Salty Studio

V. Gouet
N. Constanty
F. Veyrier
Q. Gasparotto
A. Wery

November 2016

Expires: May 2019

R-Type SaltyStudio

Status of this Memo

This rtype RFC is submitted in full conformance with the provisions of BCP 78 and BCP 79. This document may not be modified, and derivative works of it may not be created, except to publish it as an RFC and to translate it into languages other than English.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

rtype RFC are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as rtype RFC.

rtype RFC are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use rtype RFC as reference material or to cite them other than as "work in progress."

The list of current rtype RFC can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

This rtype RFC will expire on Fail May 19, 2019.

Abstract

The R-Type protocol was developed over the last week since it was first implemented as a means for users on a r-type. Now it supports a world-wide network of network of servers and clients.

The R-Type protocol is a binary protocol, with the simplest client being any socket program capable of connecting to the server.

Table of Contents

1. Introduction

The R-Type protocol has been designed over one week for use with based conferencing. This document describes the current R-Type protocol.

The R-Type protocol has been developed on systems that using the TCP/IP network protocol between the client room and the server room. But it has also been developed on systems that using the UDP network protocol between the client game and the server game.

R-Type itself is a video game, which (through the use of the client-server model) is well-suited to run on many machines in a distributed fashion. A typical setup involves four processes (two clients and two servers). The client room will be connected to the server room with TCP protocol and the client game will be connected to the server game with UDP protocol.

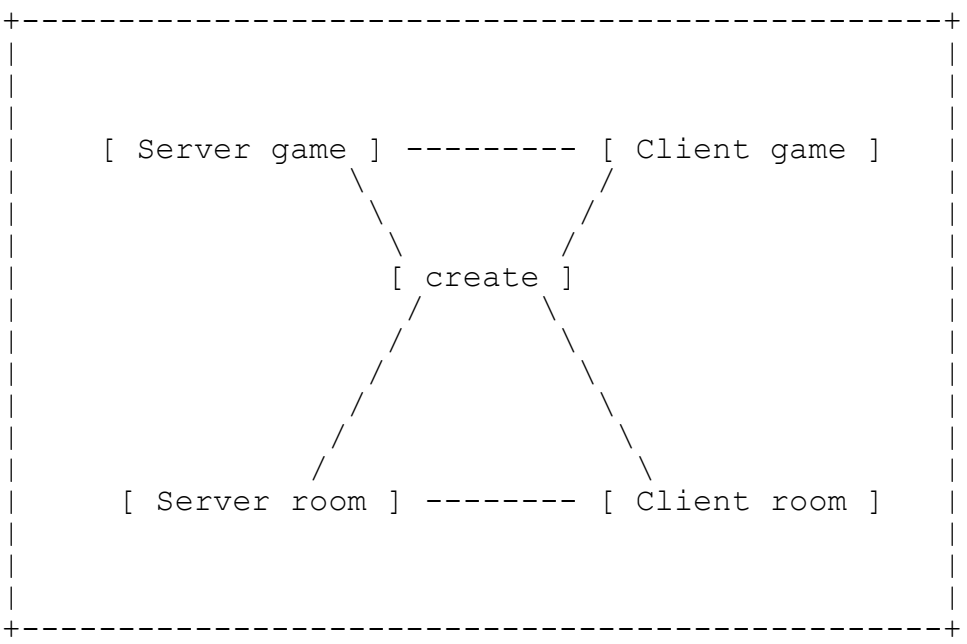


Figure 1 Format of the R-Type architecture

1.1. Servers

There are two kind of server:

- The server room is the main server of R-Type. This server shows every rooms, create server game and makes redirection.
  A room is an pre instance of a game, it contains the map of the game and players which are waiting.
  The server room uses TCP protocol.

- The server game is the second server of R-Type. This server manages the game (monster, AI, map, players). It is created by the server room with a port and a secret string.
  The secret is using to identified the client's connection. The client MUST give the secret for the connection.
  When the game is over the server stop and notify the server's room.
  The server game uses UDP protocol.

1.2. Clients

A client is anything connecting to a server that is not another server. Each client is distinguished from other clients by a unique pseudo having a maximum length of height (8) characters.

There are two kind of client:

- The client room is the main client of R-Type.
  It will display what the server room is sending to him:
  - Authentication screen (the user can choose his pseudo)
  - Room screen (display every rooms that the user MAY join)
  - Room selected (display the room selected and display which player are inside)
  - Launch screen (waiting the swap to the server game and client game)

  The client room uses TCP protocol.

- The client game is the second client of R-Type.
  It will display what the server game is sending to him:
  - Monster
  - Players
  - Map
  - Objects

  At the end of the game, the client game return to the client room and will be deleted.

  The client game uses UDP protocol.

2. Conventions used in this document

In examples, "C:" and "S:" indicate lines sent by the client and server respectively.

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 [RFC2119].

In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying significance described in RFC 2119.

In this document, the characters ">>" preceding an indented line(s) indicates a statement using the key words listed above. This convention aids reviewers in quickly identifying or finding the portions of this RFC covered by these keywords.

3. COMMON-HEADER Format

The R-Type project handle two protocols. The first protocol is used between client game and server game, the second is used between client room and server room. They are not the same.

Each packet MUST have a header. The header contains information about the status of the packet, it MUST be verified every time.

The following is the format of two protocols header.

3.1. Header Game protocol

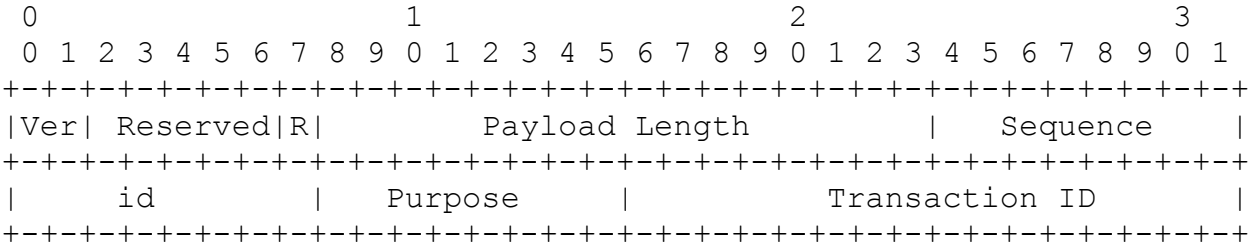


Figure 2 HEADER Game format

Ver: The 2-bit version field MUST be set to 1 to indicate this version of R-Type.

Reserved: At this point, the 5 bits in the reserved field.
The 1, 3 and 5 bits MUST be set to 1 by the sender of the message and MUST be verified by the receiver. The others bits MUST be set to 0.
R: This 1-bit field identify the main purpose of the message.
If the field is set to 1 then the packet is reliable.
That means it’s an important packet and you MUST verify if the packet is correctly arrived. This is commonly send by the server.
If you received a reliable packet, you MUST send a confirmation to the sender. You MAY ignore every identic packet after received this last.
If the field is set to 0 then the packet is unreliable.
That means you MUST NOT send a response that confirm you got it.

Payload Length: This 16-bit field contains the length of the message, excluding the game header.

Sequence id: Is a number that increases with each packet sent (and wraps around after 65535).

Purpose: That contains the type of the packet, you SHOULD identify the kind of the packet (movement, attack, defence, object apparition etc…). It will be explained later.

Transaction ID: This field contains a 16-bit value that allows users to match a given message with its response. The value of the Transaction ID in server-initiated transactions is 0.

3.2. Header Room protocol

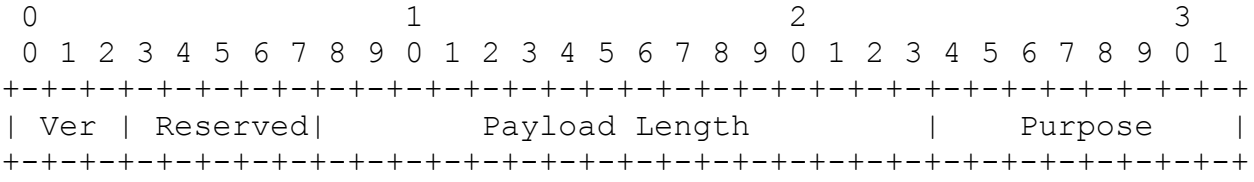


Figure 3 HEADER Room format

Ver: The 3-bit version field MUST be set to 1 to indicate this version of R-Type.

Reserved: At this point, the 5 bits in the reserved field. The 1, 3 and 5 bits MUST be set to 1 by the sender of the message and MUST be verified by the receiver. The others bits MUST be set to 0.

Payload Length: This 16-bit field contains the length of the message, excluding the room header.

Purpose: That contains the type of the packet, you SHOULD identify the kind of the packet (create/join/leave rooms, connection etc…). It will be explained later.

4. Commands Format

4.1 Commands Game protocol

4.1.1. MOVE

This packet MUST be unreliable.

Purpose: The purpose MUST be set to 1, it defines the movement packet.

This packet moves the Object ID to x/y axis.

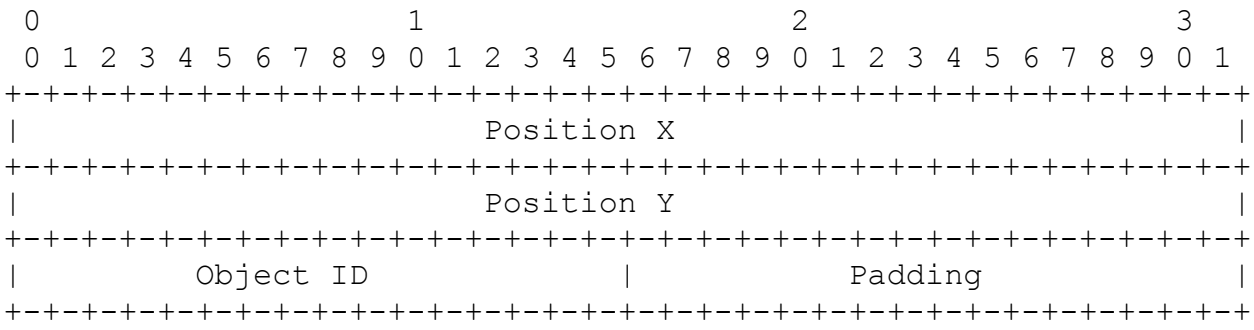


Figure 4 Move packet

Position X: Determine the position in X axis of the Object ID.

Position Y: Determine the position in Y axis of the Object ID.

Object ID: Determine the ID of the current object that is moving. It’s defined at the create of the object. (and wraps around after 65535).

4.1.2. SHOT

This packet MUST be reliable.

In case Server game receives data from Client Game: (client -> server)
The server MUST send to all client games a CREATE packet that include the type of the shot.

Purpose: The purpose is set to 2, it defines the “shot packet”.

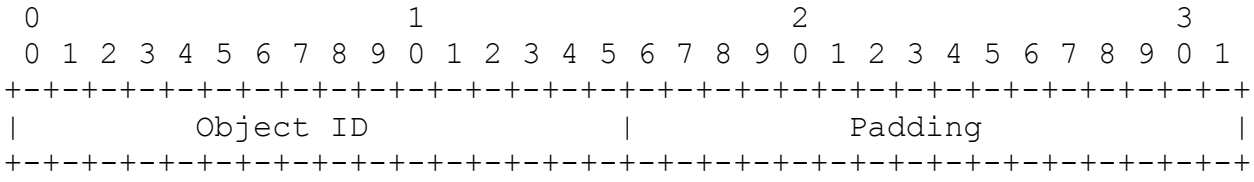


Figure 5 Shot packet

Object ID: Determine the ID of the current object that is shooting. It’s defined at the create of the object. (and wraps around after 65535).

4.1.3. TAKE

This packet MUST be reliable.

Purpose: The purpose MUST be set to 3, it defines the “take packet”.

The user MAY take item object and uses it.

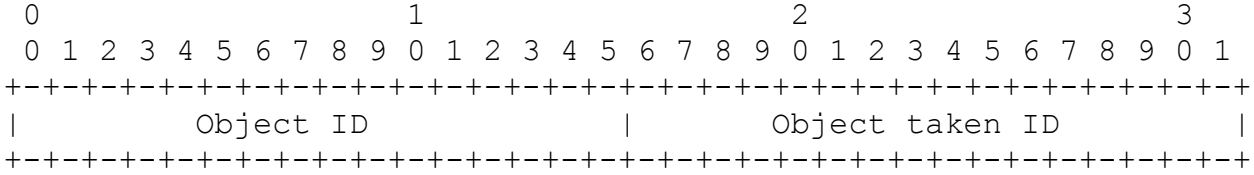


Figure 6 Take packet

Object ID: Determine the ID of the current object that is taking. It’s defined at the create of the object. (and wraps around after 65535).

Object taken ID: Determine the Object ID that is taken.

4.1.4. BEAM

This packet MUST be reliable.

Purpose: The purpose MUST be set to 4, it defines the “beam packet”.

This packet initialise the BEAM shot, you SHOULD send Beam packet before the Shot packet. The server confirms and calculates the delay between the packet beam and packet shot.

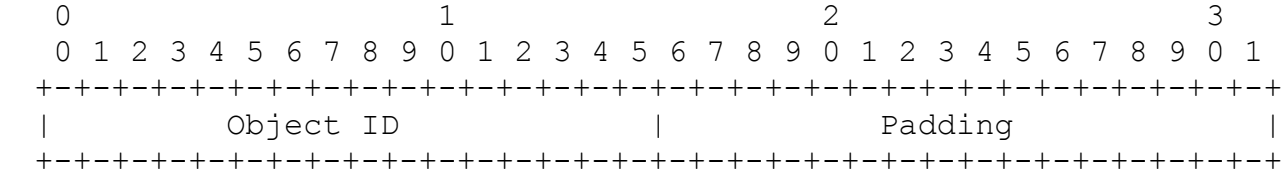


Figure 7 Beam packet

Object ID: Determine the ID of the current object that is shooting. It’s defined at the create of the object. (and wraps around after 65535).

4.1.5. DROP

This packet MUST be reliable.

Purpose: The purpose MUST be set to 5, it defines the “drop packet”.

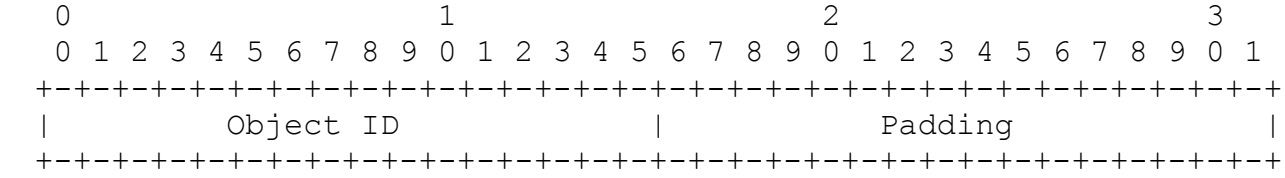


Figure 8 Drop packet

Object ID: Determine the ID of the current object that is shooting. It’s defined at the create of the object. (and wraps around after 65535).

4.1.6. DIE

This packet MUST be reliable.

Purpose: The purpose MUST be set to 6, it defines the “die packet”

The die packet deletes the object ID.

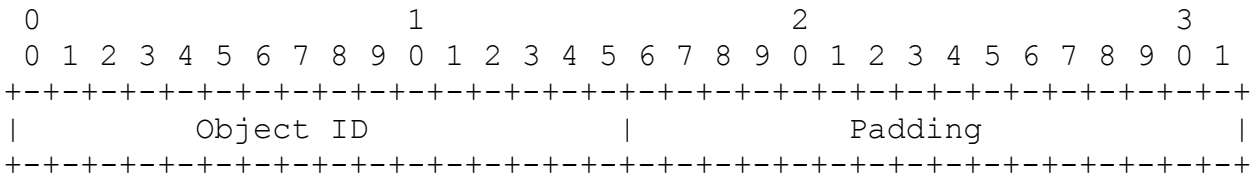


Figure 9 Die packet

Object ID: Determine the ID of the current object that is shooting. It’s defined at the create of the object. (and wraps around after 65535).

4.1.7 CREATE

This packet MUST be reliable.

Purpose: The purpose MUST be set to 7, it defines the “create packet”

The create packet creates a new instance of an object. Every object has an ID which IS NOT the Object ID. It creates an object at x and y position and set an Object ID.

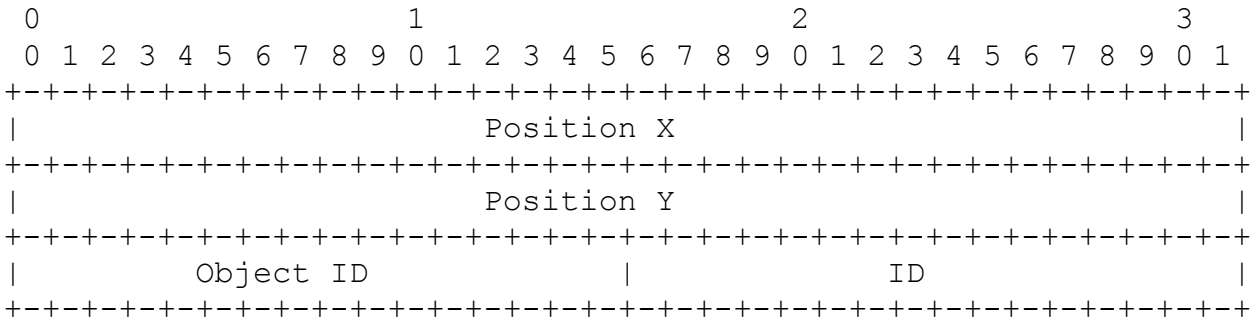


Figure 10 Create packet

Position X: Determine the position in X axis of the Object ID.

Position Y: Determine the position in Y axis of the Object ID.

Object ID: Determine the ID of the current object that is moving. It's defined at the create of the object. (and wraps around after 65535).

ID: The ID which can identified which Object has been created.

4.1.8. LAUNCH

This packet MUST be reliable.

Purpose: The purpose MUST be set to 8, it defines the "launch packet"

This package is used when the client has an item. He can launch it.
Create a move packet to the player's focus.
If the user doesn't have an item. Then you MUST NOT send launch packet.

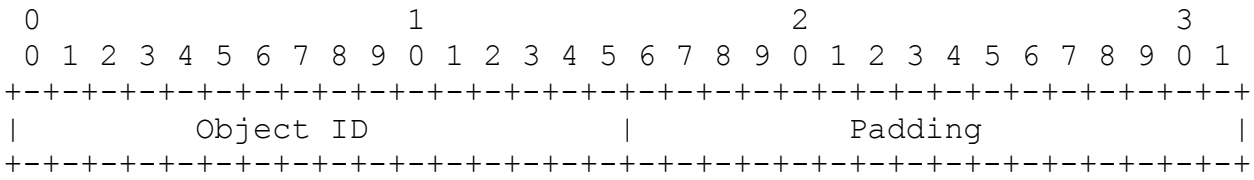


Figure 11 Launch packet

Object ID: Determine the ID of the current object that is moving. It's defined at the create of the object. (and wraps around after 65535).

4.1.9. STATUS

This packet MAY be unreliable.

Purpose: The purpose MUST be set to 9, it defines the “status packet”

The status informs the status of the game:
- High score of a user id
- Game’s status

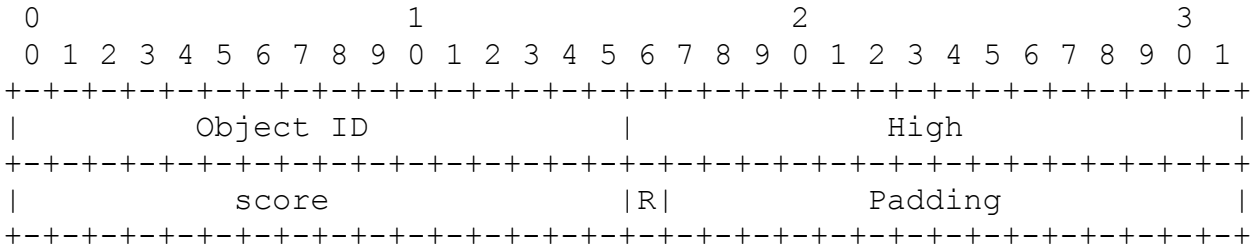


Figure 12 Status packet

Object ID: Determine the ID of the current object that is moving. It’s defined at the create of the object. (and wraps around after 65535).

High score: It’s the high score of the Object ID. (coded unsigned int).

R: Determine if the game is running or not, if set to 0 then the game has stopped, it’s the end of the game. The packet MUST be reliable. If set to 1 then the game is running. The packet MUST be unreliable.

4.1.10. AUTHENTICATE

This packet MUST be reliable.

Purpose: The purpose MUST be set to 10, it defines the “authenticate packet”

This packet MUST be sent to the server to inform him that the client is allowed to join this server. If the secret is false, then the server game deletes the client.

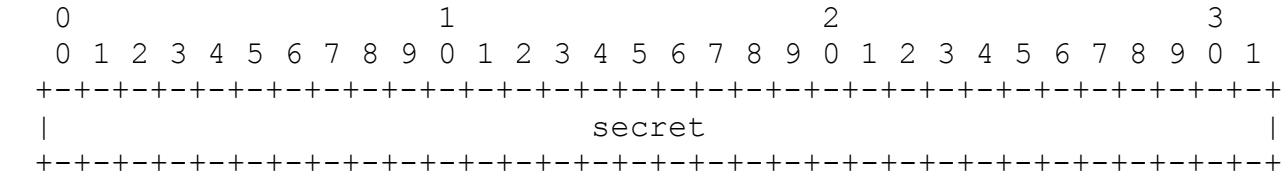


Figure 13 AUTHENTICATE Packet

Secret: is the secret code that the user MUST use for the game server authenticate. The server room gives the secret.

4.2. Commands Room protocol

4.2.1. CREATE

CREATE packet performs the operation to create a room.

If the operation succeeds:
- The client receives CREATE packet.
- The client sends JOIN packet
- The server sends GET packet.

If the operation fails:
- The server sends FAILURE packet.

Purpose: The purpose MUST be set to 1, it defines the “create packet”

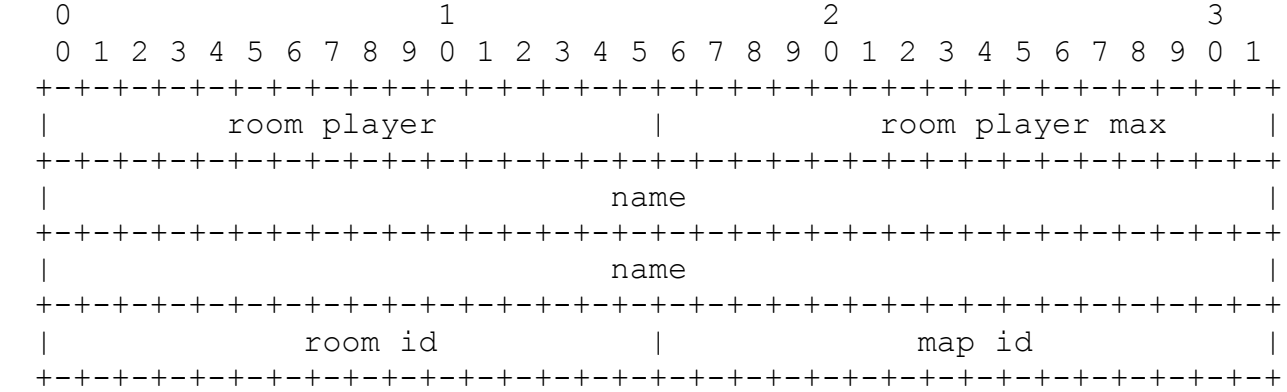


Figure 14 CREATE Packet

Room player: The number of player inside the room.

Room player max: The number max of player who MAY join the room.

Name: The name of the room. (8 bytes).

Room id: The id of the room. (2 bytes).

Map id: The id of the map (2bytes).
The map id is set by the rtype_"file".

4.2.2. JOIN

JOIN packet performs the operation to join a room.

If the operation succeeds:
- The server sends GET packet.
- The server sends PLUGED packet.
- The client receives the JOIN packet

If the operation fails:
- The server sends FAILURE packet.

Purpose: The purpose MUST be set to 2, it defines the "join packet"

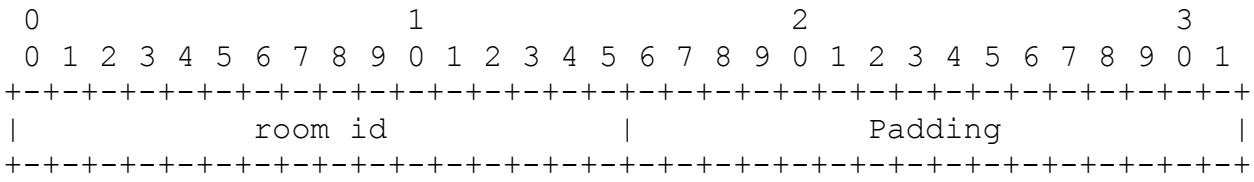


Figure 15 JOIN packet

room id: The id of the room that the user wants to join.

4.2.3. QUIT

QUIT packet performs the operation to leave a room.

If the operation succeeds:
- The client receives QUIT packet.

- The server sends GET packet.

If the operation fails:

- The server sends FAILURE packet.

Purpose: The purpose MUST be set to 3, it defines the “quit packet”

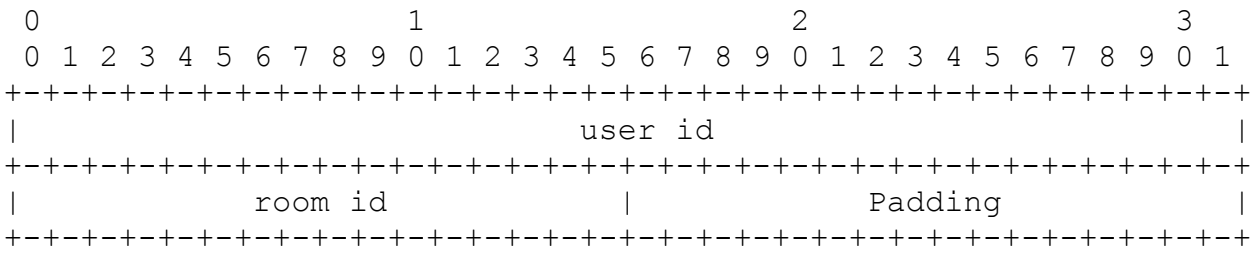


Figure 16 QUIT Packet

room id: The id of the room that the user wants to quit.

User id: the id of the user. It MUST be set to 0 if the client sends this packet. The attribute MUST be set when the server sends it.

4.2.4. AUTHENTICATE

AUTHENTICATE packet is the first operation that the user MUST do.

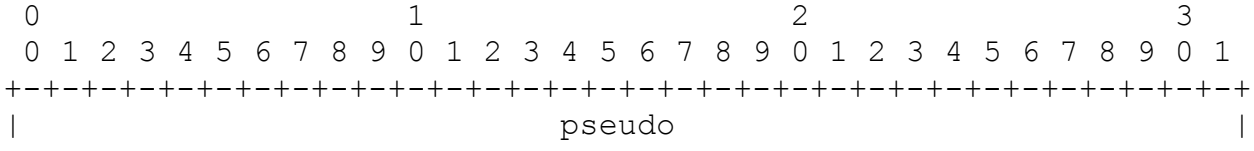
If the operation succeeds:

- The client sends AUTENTICATE packet
- The server sends GET packet (for each rooms).

If the operation fails:

- The server sends FAILURE packet.

Purpose: The purpose MUST be set to 4, it defines the “authenticate packet”



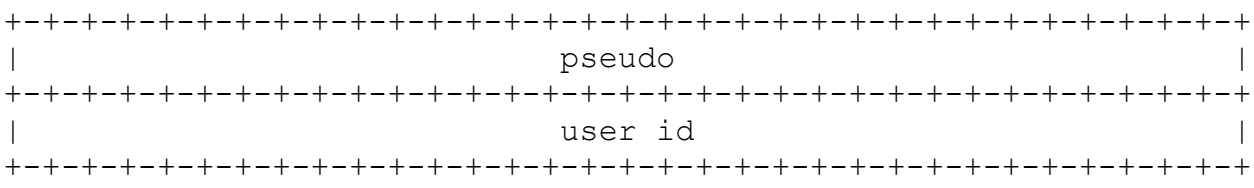


Figure 17 AUTHENTICATE Packet

pseudo: is the name of the user (8 bytes).

User id: the id of the user. It MUST be set to 0 if the client sends this packet. The attribute MUST be set when the server sends it.

4.2.5. PLUGED

PLUGED packet informs who has join your room.

Purpose: The purpose MUST be set to 5, it defines the “pluged packet”

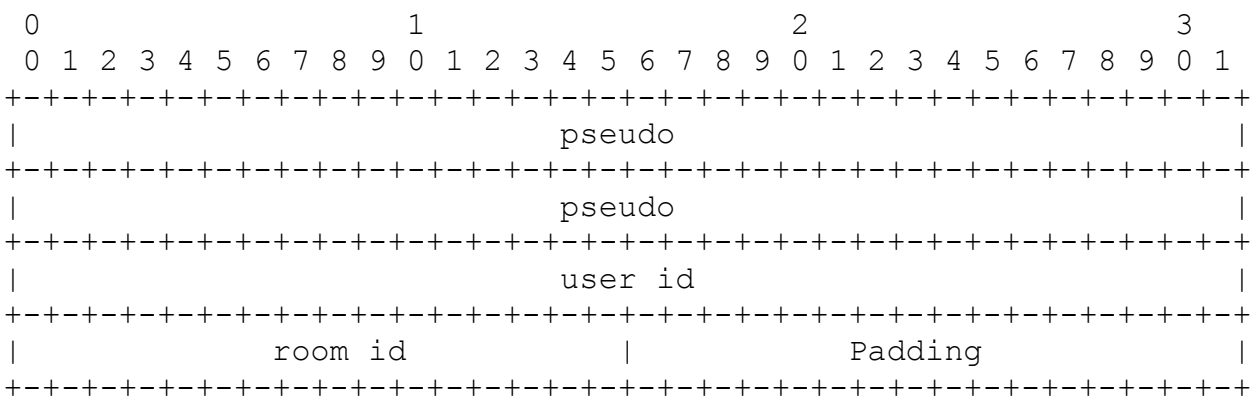


Figure 18 PLUGED Packet

pseudo: is the name of the user (8 bytes).

User id: the id of the user. It MUST be set to 0 if the client sends this packet. The attribute MUST be set when the server sends it.

room id: The id of the room that the user has join.

4.2.6. SWAP

SWAP packet MUST be send when the user MAY swap to the server game.

Purpose: The purpose MUST be set to 6, it defines the “swap packet”

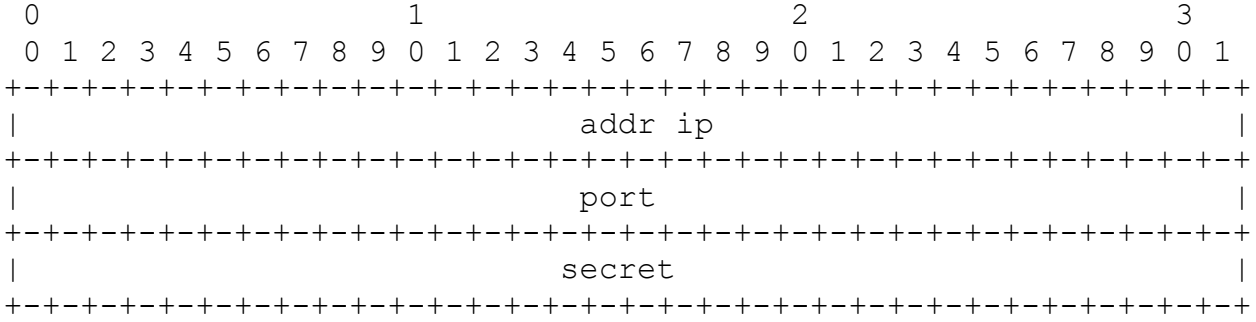


Figure 19 SWAP Packet

addr ip: is the game server’s IP that the user MAY connect.

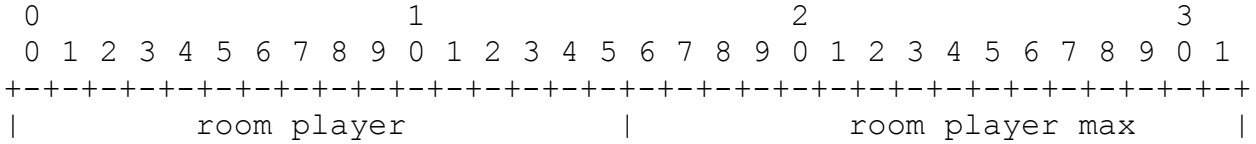
Port: is the game server’s port that the user MAY connect.

Secret: is the secret code that the user MUST use for the game server authenticate.

4.2.7. GET

GET packet SHOULD be sent when a CREATE packet is used. It gives information about a room.

Purpose: The purpose MUST be set to 7, it defines the “get packet”



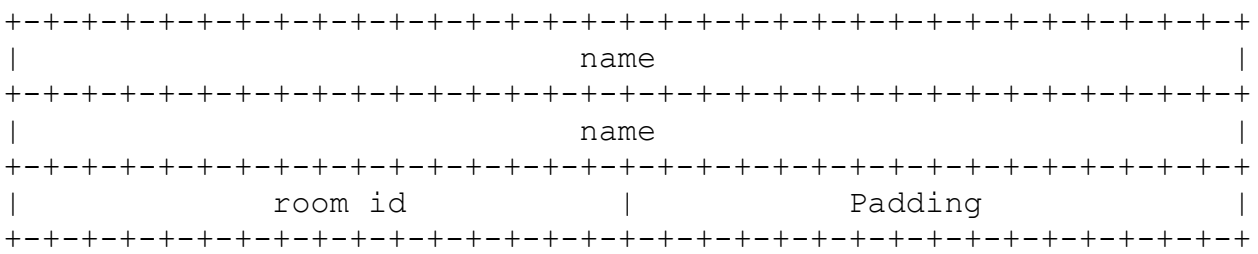


Figure 20 GET packet

Room player: The number of player inside the room.

Room player max: The number max of player who MAY join the room.

Name: The name of the room. (8 bytes).

Room id: The id of the room. (2 bytes).

4.2.8. FAILURE

FAILURE packet is used to inform the client that an operation has failed.

Purpose: The purpose MUST be set to 8, it defines the “failure packet”

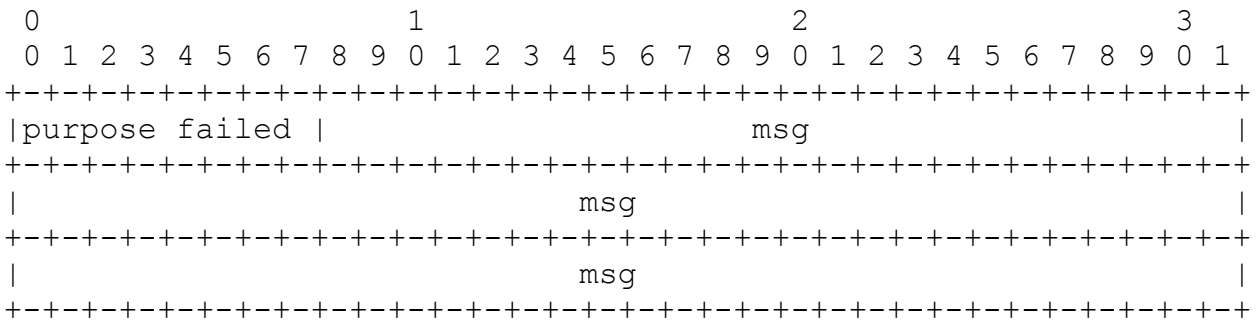


Figure 21 FAILURE packet

Purpose failed: Is the id of the command which has failed.

Msg: Is the message of the failure.

5. Security Considerations

For both server connections you have to send the AUTHENTICATE packet and follow the instruction about the packet as describe in section 4.1.10 and 4.2.4.

6. References

6.1 Normative References

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[2] Crocker, D. and Overell, P.(Editors), "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, Internet Mail Consortium and Demon Internet Ltd., November 1997.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2234] Crocker, D. and Overell, P.(Editors), "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, Internet Mail Consortium and Demon Internet Ltd., November 1997.

6.2 Informative References

[3] Faber, T., Touch, J. and W. Yue, "The TIME-WAIT state in TCP and Its Effect on Busy Servers", Proc. Infocom 1999 pp. 1573-1583.

[Fab1999] Faber, T., Touch, J. and W. Yue, "The TIME-WAIT state in TCP and Its Effect on Busy Servers", Proc. Infocom 1999 pp. 1573-1583.

7. Acknowledgments

This document was prepared using 2-Word-v2.0.template.dot.

Appendix A. Copyright

Authors' Addresses

Victor Gouet
Projet Manager
BORDEAUX

Phone: 06 06 73 10 67
Email: victor.gouet@epitech.eu

Nicolas Constanty
Programmer client
BORDEAUX

Phone: 06 10 41 96 59
Email: nicoas.constanty@epitech.eu

Fernand Veyrier
Programmer client
BORDEAUX

Phone: 06 86 42 29 15
Email: fernand.veyrier@epitech.eu

Quentin Gasparotto
Programmer server
BORDEAUX

Phone: 06 51 59 33 19
Email: quentin.gasparotto@epitech.eu

Adrien Wery
Programmer server
BORDEAUX

Phone: 06 31 11 45 23
Email: adrien.wery@epitech.eu