L. Carr
G. Buret
A. Lechat
Q. Jodelet
Telecom Bretagne
11 2016

Group 5: c2w protocol specification proposal
draft-ietf-xml2rfc-template-05

Abstract

Protocol specification for an online chatting system. The system is made up of chat rooms, each of them corresponds to an online video stream that users may discuss.

Table of Contents

1. Introduction 2
1.1. Requirements Language 2
1.2. Definitions 2
2. Protocol principle 3
3. Protocol reliability 4
4. Server configuration 4
5. Client configuration 5
6. Packet format 5
7. Message type 7
7.1. Login 7
7.1.1. Message type: PUT_LOGIN 7
7.1.2. Message type: RESPONSE_LOGIN 8
7.2. Logout 8
7.2.1. Message type: PUT_LOGOUT 8
7.2.2. Message type: RESPONSE_LOGOUT 8
7.3. Ping 9
7.3.1. Message type: GET_PING 9
7.3.2. Message type: RESPONSE_PING 10
7.4. Events 10
7.4.1. Message type: GET_EVENTS 10
7.4.2. Message type: RESPONSE_EVENTS 11
7.5. Room 11
7.5.1. Message type: GET_ROOMS 12
7.5.2. Message type: RESPONSE_ROOMS 12
7.6. Users 13
7.6.1. Message type: GET_USERS 13
7.6.2. Message type: RESPONSE_USERS 14
7.7. Switch room 15
7.7.1. Message type: PUT_SWITCH_ROOM 15

7.7.2. Message type: RESPONSE_SWITCH_ROOM 16
7.8. New message 16
7.8.1. Message type: PUT_NEW_MESSAGE 16
7.8.2. Message type: RESPONSE_NEW_MESSAGE 17
8. Event Type 17
8.1. Event Type: MESSAGE 18
8.2. Event Type: NEW_USER 19
8.3. Event Type: SWITCH_ROOM 19
8.4. Event Type: LOGOUT 20
9. Examples 21
9.1. Scenario 1: logging in 21
9.2. Scenario 2: Retrieve new messages 22
9.3. Misbehavior examples 23
9.4. Packet examples 24
10. Normative References 27
Authors' Addresses 27

1. Introduction

The goal of the protocol is to allow online chatting while watching videos. In an implementation of the protocol, a user MAY join a video chat room to post messages in a "chat room" to the users viewing the same video.

Once logged in, the user client joins the "main room" and MAY see the list of available video streams, the list of available users and whether they joined a video chat room or stayed in the main room.

In any room (i.e. the "main room" or a room linked to a certain video stream), the user client MAY post messages, receive messages sent to the video chat room and leave the video chat room. The messages are always sent to all users in the video chat room (broadcast). If the user client leaves the video chat room, he/she will join the main room.

This protocol SHOULD work on either TCP or UDP.

1.1. Requirements Language

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].

1.2. Definitions

Client: Software on the user end communicating with the server

User: The entity using the c2w software.

Video chat room: a chat room linked to a video stream.

Main room: the room not linked to any video stream that the user client joins right after logging in.

2. Protocol principle

The protocol is a simple pull protocol. For each action, the client MUST interrogate the server, and the server MUST respond with the correct response. The server CANNOT initiate an exchange with the client.

The packets that a client can send are of the type GET or PUT. GET messages are for requesting information known by the server (like requesting the rooms list). PUT messages are for sending information to the server (like switching room or sending a new message).

In each case the server MUST respond with a packet of type RESPONSE. Each response from a PUT request has a STATUS_CODE field explaining if the modification had succeed or failed, and why it has failed.

It is important that a client is able to access specifically to any type of data. So each type of message (events, room, users, events status) has its own packet type. A response MUST NOT contain several type of data.

To be considered connected with the server, the client MUST send a GET_PING message regularly. If the server hasn't received any GET_PING message within 60 seconds the server MUST consider the connection lost. If the client hasn't receive any response from the server within 60 seconds (after mutiple retries, which is described in reliability section), the client MUST consider the connection as lost.

During a PING roundtrip (GET_PING + RESPONSE_PING) the server will send the ID of the last event that occured in the user's room with the LAST_EVENT_ID field. An event is defined with its ID which is a unique 3 bytes number generated in ascending order by the server. An event can be anything like the login of any user, the loggout, the switching to a new room or a new message sent. Each event is described in the "Event Type" section.

After a PING roundtrip, the client will compare its last known LAST_EVENT_ID with the one sent by the server. Three cases arise:

- The server LAST_EVENT_ID is equal to the client's one: the client and the server are synchronized. There is nothing to do.

The server LAST_EVENT_ID is greater than the client's one: the client is out of sync and MUST initiate a GET_EVENTS request to be synchronized.

The server LAST_EVENT_ID is smaller than the client's one: the client is out of sync, the LAST_EVENT_ID reached its maximum value and the server has reset this number to 0. The client MUST initiate a GET_EVENTS request to be synchronized.

During an EVENTS roundtrip (GET_EVENTS + RESPONSE_EVENTS) the user will retrieve every event specific to user's room that occured since the last EVENTS roundtrip. The method described below allows the user to get the events paginated, so it wron't reach the bytes limit of an IP packet.

The protocol maintainers decided to separate PING rountrips with EVENTS roundtrips to allow easy evolution in the protocol. In which case, the PING packets will always remain small (for the sole purpose of retrieving the server current state), and the GET_EVENTS and RESPONSE_EVENTS can become as complicated as necessary.

3. Protocol reliability

Each message contains a SEQ_NUMBER which is a 2 byte long number. This SEQ_NUMBER MUST be the same in the request and in the response. The client MUST use it to ensure that each request message received a response message. Two different request messages MUST have a different SEQ_NUMBER. Each client SHOULD use its own automatic looping and ascending number generator.

When a client does not receive a response message after a time out for the request message, if it sends again the same request message, it MUST use the same SEQ_NUMBER as the previous one. A server MUST NOT execute several times a PUT client request which has the same SEQ_NUMBER. If a server receive several times the same request with an unique SEQ_NUMBER, it MUST return the same response message but not reexecute the PUT operation. If the SEQ_NUMBER received by the server is not the expected number, the packet is dropped.

A client does not need to acknowledge a server response.

4. Server configuration

To keep all clients synchronized, the server MUST keep a minimal configuration.

The server MUST keep track of every event that happened, it MUST ensure that if a client requests a specific EVENT_ID the server can

send it back. The default ID for the first event is 0. There are 2^24 events ID available at start. When the counter goes back to 0, the old events are overwritten by the new ones. The event IDs are reset when the server starts.

The server MUST keep track of every client ID and client username. The client IDs are attributed by the server, starting from 1 (0 is reserved). The server always gives the smallest ID available to a new user.

A user ID is specific to a user (MUST NOT be shared between users) and is available when its corresponding user logs out.

5. Client configuration

6. Packet format

All messages have the same format, as shown in the figure below (Figure 1).

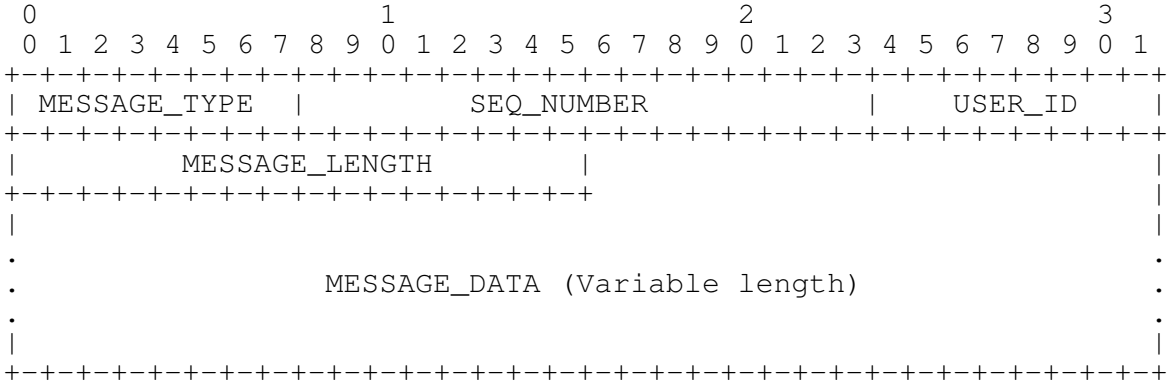
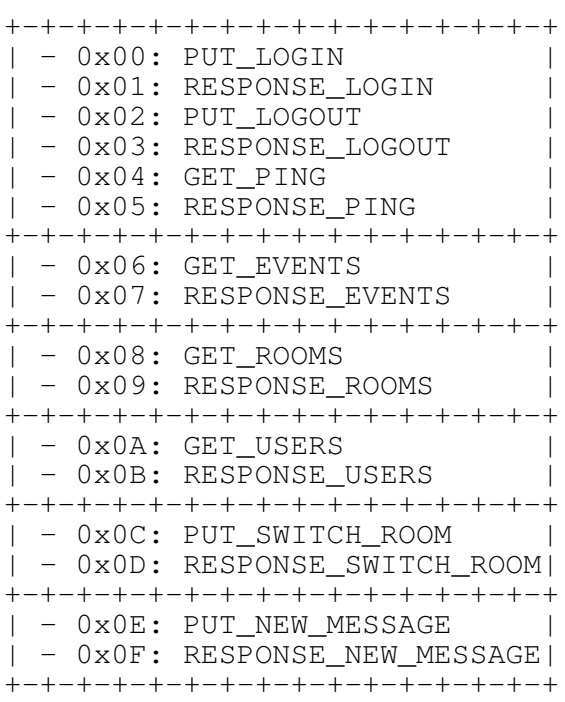


Figure 1

Message type (1 byte)

Describe the type of the message, which is one of the following:



These messages type codes are described below.

SEQ_NUMBER (2 bytes)

This field contains the identification number of the message. It is assigned by the sender.

This server MUST send it with the response message to acknowledge the previous message from the client.

USER_ID (1 byte)

This field contains the identification number of the user sending the message.

User ID is assigned to the user by the server with RESPONSE_LOGIN. The server MUST set this field to 0.

During login the client MUST set this value to 0.

This field enables the protocol to be network independant and port independant.

MESSAGE_LENGTH (2 bytes)

This field specifies the total length in bytes of the Message Data field.

The sender MUST ensure that this field contains the correct value.

MESSAGE_DATA (variable length)

This field contains the payload. It MAY be empty. Its size MUST be smaller then 65530 bytes, which is the maximum size of a TCP packet. Possible message for this field are described in Section 7. Each packet MUST NOT contain more than one message.

7. Message type

This section describes each possible message that SHOULD be put in the MESSAGE_DATA field of the packet. Each packet MUST NOT contain more than one message.

7.1. Login

7.1.1. Message type: PUT_LOGIN

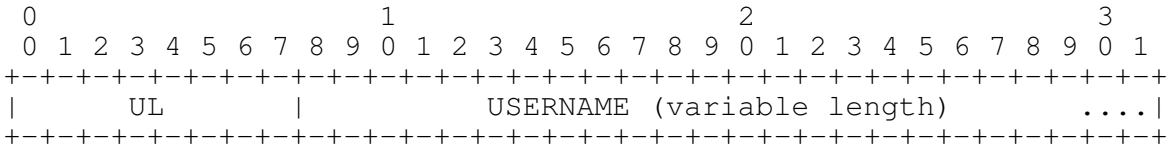


Figure 2

UL (Username Length): 1 byte

Define the length in bytes of the USERNAME field.

USERNAME: Variable length

The user can choose his/her username during login using this field.

It MUST be shorter than 255 bytes and longer than 0 bytes.

When a user logs in he joins the main room (by default), which has ID 0x0.

7.1.2. Message type: RESPONSE_LOGIN

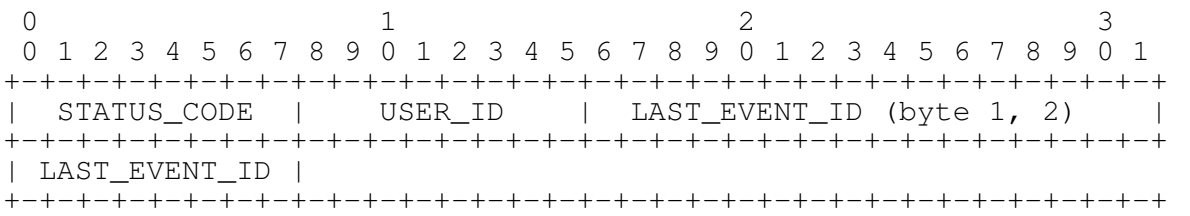


Figure 3

STATUS_CODE: 1 byte Returned by the server depending on the success of the login operation

0x0: SUCCESS

0x1: UNKWON_ERROR

0x2: TOO_MANY_USERS

0x3: INVALID_USERNAME (if the server has its own rules)

0x4: USERNAME_NOT_AVAILABLE

USER_ID: 1 byte

During login, the server generates a unique ID which represents the user. The client must save this ID for further operations.

LAST_EVENT_ID: 3 bytes

ID of the last event which occured in the main room before the user logged in.

7.2. Logout

7.2.1. Message type: PUT_LOGOUT

MESSAGE_DATA field MUST be empty.

7.2.2. Message type: RESPONSE_LOGOUT

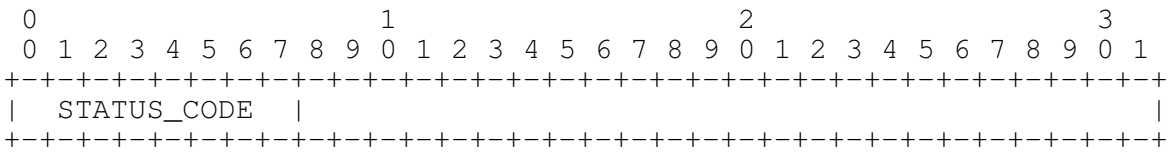


Figure 4

STATUS_CODE: 1 byte Returned by the server depending on the success of the logout operation

0x0: SUCCESS

0x1: UNKWON_ERROR

7.3. Ping

7.3.1. Message type: GET_PING

Each client MUST at least send a GET_PING message every 1 minute, if not the server MUST considers the client offline.

Each client MUST at most send one GET_PING message every 500 ms, otherwise the server SHOULD not send a response. This restriction is related to performance issues. A good practice for the client SHOULD be to increase exponentially the delay between two GET_PING when there is no change. This optimisation MAY NOT be implemented and a constant value MAY be used.

A default, constant value for PING frequency could be 1 second.

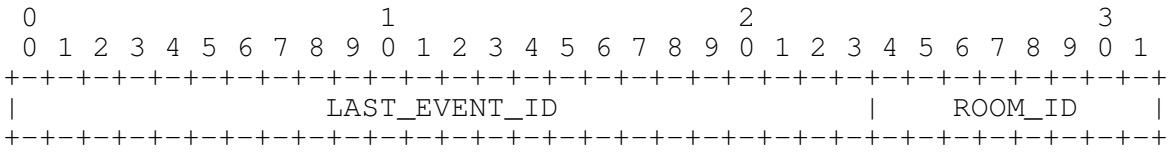


Figure 5

LAST_EVENT_ID: 3 bytes

ID of the last event known by the client.

ROOM_ID: 1 byte

ID of the room in which the user is located.

7.3.2. Message type: RESPONSE_PING

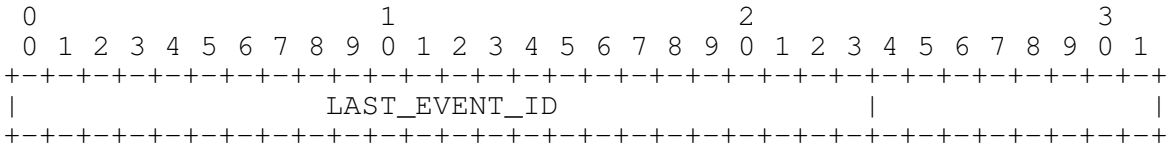


Figure 6

LAST_EVENT_ID: 3 bytes

ID of the last event which occured in the room specified in the GET_PING request.

7.4. Events

7.4.1. Message type: GET_EVENTS

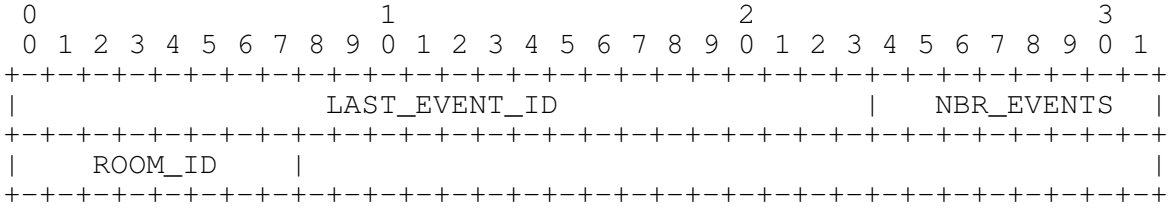


Figure 7

LAST_EVENT_ID: 3 bytes

ID of the last event known by the client.

NBR_EVENTS: 1 byte

Number of new events that the client retrieves from the server since last event known by the client.

MUST by a number greater than 0 an lesser than 255.

ROOM_ID: 1 byte

ID of the room in which the events that we want to retrieve occured. If ROOM_ID is set to 0, retrieved events are not room dependant (i.e. may come from every room).

7.4.2. Message type: RESPONSE_EVENTS

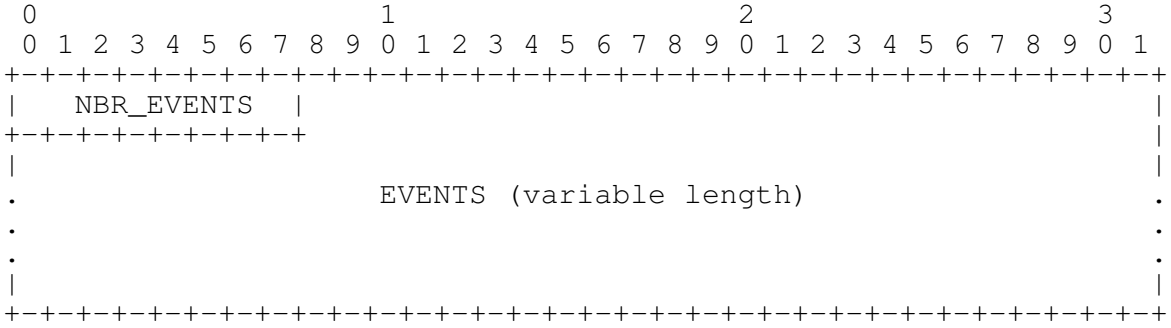


Figure 8

NBR_EVENTS: 1 byte

Number of events sent by the server. The server MUST NOT send more than 255 events.

EVENTS: variable length

List of the events specific to the room specified in the GET_EVENTS request. Each event follows the format as described below. There are 4 different types of events (MESSAGE, NEW_USER, SWITCH_ROOM, LOGOUT). Those events are described in section 8 below. In summary, those events are used to organize and notify the client about everything that happens on the server. Thus the client can know about the traffic of user and retrieve the messages that are sent.

7.5. Room

The protocol retrieves information for a list of room IDs specified by the user. As defined below, the list countains NBR_ROOMS. The list of room IDs is populated by ascending order, its first element being FIRST_ROOM_ID.

7.5.1. Message type: GET_ROOMS

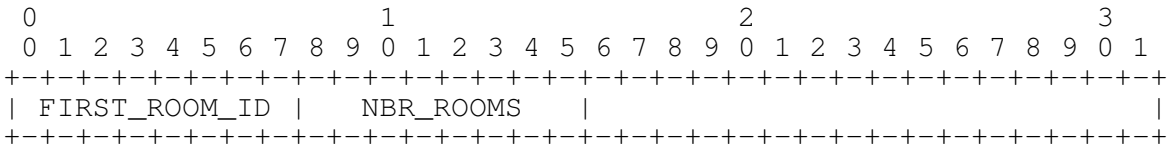


Figure 9

FIRST_ROOM_ID: 1 byte

ID of the first room that the user seeks information about.

NBR_ROOMS: 1 byte

Number of rooms that the user seeks information about.

It is mandatory to have sent a GET_USERS request before sending a GET_ROOMS request.

7.5.2. Message type: RESPONSE_ROOMS

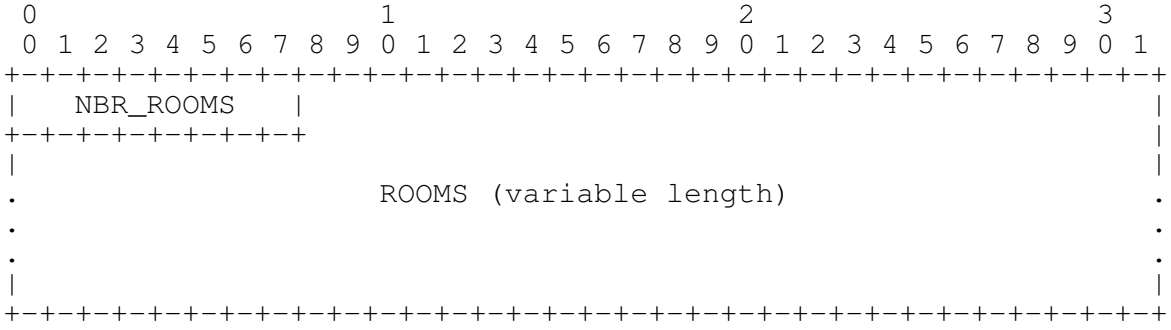


Figure 10

NBR_ROOMS: 1 byte

Number of rooms sent by the server. The server MUST NOT send information about more than 255 rooms.

ROOMS: variable length

List of rooms. Each room follows the format as described below.

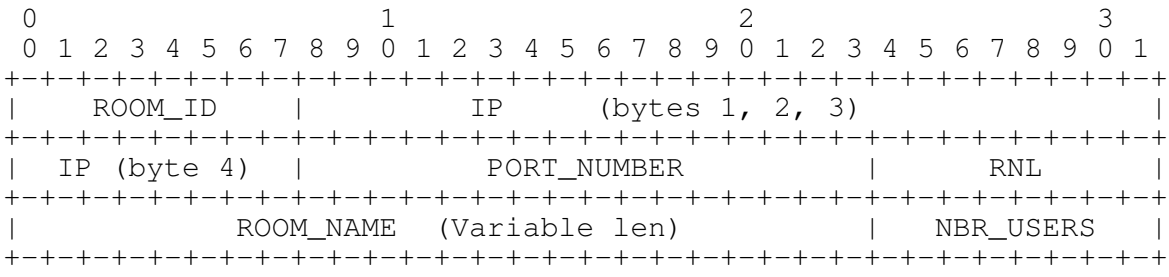


Figure 11

ROOM_ID: 1 byte

The ID of the room.

IP: 4 bytes

The IP address of the video server.

PORT_NUMBER: 2 bytes

The port number of the video server.

RNL (Room Name Length): 1 byte

The length of the room name.

ROOM_NAME: variable length

The displayed name of the room.

NBR_USERS: 1 byte Current number of users inside the room. The maximum number of user per room is 255.

7.6. Users

7.6.1. Message type: GET_USERS

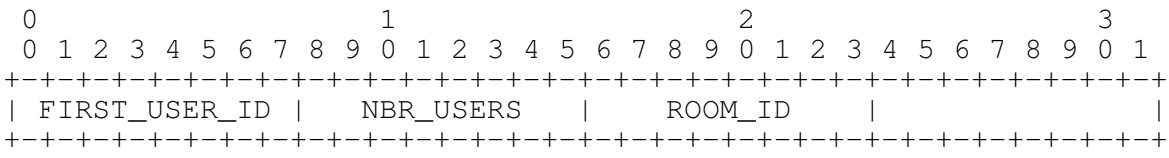


Figure 12

FIRST_USER_ID: 1 byte ID of the first user in the list of users to retrieve.

NBR_USERS: 1 byte Size of the list of users to retrieve.

ROOM_ID: 1 byte ID of the room where the users are. Set to 0 to get users all rooms.

7.6.2. Message type: RESPONSE_USERS

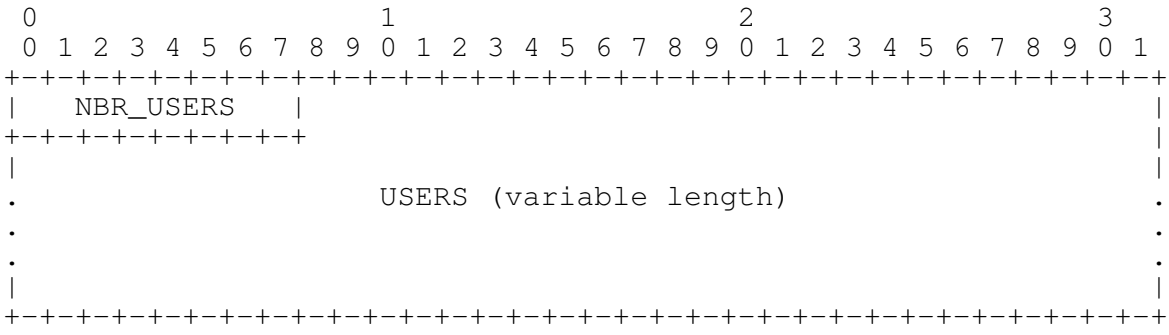


Figure 13

NBR_USERS: 1 byte

Number of users sent by the server. The server MUST NOT send information about more than 255 users

USERS: variable length

List of users. Each user follows the format as described below.

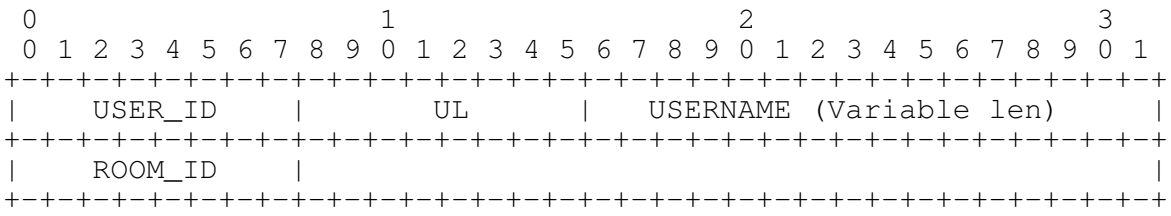


Figure 14

USER_ID: 1 byte

The unique ID of the user.

UL (Username Length): 1 byte

Length of the username.

USERNAME: variable length

Username of the user.

ROOM_ID: 1 byte

ID of the room in which the user is located.

If this ID is set to 0, it means that the user is in the main room (A). Else, the user is in a movie room (M).

7.7. Switch room

7.7.1. Message type: PUT_SWITCH_ROOM

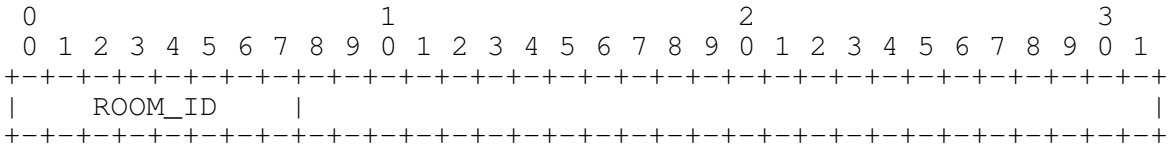


Figure 15

ROOM_ID: 1 byte

ID of the destination room. If this ID is set to 0, it means the user wants to switch to the main room (A). Else, the user will switch to a movie room (M).

It is mandatory to send a GET_USERS request after switching room, as events received by user were specific to user's previous room.

7.7.2. Message type: RESPONSE_SWITCH_ROOM

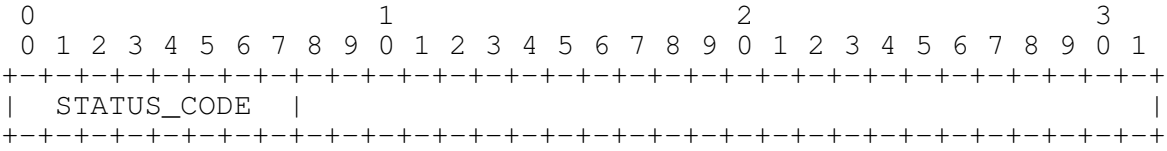


Figure 16

STATUS_CODE: 1 byte

Returned by the server depending on the success or not of the switching operation

0x0: SUCCESS

0x1: UNKWON_ERROR

7.8. New message

7.8.1. Message type: PUT_NEW_MESSAGE

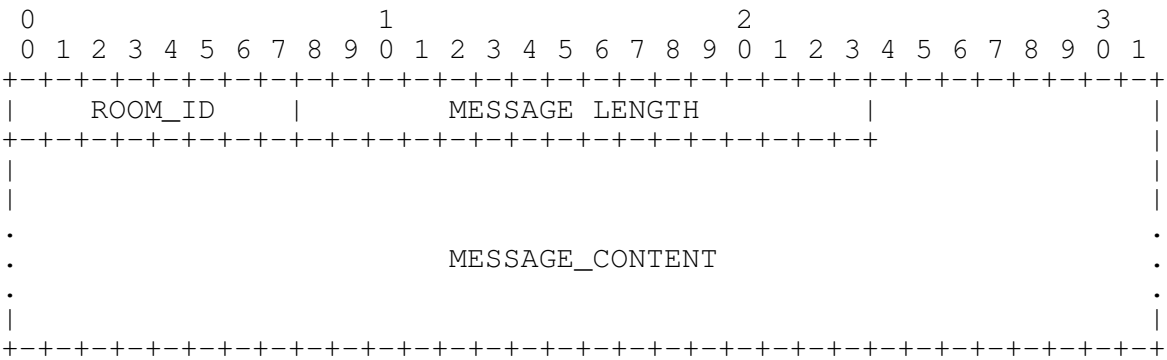


Figure 17

ROOM_ID: 1 byte

ID of the room if which the user sends the message. Server MAY refuse the message if this room is different from the current user room. (error INCORRECT_ROOM)

MESSAGE_LENGTH: 2 bytes

Length of the message.

MESSAGE_CONTENT: variable length

The message sent by the user.

7.8.2. Message type: RESPONSE_NEW_MESSAGE

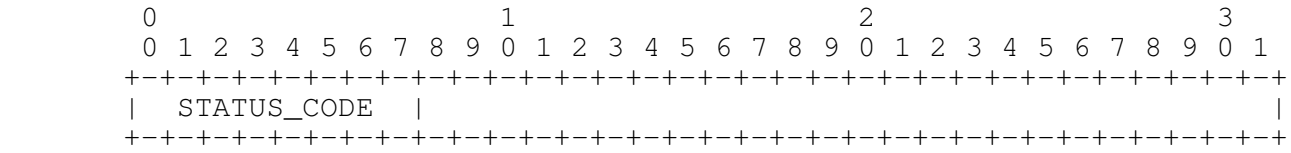


Figure 18

STATUS_CODE: 1 byte Returned by the server depending on the success or not of the message sending operation

0x0: SUCCESS

0x1: UNKWON_ERROR

0x2: INVALID_ROOM

0x3: INCORRECT_ROOM: Server MAY refuse the message if the room is different from the current user room.

8. Event Type

This section will describe the content of the field EVENTS in the message RESPONSE_EVENTS (section 7.4.2). EVENTS is actually a list of different events which are splitted in 4 types : MESSAGE (a message has been sent in a chatroom), NEW_USER (a new client connected to the server), SWITCH_ROOM (a client moved from one room to another) and LOGOUT (a client disconnected or timed out from the server).

8.1. Event Type: MESSAGE

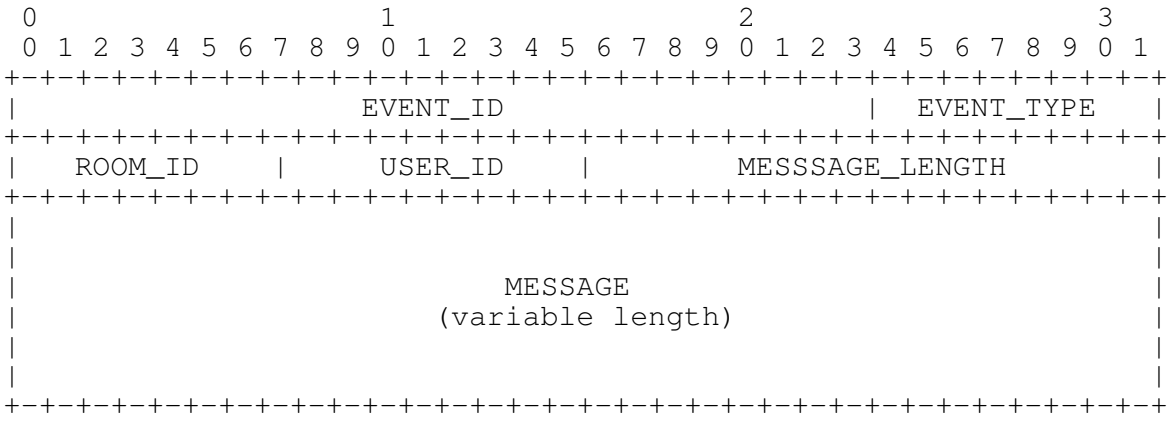


Figure 19

EVENT_TYPE: MESSAGE = 0x1

This event represents a message sent by a user.

EVENT_ID: 3 bytes:

Unique ID of the event generated by the server. MUST be generated in ascending order.

ROOM_ID: 1 byte

The ID of the room in which the message was sent.

USER_ID: 1 byte

The ID of the user who sent the message.

MESSAGE_LENGTH: 2 bytes

The length of the message content

MESSAGE: variable length

The content of the message. MUST be longer than 0 byte and shorter than 65 531 bytes.

8.2. Event Type: NEW_USER

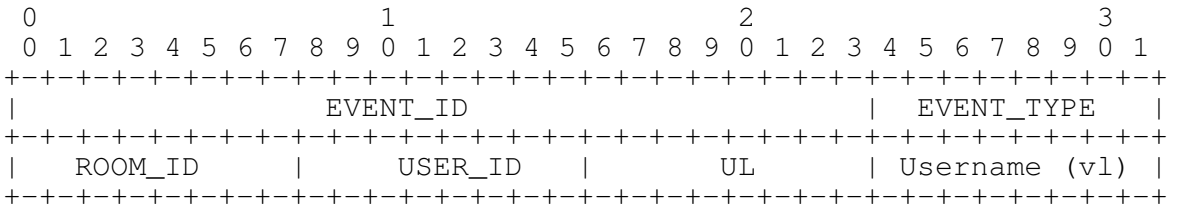


Figure 20

EVENT_TYPE: NEW_USER = 0x2

This event represents an arrival of a user on the server.

EVENT_ID: 3 bytes:

Unique ID of the event generated by the server. MUST be generated in ascending order.

ROOM_ID: 1 byte

The ID of the room in which the new user arrives (by default the user arrives in main room, so the value is always 0x0).

USER_ID: 1 byte

The ID of the user who arrived on the server.

USERNAME_LENTGH: 1 bytes

The length of the username.

USERNAME: variable length

MUST be longer than 0 byte and shorter than 255 bytes.

8.3. Event Type: SWITCH_ROOM

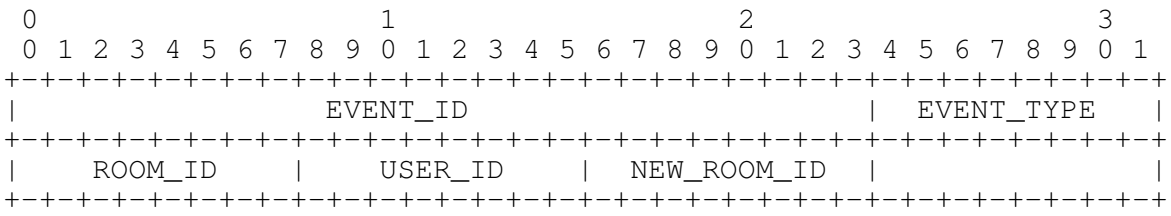


Figure 21

EVENT_TYPE: SWITCH_ROOM = 0x3

This event represents the fact that a user switched to a new room.

EVENT_ID: 3 bytes:

Unique ID of the event generated by the server. MUST be generated in ascending order.

ROOM_ID: 1 byte

The ID of the room in which the user was before moving to the new room.

USER_ID: 1 byte

The ID of the user who arrived on the server.

NEW_ROOM_ID: 1 byte

The ID of the room where the user moved.

8.4. Event Type: LOGOUT

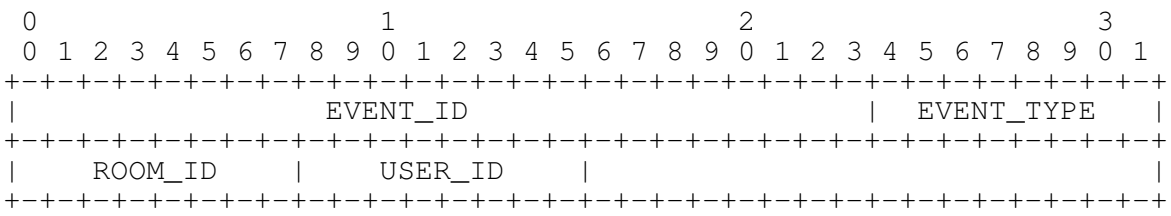


Figure 22

EVENT_TYPE: LOGOUT = 0x4

EVENT_ID: 3 bytes:

Unique ID of the event generated by the server. MUST be generated in ascending order.

ROOM_ID: 1 byte

The ID of the room in which the user was before logging out.

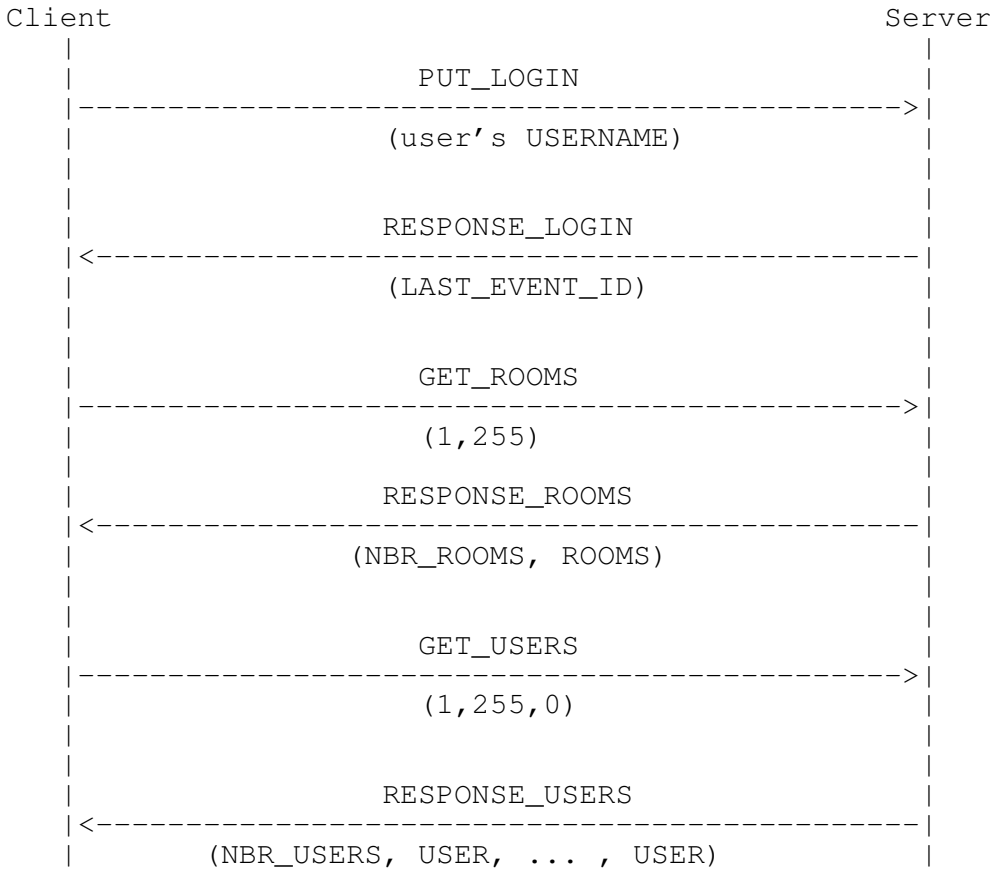
USER_ID: 1 byte

The ID of the user who arrived on the server.

9. Examples

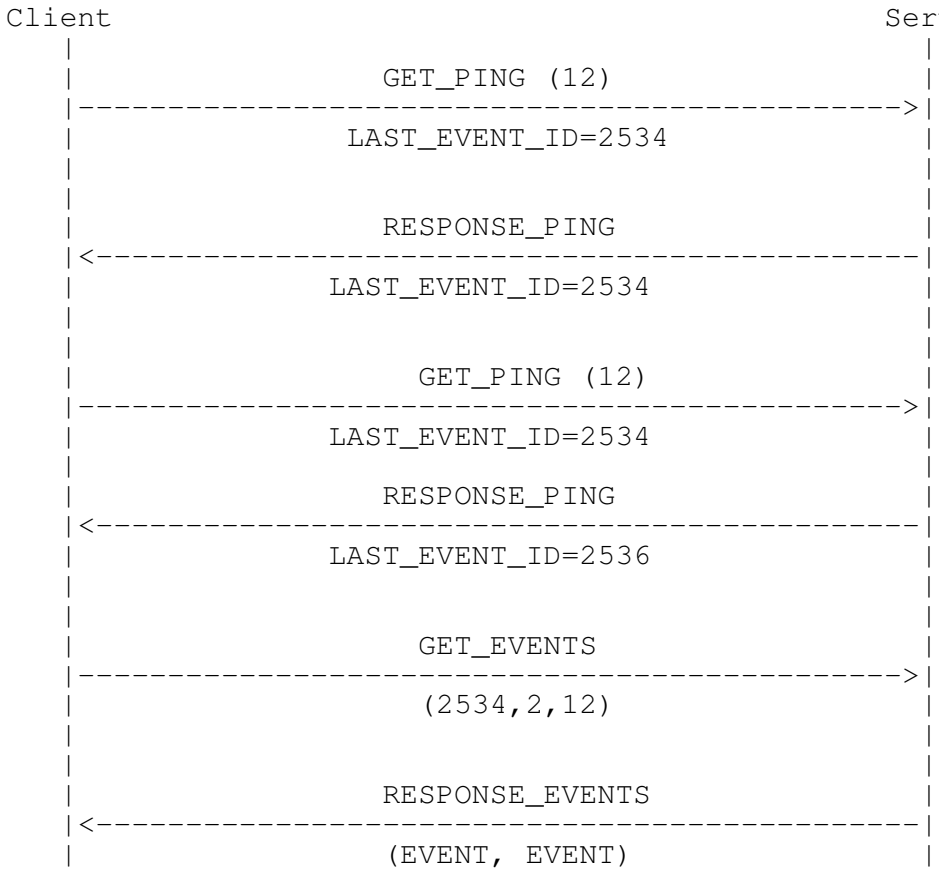
9.1. Scenario 1: logging in

The user client logs in and joins the main room.



9.2. Scenario 2: Retrieve new messages

The user in a room gets the last events that happened in the room. We assume here that the last event ID known by the client is 2534 and that the user is in room 12.



9.3. Misbehavior examples

Logging in while already connected: PUT_LOGIN is sent by the client. The server reply with RESPONSE_LOGIN rejecting the connection if the username is the same. Otherwise, it will create a second connection with as a different user. The first one will stay active.

Trying to join an nonexistant room: PUT_SWITCH_ROOM is sent by the client. The server cannot identify the destination room. It replies with RESPONSE_SWITCH_ROOM with the field STATUS_CODE set to "error".

Sending a message in an other room: PUT_NEW_MESSAGE is sent by the client with a ROOM_ID different from the ROOM_ID corresponding to the room where the user is. The server refuses the message, responding with RESPONSE_NEW_MESSAGE notifying the refusal.

9.4. Packet examples

Suppose a client would like to connect to the server video.exemple.com and watch a video on it.

To connect to the server, the user may input server name: video.exemple.com, port number: 8888, username: Anon12. It should send to the server the following message (in hexadecimal notation):

- Message Type: 00
- SEQ_NUMBER: 00 00
- User ID: 00
- Message Length: 07
- Username Length: 06
- Username: 41 6e 6f 6e 31 32

If the login is succesful, the server will reply with the following message, providing the client with his ID and the ID of the last event in the main room:

- Message Type: 01
- SEQ_NUMBER: 00 00
- User ID: 00
- Message length: 05
- Status code: 00
    - User ID: 08
- Last event ID: 00 22 4b

The client (which got the ID 08) will now retrieve the list of the rooms. First, for the room, the client sends a GET_USER request:

- Message Type: 08
- SEQ_NUMBER: 00 01
- User ID: 08
- Message length: 02
- First room ID: 00
- Number rooms: FF

The server will reply with the list of all movies available. For instance, if there are two videos: (Room ID: 01, IP: 05 74 7D 10, Port number: 02 22, Room name: video example, Nbr users: 5) and (Room ID: 02, IP: 11 77 01 44, Port number: 88 88, Room name: louis-san, Nbr users: 10)

- Message Type: 09
- SEQ_NUMBER: 00 01
- User ID: 00
- Message length: 29
- Number rooms: 02
- Room ID: 00
- IP: 05 74 7D 10
- Port number: 02 22
- Room number length: 0D
- Room name: 76 69 64 65 6f 20 65 78 61 6d 70 6c 65
- Number users: 05
- Room ID: 01
- IP: 11 77 01 44
- Port number: 88 88
- Room number length: 0D
- Room name: 6c 6f 75 69 73 2d 73 61 6e
- Number users: 0A

Suppose the user wants to join the room "video example":

- Message Type: 0C
- SEQ_NUMBER: 00 03
- User ID: 08
- Message length: 01
- Room ID: 01

The server will reply, confirming the operation has been succesful.

- Message Type: 0D
- SEQ_NUMBER: 00 03
- User ID: 00
- Message length: 01
-Status code: 00

Suppose user sends "Hello" in the chat room. (The acknowledgement is not presented here).

- Message Type: 0E
- SEQ_NUMBER: 00 04
- User ID: 08
- Message length: 01
- Room ID: 01
- Message length: 05
- Message: 48 65 6c 6c 6f

The following examples will deal with how the client receives the events from the server. Suppose the user is in the main room (room no. 0) and that two event occured since his last update: a message

("Hi everyone!") was sent by another user (ID: 02) and a user (ID: 0E) left a movie room (0x05) to come back to the main room. First, the client will send a ping to the server to get noticed if some event happened:

- Message Type: 04
- SEQ_NUMBER: 00 05
- User ID: 08
- Message length: 01
- Last event ID: 00 00 D4
    - Room: 00

The server will send the ID of the last event in the user room (i.e. main room).

- Message Type: 05
- SEQ_NUMBER: 00 05
- User ID: 00
- Message length: 01
- Last event ID: 00 00 D6

The client now can check the difference between the ID of the last event it knows and the ID it received. Thus, the client knows that 2 new events happened since last update. He will send a GET_EVENTS to retrieve them.

- Message Type: 06
- SEQ_NUMBER: 00 06
- User ID: 08
- Message length: 04
- Last event ID: 00 00 D4
- Number events: 02
    - Room: 00

The server, upon receiving this message, has to send back the 2 events which occured right after the one corresponding to the ID notified.

- Message Type: 07
- SEQ_NUMBER: 00 06
- User ID: 00
- Message length: 04
- Number events: 02
- Event ID: 00 00 D5
- Event type: 01
    - Room ID: 00
- User ID: 02
- Message length: 0C
- Message: 48 69 20 65 76 65 72 79 6f 6e 65 21
- Event ID: 00 00 D6
- Event type: 03
    - Room ID: 05
- User ID: 0E
- New room ID: 00

10. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

Authors' Addresses

Loic Carr
Telecom Bretagne
Plouzane
France

Email: loic.carr@telecom-bretagne.eu

Guillaume Buret
Telecom Bretagne
Loft Manganime souvent
Plouzane
France

Email: guillaume.buret@telecom-bretagne.eu

Alexis Lechat
Telecom Bretagne
Loft Manganime
Plouzane
France

Email: alexis.lechat@telecom-bretagne.eu

Quentin Jodelet
Telecom Bretagne
Loft Manganime
Plouzane
France

Email: quentin.jodelet@telecom-bretagne.eu