\_t code;

This command can be invoqued only by the server.

The "cmdType" MUST be CommandType::Error.  
 The "code" MUST be:  
  
 0: OK  
 1: NOT FOUND  
 2: BAD PASSWORD  
 3: ALREADY EXIST

**Disconnected:**  
 CommandType cmdType;  
 uint16\_t id;

This command can be invoqued only by the client.

The "cmdType" MUST be CommandType::Disconnected.  
 The "id" MUST be the player ID to disconnect.

**Connect:**  
 CommandType cmdType;  
 uint16\_t size;  
 char data[0];  
  
 This command can be invoqued only by the client.

The "cmdType" MUST be CommandType::Connect.  
 The "size" MUST be the size of the "data" field.  
 The "data" MUST be formated like that: "partyName;partyPassword"

**On Succes**: 0  
 **On Error**: 1 OR 2

**CreateParty:**  
 CommandType cmdType;  
 uint16\_t size;  
 char data[0];

This command can be invoqued only by the client.

The "cmdType" MUST be CommandType::CreateParty.  
 The "size" MUST be the size of the "data" field.  
 The "data" MUST be formated like that: "partyName;partyPassword"

**On Succes**: 0  
 **On Error**: 1 OR 3

**Spawn:**  
 CommandType cmdType;  
 ObjectType object;  
 uint16\_t id\_tospawn;  
 uint16\_t position;  
 uint8\_t type;   
 uint8\_t effect;  
 bool is\_player;

This command can be invoqued either by the client or by the server.

The "cmdType" MUST be CommandType::Spawn.  
 The "object" MUST be the ObjectType to Spawn.  
 The "id\_tospawn" MUST be the ID associate with the newly object.  
 The "position" MUST be the position of the newly object.  
 The "type" MUST indicate witch type is.  
 The "effect" MUST contain the effect to apply to this object.  
 The "is\_player" tell us if the spawned entity MUST be the player.

**Move:**  
 CommandType cmdType;  
 uint16\_t id\_tomove;  
 uint32\_t position;  
 uint16\_t velocity;  
 uint8\_t direction;

This command can be invoqued either by the client or by the server.

The "cmdType" MUST be CommandType::Move.  
 The "id\_tomove" MUST be the id of the entity to move.  
 The "position" MUST be the new position to apply.  
 The "velocity" MUST be the velocity of the direction.  
 The "direction" MUST be the direction of the entity.

**Fire:**  
 CommandType cmdType;  
 MissileType type;  
 uint16\_t id;  
 uint16\_t id\_launcher;  
 uint16\_t position;  
 uint8\_t velocity;  
 uint8\_t angle;  
 uint8\_t effect;  
 uint8\_t level;

This command can be invoqued either by the client or by the server.

The "cmdType" MUST be CommandType::Fire.  
 The "type" MUST be the type of the missile to be invoqued.  
 The "id\_launcher" MUST be the ID of the launcher.  
 The "position" MUST be the starting position of the missile.  
 The "velocity" MUST be the orientation of the missile (1 or -1)  
 The "angle" MUST be the angle of the missible in degrees  
 The "effect" MUST be the effect to play to this missile.  
 The "level" MUST be the level of this missile.

**Collision:**  
 CommandType cmdType;  
 CollisionType type;  
 uint16\_t id\_first;  
 uint16\_t id\_second;

This command can be invoqued either by the client or by the server.

The "cmdType" MUST be CommandType::Collision.  
 The "type" MUST be the type of the collision.  
 The "id\_first" MUST be the ID of the first entity  
 The "id\_second" MUST be the ID of the second entity

**Game:**  
 CommandType cmdType;  
 GameStatus status ;

This command can be invoqued only by the server.

The "cmdType" MUST be CommandType::Game.  
 The "status" MUST be the status of the Party.

**Level:**  
 CommandType cmdType;  
 uint8\_t level;

This command can be invoqued only by the server.  
  
 The "cmdType" MUST be CommandType::Level.  
 The "level" MUST be the level number to play.

**Score:**  
 CommandType cmdType;  
 uint16\_t id;  
 uint16\_t score;

This command can be invoqued by the server and the client.  
  
 The "cmdType" MUST be CommandType::Level.  
 The "id" MUST be the id of the player.  
 The "score" MUST be the score of the player.

**Life:**  
 CommandType cmdType;  
 uint16\_t id;  
 uint16\_t life;

This command can be invoqued by the server and the client.  
  
 The "cmdType" MUST be CommandType::Level.  
 The "id" MUST be the id of the player.  
 The "life" MUST be the life of the player.

5. References

[RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791,

September 1981.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7,

RFC 793, September 1981

[RFC0768] Postel, J., "User Datagram Protocol", RFC 793,

28 August 1980

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate

Requirement Levels", BCP 14, RFC 2119, March 1997.

6. Author's Address

Jordan Aveline

R-Tacos, Inc.

5678 6 avenue de tartanpion

Jupiter

Voie lactée

EMail: jordan.aveline@epitech.eu