

## Formato de los mensajes del protocolo.

### Formatos del protocolo UDP (Servidor DNS)

FORMATOS: Binario

Mensajes de control:

Opcode (1 Byte)

Formato con un parámetro:

Opcode (1 Byte)	Parámetro1 (4 bytes)

Formato con dos parámetros:

Opcode (1 Byte)	Parámetro1 (4 bytes)	Parámetro2 (4 bytes)

*Tipos y descripción de mensajes:*

Mensaje: RegisterServer( Opcode = 2)

Formato: TwoParameter

Sentido de la comunicación: Servidor de chat -> Directorio

Descripción: Mensaje enviado por el servidor de chat al directorio a modo de solicitud de registro de su dirección, especificando mediante dos enteros de 4 bytes el puerto Num\_PORT y el protocolo Protocol\_ID para el que opera.

Ejemplo:

Opcode (1 Byte)	Protocol_ID (4 bytes)	Num_PORT (4 bytes)
2	127	6868

Mensaje: RegistrationOk( Opcode = 1)

Formato: Control

Sentido de la comunicación: Directorio -> Servidor de Chat

Descripción: Mensaje enviado por el directorio a un servidor de chat como respuesta a una solicitud de registro, informando a este de que se ha efectuado correctamente.

Ejemplo:

Opcode (1 Byte)
2

Mensaje: Empty( Opcode = 0 )

Formato: Control

Sentido de la comunicación: Directorio -> Cliente

Descripción: Mensaje enviado por el directorio para informar de la ausencia de información sobre el servidor buscado para cierto protocolo.

Ejemplo:

Opcode (1 Byte)
0

Mensaje: QueryProtocol ( Opcode = 3)

Formato: OneParameter

Sentido de la comunicación: Cliente -> Directorio

Descripción: Mensaje enviado por el cliente del chat al directorio a modo de solicitud de la dirección del servidor para el protocolo especificado en el campo Protocol\_ID, el cual es un entero de 4 bytes.

Ejemplo:

Opcode (1 Byte)	Protocol_ID (4 bytes)
3	127

Mensaje: ServerInfo( Opcode = 4)

Formato: TwoParameter

Sentido de la comunicación: Directorio -> Cliente

Descripción: Contiene la respuesta al Query realizado por el cliente, y adjunta la dirección Ip y el puerto del protocolo solicitado.

Ejemplo:

Opcode (1 Byte)	Ip (4 bytes)	Num_PORT (4 bytes)
4	127.0.0.1	6868

## Formato de mensajes TCP

### Tipo de formato: Field-Value

Nuestro identificador de protocolo, calculado como la suma los DNIs de ambos miembros del grupo de prácticas, es 127, por lo que nos corresponde usar el formato de mensajes basado en

**Field-Value.**

#### Formato

##### *InternalMessage*

**Descripción:** operaciones sin parámetros que informan de un hecho concreto al servidor o al cliente, como por ejemplo puede ser informar del éxito de una petición del cliente.

**Fields:** operation.

**Apariencia:**

operation: [operación perteneciente a operaciones disponibles]\n

\n

##### *RoomMessage*

**Descripción:** en algunas ocasiones es necesario intercambiar información con un parámetro, por ejemplo, a la hora de entrar a una sala, es necesario indicar la sala concreta. Dan soporte a mensajes con un parámetro.

**Fields:** Operation, value.

**Apariencia:**

operation: [operación perteneciente a operaciones disponibles]\n

value:[cadena de texto]\n

\n

##### *ChatMessage*

**Descripción:** surge de la necesidad de enviar mensajes de chat. Un mensaje de chat viene identificado por un Nick, y un mensaje. Por tanto, necesita utilizar dos parámetros.

**Fields:** operation, value, message.

**Apariencia:**

operation: [operación perteneciente a operaciones disponibles]\n

value:[cadena de texto]\n

*message:[cadena de texto]\n*

*\n*

### *ListMessage*

**Descripción:** puede surgir la necesidad de enviar un número *indeterminado* de cadenas. Este tipo de mensajes realizan contienen listas de listas de cadenas.

**Fields:** operation, elements.

### **Apariencia:**

*operation: [operación perteneciente a operaciones disponibles]\n*

*value:[cadena1, cadena2, cadena3, (...), cadenaN]\n*

*\n*

## Detalles sobre los principales aspectos de implementación

Implementación del formato de los mensajes.

*Propósito*

Implementar de una manera coherente al diseño anterior los diferentes mensajes que existen.

*Correspondencia*

Mensaje Diseño	Implementación
RoomMessage	NCRoomMessage.java
ChatMessage	NCChatMessage.java
InternalMessage	NCInternalMessage.java
InternalListMessage	NCInternalMessage.java

## Implementación

### Clase NCMessage

La clase NCMessage es una clase abstracta que predefine las propiedades generales de los mensajes:

#### -Opcode:

Definición de un identificador único, publico y final para cada tipo de mensaje.

Algunos ejemplos:

```
public static final byte OP_ACESS_DENIED = 1;
public static final byte OP_BAN_FROM_ROOM = 2;
public static final byte OP_CHANGE_ROOM_NAME = 3;
public static final byte OP_CHAT_MESSAGE = 4;
```

Se hace un control de Opcodes validos en esta misma clase, mediante el uso de estructuras de datos que almacenan todos los Opcode válidos. En caso de dar fallo se predefine un Opcode especial **OP\_INVALIDE\_COD**.

#### Funcionalidad:

##### -Factoría de mensajes:

Para obtener una instancia de un mensaje se ofrecen diferentes funciones para construir dichos mensajes:

```
public static NCMessage makeRoomMessage(byte code, String name);
public static NCMessage makeInternalListMessage(byte code, Collection<String> list);
public static NCMessage makeInternalMessage(byte code);
public static NCMessage makeChatMessage(byte code, String nick, String message);
```

#### -Lectura de mensajes:

Los mensajes deben de ser **convertibles a cadenas de texto**, y deben de ser **extraíbles a partir de una cadena de texto**, es decir debemos implementar una función:

ConvertirACadenaCodificada(Mensaje) = Cadena texto

LeerDesdeCadena( ConvertirACadenaCodificada(Mensaje) )= Mensaje

#### Conversión a cadena codificada

*Misión: convertir a cadena de texto de acuerdo con el diseño de mensajes.*

```
public final String toEncodedString() {
    StringBuffer sb = new StringBuffer();
    sb.append(OPCODE_FIELD+DELIMITER+opcodeToOperation(opcode)
            + END_LINE);
    sb.append(toBufferedString());
    sb.append(END_LINE);
    return sb.toString();
}
```

Se crea una cadena como la siguiente:

---

Opcode:[OPCODE]

..... **toBufferedString()** .....

END\_LINE

---

```
protected abstract StringBuffer toBufferedString();
```

Se crea una cadena común y mediante la función abstracta **toBufferedString()** cada tipo de mensaje inserta su parte.

Cada implementación particular deberá tener su propio **readFromString()** que convertirá el mensaje en cadena a una instancia de su clase de mensaje.

### Lectura desde un canal de entrada

Permite leer el mensaje mediante un flujo de entrada. Una vez leído el mensaje, separa su Opcode de su contenido en cadena y redirige esa información al **readFromString()** de su tipo de clase (identificada inequívocamente por su Opcode).

### Clase NCInternalMessage

Es la clase más básica de mensajes, que **informa de un hecho muy específico**.



Solo tiene el campo OPCODE y por tanto su funcionalidad es crear una instancia valida de NCMessage.

```
public NCInternalMessage(byte type){
    this.opcode = type;
}
protected StringBuffer toBufferedString(){
    return new StringBuffer();
}
public static NCInternalMessage readFromString(byte code) {
    return new NCInternalMessage(code);
}
```

Nótese que el buffer no inserta ningún dato.

---

**toBufferedString()** → VACIO

---

Mensaje creado:

---

Opcode:[OPCODE]

END\_LINE

---

### Clase NCListMessage

Su cometido es enviar un número indeterminado de cadenas de texto. Internamente se separan dichas cadenas por comas. Ofrece método de obtención de dichas cadenas:

public List<String> getList()

```
String listed = list.stream()
    .map(n -> String.valueOf(n))
    .collect(Collectors.joining(delimiter: ",", ""));
```

Para lograr codificar el mensaje en cadena simplemente se crea una cadena que une todos los elementos de la colección de cadenas insertar junto a su delimitador:

Lista: "elem1" "elem2" "elem3" >>>>>> "elem1, elem2, elem3"

```
protected StringBuffer toBufferedString() {
    StringBuffer sb = super.toBufferedString();
    String listed = list.stream()
        .map(n -> String.valueOf(n))
        .collect(Collectors.joining(delimiter: ",", ""));
    sb.append(LIST_FIELD+DELIMITER+listed+END_LINE);
    return sb;
}
```

Y mediante el uso del toBufferedString() se inserta en el texto:

---

toBufferedString() → elements:elem1,elem2,elem3

---

Mensaje creado:

---

Opcode:[OPCODE]

Elements:elem1,elem2,elem3

END\_LINE

---

Para recuperar el mensaje a partir de la cadena el procedimiento consiste en separar por sus apartados mediante el **delimitador general** ["\n"] y procesar individualmente cada apartado.

En este caso el apartado de lista se vuelve a separar por su **delimitador particular** "," y se inserta en una lista.

```
public static NCInternalListMessage readFromString(byte code, String message) {
    Collection<String> list = null;
    String[] lines = message.split(String.valueOf(END_LINE));
    int idx = lines[1].indexOf(DELIMITER); // Posición del delimitador
    String field = lines[1].substring(beginIndex: 0, idx).toLowerCase(); // minúsculas
    String value = lines[1].substring(idx + 1).trim();
    if (field.equalsIgnoreCase(LIST_FIELD))
        list = Arrays.asList(value.split(regex: ","));
    return new NCInternalListMessage(code, list);
}
```

### Clase NCRoomMessage

Dirigida en esencia a mensajes que necesitan algún tipo de parámetro, por ejemplo, un nick o un nombre de sala.

Ofrece la posibilidad de recuperar dicho valor mediante la operación `getValue()`.

El parámetro es de tipo cadena y su codificación es directa. Para obtener el mensaje en formato de cadena la inserción se produce de la siguiente manera:

```
@Override
protected StringBuffer toBufferedString() {
    StringBuffer sb = new StringBuffer();
    sb.append(NAME_FIELD+DELIMITER+value+END_LINE); //Construimos el campo
    return sb;
}
```

---

`toBufferedString()` → “parameter”

---

Mensaje creado:

---

Opcode:[OPCODE]

**Value:parameter**

END\_LINE

---

La lectura, es igualmente sencilla:

```
public static NCRoomMessage readFromString(byte code, String message) {
    String[] lines = message.split(String.valueOf(END_LINE));
    String name = null;
    int idx = lines[1].indexOf(DELIMITER); // Posición del delimitador
    String field = lines[1].substring(beginIndex: 0, idx).toLowerCase();
    String value = lines[1].substring(idx + 1).trim();
    if (field.equalsIgnoreCase(NAME_FIELD))
        name = value;
    return new NCRoomMessage(code, name);
}
```

### Clase NCRoomChatMessage

Es una clase que surge de la necesidad de usar dos tipos de parámetros, en concreto da soporte a mensajes ya sean notificaciones de mensajes o, mensajes privados. Al ser heredera de `NCRoomMessage` su cometido es tan solo agregar un parámetro. Por tanto, soporta la función `getValue()` del tipo anterior y le agrega `getMessage()`. **Para más información sobre el `NCRoomMessage`, ver el apartado anterior.**

Lo hace de una manera totalmente idéntica al mensaje anterior:

```
@Override
protected StringBuffer toBufferedString() {
    StringBuffer sb = super.toBufferedString();
    sb.append(MESSAGE_FIELD+DELIMITER+message+END_LINE);
    return sb;
}
```

---

**toBufferedString()** → "message"

---

Mensaje creado:

---

Opcode:[OPCODE]

Value:parameter

**Message:** message

END\_LINE

---

Para la lectura de mensajes se realiza una separación por delimitador y se rescatan los valores. Como es habitual los nombres de apartados sirven para detectar errores

```
protected static NCMessage getreadFromString(byte code, String message) {

    String[] lines = message.split(String.valueOf(END_LINE));
    String nick = null;
    int idx = lines[1].indexOf(DELIMITER); // Posición del delimitador
    String field = lines[1].substring(beginIndex: 0, idx).toLowerCase();
    String value = lines[1].substring(idx + 1).trim();
    if (field.equalsIgnoreCase(NAME_FIELD))
        nick = value;

    String msg = null;
    idx = lines[2].indexOf(DELIMITER); // Posición del delimitador
    field = lines[2].substring(beginIndex: 0, idx).toLowerCase(); // minúsculas
    value = lines[2].substring(idx + 1).trim();
    if (field.equalsIgnoreCase(MESSAGE_FIELD))
        msg = value;
    else
        throw new IllegalArgumentException(s: "Unknown field");

    return new NCChatMessage(code, msg, nick);
}
```

Dado su propósito de naturaliza especifica se agrega un método que da una cadena formateada del mensaje:

```
public String toPrintableString() {
    return getValue() + ": " + getMessage();
}
```



## Mecanismo de gestión de salas.

Nota sobre las salas

Las salas creadas por el sistema a la hora de iniciar el servidor *no tienen administrador inicialmente*. Por tanto, *nadie podrá ser administrados jamás, y, en consecuencia, las operaciones sujetas a administradores no estarán permitidas, entre ellas el baneo de usuarios*. Esas salas *tampoco pueden ser borradas*.

### Creación

La creación de una sala comienza con la petición del cliente con el opcode asociado a la operación "Create\_room".

#### Lado del cliente

En estado **registrado** del autómata del cliente es posible procesar el comando create\_room <room>.

```
case NCCommands.COM_CREATE_ROOM:
    if(clientStatus == REGISTERED){
        makeRoom();
    } else{
        System.out.println("* You must register a nickname in first place. Please use nick (desired nick)");
    }
    break;
```

Dicha petición será procesada y en caso de éxito el estado del autómata pasará ser **IN\_ROOM**. Es decir, *crear una sala implicara, en caso de existir, entrar en esa sala*.

```
private void makeRoom() {
    boolean result = false;

    try {
        result = ncConnector.makeRoom(room);
    } catch (IOException e) {
        // TOD Auto-generated catch block
        e.printStackTrace();
    }

    if (!result) {
        System.out.println("* Room already exists");
        return;
    }

    System.out.println("* Your are now in: " + room);
    clientStatus = IN_ROOM;

    do {
        readRoomCommandFromShell();
        processRoomCommand();
    } while (currentCommand != NCCommands.COM_EXIT && clientStatus == IN_ROOM);
    System.out.println("* Your are out of the room");

    clientStatus = REGISTERED;
}
```

La comunicación con el servidor será dirigida por **ncconector** y consistirá en mandar una petición de crear una sala. Dicha petición puede resultar **exitosa**, si recibimos **OP\_OK** o **incorrecta** si recibimos **OP\_ROOM\_DUPLICATED**, que nos informa de que esa sala ya existe. En otro caso, habría ocurrido un error desconocido.

```
public boolean makeRoom(String room) throws IOException {
    NCRoomMessage message = (NCRoomMessage) NCMessage.makeRoomMessage(NCMessage.OP_CREATE_ROOM, room);
    String rawMessage = message.toEncodedString();
    dos.writeUTF(rawMessage);
    NCInternalMessage response = (NCInternalMessage) NCMessage.readMessageFromSocket(dis);
    switch (response.getOpcode()) {
        case NCMessage.OP_OK:
            return true;
        case NCMessage.OP_ROOM_DUPLICATED:
            System.out.println("The room " + room + " already exists");
            return false;
        default:
            throw new IllegalStateException("Unexpected opcode: " + response.getOpcode());
    }
}
```

#### *Lado del servidor*

Esta operación desde el punto de vista del servidor consiste en recibir una petición de creación de sala en el estado **procesando\_general\_commands**.

```
case NCMessage.OP_CREATE_ROOM:
    makeRoom((NCRoomMessage) message);
    break;
```

El procesamiento del mensaje conducirá a la operación de creación:

```
private void makeRoom(NCRoomMessage message) {
    byte opcode;
    String roomName = message.getValue();
    boolean success = false;

    if (!serverManager.registerRoomManager(roomName)) {
        opcode = NCMessage.OP_ROOM_DUPLICATED;
        System.out.println("Room " + roomName + " already exists");
    } else {
        opcode = NCMessage.OP_OK;
        System.out.println("** Room " + roomName + " created." + "by " + user);
        success = true;
    }

    NCInternalMessage response = (NCInternalMessage) NCMessage.makeInternalMessage(opcode);
    String rawMessage = response.toEncodedString();
    try {
        dos.writeUTF(rawMessage);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

El servidor pedirá al server manager el registro de la sala, e informará al cliente del resultado (exitoso o no).

El server manager **verifica** la existencia de una sala con el mismo nombre, y si esta no existe la crea y añade dentro de las salas que gestiona.

```
//Método para registrar un RoomManager
public boolean registerRoomManager(String id) {
    if (rooms.containsKey(id)) {
        return false;
    }
    NCRoomManager roomManager = new NCRoomManager(id);
    rooms.put(id, roomManager);

    return true;
}
```

En caso de éxito se añade al usuario que creo la sala como **administrador** y pasa al estado **procesando\_room\_messages**.

```
if (sucess) {
    roomManager = serverManager.enterRoom(user, roomName, socket);
    roomManager.addAdmin(user);
    currentRoom = roomName;

    try {
        processRoomMessages();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

## Eliminación

Una sala se eliminará si se queda **vacía y no es creada por el propio servidor** [Ver notar inicial].

Lo hará el propio servidor en el proceso de salida [ver proceso de salida].

## Entrada a la sala

### Lado del cliente

Caso similar al anterior, el cliente inicia la petición si reconoce un comando de entrada:

```
case NCCommands.COM_ENTER:

    if(clientStatus == REGISTERED){
        enterChat();
    }
    else
        System.out.println("** You must register a nickname in first place. Please use nick (desired nick)");

    break;
```



Y hace uso de Ncconnector para enviar la petición al servidor:

```
private void enterChat() {
    boolean result = false;
    try {
        result = ncConnector.enterRoom(room);
    } catch (IOException e) {
        e.printStackTrace();
    }
    if(!result) {
        System.out.println("* The room does not exist");
        return;
    }
}
```

```
public boolean enterRoom(String room) throws IOException {
    //Funcionamiento resumido: SND(ENTER_ROOM<room>) and RCV(IN_ROOM) or RCV(REJECT)
    NCRoomMessage message = (NCRoomMessage) NCMessage.makeRoomMessage(NCMessage.OP_ENTER_ROOM, room);
    dos.writeUTF(message.toEncodedString());
    NCInternalMessage response = (NCInternalMessage) NCMessage.readMessageFromSocket(dis);

    switch(response.getOpcode()) {
        case NCMessage.OP_OK:
            return true;
        case NCMessage.OP_INVALID_ROOM:
            System.out.println("The room " + room + " is not valid");
            return false;
        case NCMessage.OP_ACCESS_DENIED:
            System.out.println("The room " + room + " is not open");
            return false;
        default:
            throw new IllegalStateException("Unexpected opcode: " + response.getOpcode());
    }
}
```

Ncconnector reconoce la respuesta de entrada exitosa si recibe el opcode asociado al OK. **En caso de fallo** el cliente recibirá un opcode asociado a sala invalida o acceso denegado. **En la implementación actual del servidor siempre se recibirá Acces\_Denied.**

```

,
if(!result) {
    System.out.println("* The room does not exist");
    return;
}

System.out.println("* You are now in: " + room);
clientStatus = IN_ROOM;
do {
    //Pasamos a aceptar sólo los comandos que son válidos dentro de una sala
    readRoomCommandFromShell();
    processRoomCommand();
} while (currentCommand != NCCommands.COM_EXIT && clientStatus == IN_ROOM);
System.out.println("* You are out of the room");
clientStatus = REGISTERED;

```

**En caso de fallo** se informa al usuario del error.

**En caso de éxito** se entra al estado **IN\_ROOM** y se procede a procesar comandos asociadas al estado indicado.

*Lado del servidor*

```

case NCMessage.OP_ENTER_ROOM:
    String room = ((NCRoomMessage) message).getValue();
    if (enterRoom(room))
        processRoomMessages();
    break;

```

En el caso del servidor, primero intenta entrar a la sala, si es posible cambia de estado y empieza a procesar mensajes asociados al estado **Process\_Room\_Message**.

En el proceso de EnterRoom(room) se pedirá al server manager la sala correspondiente.

```

NCRoomManager rm = serverManager.enterRoom(user, room, socket);

```

El server busca la sala entre las disponibles y intenta registrar en ella al usuario. Si el usuario no ha sido baneado, ni está en la sala, la sala registrara al usuario:

```

public synchronized NCRoomManager enterRoom(User u, String room, Socket s) {

    NCRoomManager rm = rooms.get(room);
    if(rm != null && rm.registerUser(u, s)) { //Si la sala existe y se ha podido registrar al usuario
        return rm;
    }

    return null;
}

```

Si el usuario está bloqueado el **proceso de registro falla o la sala no existe** el server manager informa al server thread mediante el código de error **null**.

```

public boolean registerUser(User u, Socket s) {

    if(blocked.contains(u)){
        return false;
    }

    boolean result = false;
    try {
        result = users.putIfAbsent(u, new DataOutputStream(s.getOutputStream())) == null;
    } catch (IOException e) {
        e.printStackTrace();
    }
    return result;
}

```

Nótese que el server thread no conoce el motivo del fallo, en consecuencia, no puede informar con exactitud al cliente. En caso de que falle envía un **ACCES\_DENIED**.

```

if(rm == null) {
    System.out.println("User " + user + " not accepted in the room " + room);
    NCInternalMessage msg = new NCInternalMessage(NCMessage.OP_ACCESS_DENIED);
    String rawResponse = msg.toEncodedString();
    try {
        dos.writeUTF(rawResponse);
    } catch (IOException e) {
        e.printStackTrace();
    }
    return false;
}

```

En **caso de éxito** se informa al cliente:

```

System.out.println("User " + user + " joined room " + room);
NCInternalMessage msg = new NCInternalMessage(NCMessage.OP_OK);
String rawResponse = msg.toEncodedString();
currentRoom = room;
roomManager = rm;

try {
    dos.writeUTF(rawResponse);
} catch (IOException e) {
    e.printStackTrace();
}

return true;

```

Y como se mostraba al inicio se comenzarán a procesar los mensajes asociados a las salas:

```

case NCMessage.OP_ENTER_ROOM:
    String room = ((NCRoomMessage) message).getValue();
    if (enterRoom(room))
        processRoomMessages();
    break;

```

## Salida

El proceso de salida al igual que todos, comienza con una petición del cliente al leer el comando pertinente.

### *Lado del cliente*

Simplemente se envía la informa de la salida al servidor y se cambia de estado.

```

private void exitTheRoom() {
    try {
        ncConnector.leaveRoom(room);
        clientStatus = REGISTERED;
    } catch (IOException e) {
        e.printStackTrace();
    }
}

public void leaveRoom(String room) throws IOException {
    NCInternalMessage message = (NCInternalMessage) NCMessage.makeInternalMessage(NCMessage.OP_EXIT_ROOM);
    dos.writeUTF(message.toEncodedString());
}

```

### *Lado del servidor*

El servidor recibe la petición de salida del usuario y encarga al server manager el abandono de la sala en concreto:

```
case NCMessages.OP_EXIT_ROOM: {
    serverManager.leaveRoom(user, currentRoom);
    roomMessage = (NCRoomMessage) NCMessages.makeRoomMessage(NCMessages.OP_LEFT_USER, user.getName());
    roomManager.broadcastSystemMessage(roomMessage);
    roomManager = null;
    currentRoom = null;
    exit = true;
    break;
}
```

También cambia de estado y advierte al resto de usuarios de la salida.

El server manager elimina el usuario de la sala:

```
public synchronized void leaveRoom(User u, String room) {
    NCRoomManager rm = rooms.get(room);
    if (rm == null) {
        return;
    }
    rm.removeUser(u);
    if (!defaultRooms.contains(room)) {
        {
            if (rm.usersInRoom() == 0) {
                rooms.remove(room);
            } // If there are no users in the room, remove it
            else if (rm.adminsInRoom() == 0) {
                rm.addRandomAdmin();
            } // Select random admin
        }
    }
}
```

Lo primero que se procede es a **eliminar el usuario** de la sala:

```
public void removeUser(User u) {
    removeAdmin(u);
    users.remove(u);
}
```

Esto implica **eliminar** también su **categoría de administrador**.

Si la sala es creada por el sistema, se termina el proceso. No obstante, en caso contrario si se queda **vacía se elimina** y si al eliminar el usuario **no hay admins** en la sala se asigna un **administrador aleatorio**.

## Mejoras adicionales implementadas.

Notificar a los demás usuarios de la sala las entradas/salidas de otros usuarios en las salas conforme se vayan produciendo

Esta mejora es muy simple, pues tan solo tenemos que realizar un broadcast de un mensaje interno de tipo RoomMessage que informa de la entrada/salida de un usuario.

```
serverManager.leaveRoom(user, roomManager.getRoomName());
roomMessage = (NCRoomMessage) NCMessage.makeRoomMessage(NCMessage.OP_LEFT_USER, user.getName());
roomManager.broadcastSystemMessage(roomMessage);

String entryWarning = user.getName();
roomMessage = (NCRoomMessage) NCMessage.makeRoomMessage(NCMessage.OP_ENTERED_USER, entryWarning);
roomManager.broadcastSystemMessage(roomMessage);
```

Y esto se realiza cada vez que un usuario entre a una sala o sale de una sala.

## La posibilidad de renombrar una sala.

*Creo que el cómo se ha añadido el comando no aporta mucho a la práctica, por lo tanto, lo resumiré brevemente, sin aportar el código. Se crea una nueva constante asociada al comando, y se añade al vector de comandos válidos, y su valor de cadena, así como la línea correspondiente en NCCommands. Se añade por otra parte la verificación de argumentos en NCShell en el método para leer comandos de sala.*

### Lado de cliente

El renombramiento de la sala surge de la petición de un usuario para ello. Como todas las peticiones, la implementación es bastante directa, y comienza cuando el cliente lee el comando `change_room_name <nuevo_nombre>`.

El cliente hace uso de la clase Ncconnector para realizar la petición y informa al usuario en caso de que el nombre no se haya cambiado:

```
private void processChangeRoomNameRequest() {
    boolean success = false;
    try {
        success = ncConnector.changeRoomName(room);
    } catch (IOException e) {
        e.printStackTrace();
    }
    if(!success) {
        System.out.println("* The room name has not been changed");
    }
}
```

```

public boolean changeRoomName(String room) throws IOException {
    NCRoomMessage message = (NCRoomMessage) NCMessage.makeRoomMessage(NCMessage.OP_CHANGE_ROOM_NAME, room);
    String rawMessage = message.toEncodedString();
    dos.writeUTF(rawMessage);
    NCInternalMessage response = (NCInternalMessage) NCMessage.readMessageFromSocket(dis);

    switch (response.getOpcode()) {
        case NCMessage.OP_OK:
            System.out.println("The room name has been changed");
            return true;
        case NCMessage.OP_ROOM_DUPLICATED:
            System.out.println("The room " + room + " already exists");
            return false;
        case NCMessage.OP_PERMISSION_INVALID:
            System.out.println("You are not an admin");
            return false;
        default:
            throw new IllegalStateException("Unexpected opcode: " + response.getOpcode());
    }
}

```

En NCConector enviamos el mensaje de tipo RoomMessage con el valor asociado al nuevo nombre deseado y esperamos un mensaje interno, que nos indicara el éxito o la causa posible de denegación de cambio de nombre.

#### Lado de Servidor

```

case NCMessage.OP_CHANGE_ROOM_NAME:
    roomMessage = (NCRoomMessage) message;
    String roomName = roomMessage.getValue();
    success = false;

    if (roomManager.isAdmin(user)) {

```

En el caso del servidor podemos encontrar un proceso más interesante. Al leer el mensaje se realiza la verificación asociada a los permisos, **es necesario ser administrador para cambiar de nombre de sala.**

En caso de que esta verificación no se cumpla se informa al usuario y se termina:

```

} else {
    response = (NCInternalMessage) NCMessage.makeInternalMessage(NCMessage.OP_PERMISSION_INVALID);
    rawResponse = response.toEncodedString();
    dos.writeUTF(rawResponse);
}
break;

```

En caso de que seamos administradores podemos continuar.

*Quiero aclarar antes de continuar este trozo de código que se verá a continuación:*

```

(roomManager.getRoomName(),

```

La variable originalmente propuesta **current\_room** puede llegar a ser problemática con esta función. ¿Qué ocurre si se cambia el nombre de la sala, y luego se crea una sala con el nombre antes de cambiar? Podría dar lugar a una **current\_room** asociada a un manager de sala diferente al que debería. Por ello, **es más recomendable preguntarle al manager su nombre directamente.** Otra solución, podría ser operar directamente con el objeto manager para las peticiones sobre una sala desde Server Thread y Server Manager.

```

if (roomManager.isAdmin(user)) {
    success = serverManager.changeRoomName(roomManager.getRoomName(), roomName);
    if (success) {

```

Volviendo al origen, ahora el servidor sabe que el usuario puede realizar el cambio solicitado y para ello pedirá al serverManager realizar dicho cambio.

```

public boolean changeRoomName(String currentRoom, String roomName) {
    NCRoomManager rm = rooms.get(currentRoom);
    if(rm != null && isRoomNameAvailable(roomName)) {
        rm.setRoomName(roomName);
        rooms.put(roomName, rm);
        rooms.remove(currentRoom);
        return true;
    }
    return false;
}

```

El server manager verificara si el nombre se encuentra libre. Si es así, realizara los cambios pertinentes para que el nuevo nombre se asocie a la sala en cuestión y se elimine el anterior. Informará al server Thread del éxito de la operación y este a su vez al cliente:

```

case NCMMessage.OP_CHANGE_ROOM_NAME:
    roomMessage = (NCRoomMessage) message;
    String roomName = roomMessage.getValue();
    success = false;

    if (roomManager.isAdmin(user)) {
        success = serverManager.changeRoomName(roomManager.getRoomName(), roomName);
        if (success) {
            response = (NCInternalMessage) NCMMessage.makeInternalMessage(NCMMessage.OP_OK);
            rawResponse = response.toEncodedString();
            dos.writeUTF(rawResponse);
            NCRoomMessage advertisingMessage = (NCRoomMessage) NCMMessage.makeRoomMessage(NCMMessage.OP_SYSTEM_MESSAGE, "RoomName is");
            rawResponse = advertisingMessage.toEncodedString();
            roomManager.broadcastSystemMessage(advertisingMessage);
        } else {
            response = (NCInternalMessage) NCMMessage.makeInternalMessage(NCMMessage.OP_ROOM_DUPLICATED);
            rawResponse = response.toEncodedString();
            dos.writeUTF(rawResponse);
        }
    } else {
        response = (NCInternalMessage) NCMMessage.makeInternalMessage(NCMMessage.OP_PERMISSION_INVALID);
        rawResponse = response.toEncodedString();
        dos.writeUTF(rawResponse);
    }
    break;

```

La posibilidad de crear nuevas salas en el servidor

**La creación de la sala se encuentra ya explicada en el mecanismo de gestión de salas. Realizo un “copia-pegar” aquí para ahorrar volver atrás en caso necesario:**



## Creación

La creación de una sala comienza con la petición del cliente con el Opcode asociado a la operación "Create\_room".

### *Lado del cliente*

En estado **registrado** del autómata del cliente es posible procesar el comando create\_room <room>.

```
case NCCommands.COM_CREATE_ROOM:
    if(clientStatus == REGISTERED){
        makeRoom();
    } else{
        System.out.println("** You must register a nickname in first place. Please use nick (desired nick)");
    }
    break;
```

Dicha petición será procesada y en caso de éxito el estado del autómata pasará ser **IN\_ROOM**. Es decir, *crear una sala implicara, en caso de existir, entrar en esa sala.*

```
private void makeRoom() {
    boolean result = false;

    try {
        result = ncConnector.makeRoom(room);
    } catch (IOException e) {
        // TOD Auto-generated catch block
        e.printStackTrace();
    }

    if (!result) {
        System.out.println("** Room already exists");
        return;
    }

    System.out.println("** Your are now in: " + room);
    clientStatus = IN_ROOM;

    do {
        readRoomCommandFromShell();
        processRoomCommand();
    } while (currentCommand != NCCommands.COM_EXIT && clientStatus == IN_ROOM);
    System.out.println("** Your are out of the room");

    clientStatus = REGISTERED;
}
```

La comunicación con el servidor será dirigida por **ncconnector** y consistirá en mandar una petición de crear una sala. Dicha petición puede resultar **exitosa**, si recibimos **OP\_OK** o **incorrecta** si recibimos **OP\_ROOM\_DUPLICATED**, que nos informa de que esa sala ya existe. En otro caso, habría ocurrido un error desconocido.

```

public boolean makeRoom(String room) throws IOException {
    NCRoomMessage message = (NCRoomMessage) NCMMessage.makeRoomMessage(NCMessage.OP_CREATE_ROOM, room);
    String rawMessage = message.toEncodedString();
    dos.writeUTF(rawMessage);
    NCInternalMessage response = (NCInternalMessage) NCMMessage.readMessageFromSocket(dis);
    switch (response.getOpcode()) {
        case NCMessage.OP_OK:
            return true;
        case NCMessage.OP_ROOM_DUPLICATED:
            System.out.println("The room " + room + " already exists");
            return false;
        default:
            throw new IllegalStateException("Unexpected opcode: " + response.getOpcode());
    }
}

```

#### *Lado del servidor*

Esta operación desde el punto de vista del servidor consiste en recibir una petición de creación de sala en el estado **procesando\_general\_commands**.

```

case NCMessage.OP_CREATE_ROOM:
    makeRoom((NCRoomMessage) message);
    break;

```

El procesamiento del mensaje conducirá a la operación de creación:

```

private void makeRoom(NCRoomMessage message) {
    byte opcode;
    String roomName = message.getValue();
    boolean success = false;

    if (!serverManager.registerRoomManager(roomName)) {
        opcode = NCMessage.OP_ROOM_DUPLICATED;
        System.out.println("Room " + roomName + " already exists");
    } else {
        opcode = NCMessage.OP_OK;
        System.out.println("** Room " + roomName + " created." + "by " + user);
        success = true;
    }

    NCInternalMessage response = (NCInternalMessage) NCMMessage.makeInternalMessage(opcode);
    String rawMessage = response.toEncodedString();
    try {
        dos.writeUTF(rawMessage);
    } catch (IOException e) {
        e.printStackTrace();
    }
}

```

El servidor pedirá al server manager el registro de la sala, e informará al cliente del resultado (exitoso o no).

El server manager **verifica** la existencia de una sala con el mismo nombre, y si esta no existe la crea y añade dentro de las salas que gestiona.

```
//Método para registrar un RoomManager
public boolean registerRoomManager(String id) {
    if (rooms.containsKey(id)) {
        return false;
    }
    NCRoomManager roomManager = new NCRoomManager(id);
    rooms.put(id, roomManager);

    return true;
}
```

En caso de éxito se añade al usuario que creo la sala como *administrador* y pasa al estado `procesando_room_messages`.

```
if (sucess) {
    roomManager = serverManager.enterRoom(user, roomName, socket);
    roomManager.addAdmin(user);
    currentRoom = roomName;

    try {
        processRoomMessages();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Enviar un mensaje privado a un usuario de la sala (sin que los demás puedan verlo).

Lado de cliente

Desde el lado de cliente esta mejora implica dos cambios: en primer lugar, recoger un comando como el `send_private <nick> <msg>` y realizar el envío hacia el servidor especificando el destinatario. Para que el comando acepte dos argumentos basta con agregar el primer argumento leído a la lista de argumentos.

```
case NCCommands.COM_SEND_PRIVATE:
    StringBuffer msg = new StringBuffer();
    if(st.hasMoreTokens())
        vars.add(st.nextToken());
    while (st.hasMoreTokens())
        msg.append(st.nextToken()+" ");
    vars.add(msg.toString());
    break;
```

En cuanto a la comunicación con el servidor es igual al resto de comandos, se lee el comando y se realiza una llamada a `NCConector` para comunicarse con el servidor:

```
public boolean sendPrivateMessage(String nickname, String chatMessage) throws IOException {
    NCChatMessage message = (NCChatMessage) NCMessage.makeChatMessage(NCMessage.OP_SEND_PRIVATE, nickname, chatMessage);
    String rawMessage = message.toEncodedString();
    dos.writeUTF(rawMessage);
    NCInternalMessage response = (NCInternalMessage) NCMessage.readMessageFromSocket(dis);
    switch (response.getOpcode()) {
        case NCMessage.OP_OK:
            System.out.println("The message has been sent");
            return true;
        case NCMessage.OP_NICK_INVALID:
            System.out.println("The user " + nickname + " is not in the room");
            return false;
        default:
            throw new IllegalStateException("Unexpected opcode: " + response.getOpcode());
    }
}
```

Las posibles respuestas del servidor pueden ser, o bien, un envío correcto, o bien el nick puede no existir en la sala.

Por otra parte, debemos preparar al cliente para poder recibir mensajes privados:

```
public String getIncomingMessage() throws IOException {
    String text = null;
    NCMessage message = NCMessage.readMessageFromSocket(dis);
    NCRoomMessage roomMessage = null;
    NCChatMessage chatMessage = null;

    switch (message.getOpcode()) {
        case NCMessage.OP_CHAT_MESSAGE:
            chatMessage = (NCChatMessage) message;
            text = chatMessage.toPrintableString();
            break;
        case NCMessage.OP_SYSTEM_MESSAGE:
            roomMessage = (NCRoomMessage) message;
            text = "System: " + roomMessage.getValue();
            break;
        case NCMessage.OP_GOT_BANNED:
            text = "You got banned";
            break;
        case NCMessage.OP_ENTERED_USER:
            roomMessage = (NCRoomMessage) message;
            text = "User " + roomMessage.getValue() + " entered the room";
            break;
        case NCMessage.OP_LEFT_USER:
            roomMessage = (NCRoomMessage) message;
            text = "User " + roomMessage.getValue() + " left the room";
            break;
        case NCMessage.OP_PRIVATE_CHAT_MESSAGE:
            chatMessage = (NCChatMessage) message;
            text = "Private message: " + chatMessage.toPrintableString();
            break;
        default:
            throw new IllegalStateException("Unexpected opcode: " + message.getOpcode());
    }

    return text;
}
```

Como se observa en este trozo de código hemos añadido la posibilidad de recibir mensajes privados.

#### Lado del servidor

Desde el punto de vista del servidor se debe realizar una búsqueda del usuario en la sala, y en caso de que este no exista, devolver el código de error (NICK\_INVALID). Si existe simplemente se envía a su destinatario el mensaje privado en cuestión, ya que la sala tiene su `DataOutputStream`. Veámoslo paso a paso:

1. Buscamos al usuario (destinatario)

```
case NCMMessage.OP_SEND_PRIVATE:
    chatMessage = (NCChatMessage) message;
    String privateMessage = chatMessage.getMessage();
    User toUser = serverManager.getUser(chatMessage.getValue());
    if (toUser != null && roomManager.isUser(toUser)) {
```

2. Si existe primero enviamos la confirmación y posteriormente le enviamos el mensaje:

```
if (toUser != null && roomManager.isUser(toUser)) {
    response = (NCInternalMessage) NCMMessage.makeInternalMessage(NCMMessage.OP_OK);
    rawResponse = response.toEncodedString();
    dos.writeUTF(rawResponse);
    chatMessage = (NCChatMessage) NCMMessage.makeChatMessage(NCMMessage.OP_PRIVATE_CHAT_MESSAGE, toUser.getName(), privateMessage);
    rawResponse = chatMessage.toEncodedString();
    roomManager.sendPrivateMessage(toUser, rawResponse); //The order matters: if we send the private message before the confirmatio
```

El orden de envío es importante ya que, si enviamos la confirmación primero, evitamos el caso especial de que un usuario se auto envíe un mensaje privado (en otro caso el autómata de cliente esperara respuesta del proceso del envío del mensaje, pero recibirá un mensaje privado)

- 2.1 El envío del mensaje se realizará a través de `roomManager`:

```
public void sendPrivateMessage(User user, String message) throws IOException {

    DataOutputStream dos = users.get(user);
    dos.writeUTF(message);

}
```

3. En otro caso se advertirá al cliente de que el envío no se ha podido realizar (el usuario no está en la sala).

```
} else {  
    response = (NCInternalMessage) NCMessage.makeInternalMessage(NCMessage.OP_NICK_INVALID);  
    rawResponse = response.toEncodedString();  
    dos.writeUTF(rawResponse);  
}  
break;
```

Ver el histórico de mensajes enviados a la sala (almacenados en el servidor)

Lado del servidor

En esta ocasión la implementación de la mejora comienza antes de la petición del cliente, lo hace con el almacenamiento de los mensajes privados.

```
public void broadcastMessage(User u, NCChatMessage message) throws IOException {  
  
    if(!users.containsKey(u)){  
        return ;  
    }  
  
    for (DataOutputStream dos : users.values()) {  
        dos.writeUTF(message.toEncodedString());  
    }  
  
    Instant now = Instant.now();  
    history.add "[" + now.toString() + "]" + " " + message.getValue() + ": " + message.getMessage();  
    lastMessageTime = now.toEpochMilli();  
}
```

Todo mensaje **de chat** que se envía se incluye en la historia. **No hay limite de mensajes, ni de tiempo, se almacenan todos los mensajes.**

Cuando se recibe el mensaje de chat del cliente solicitando la historia se siguen los siguientes pasos:

1. Se recupera la historia

```
case NCMessage.OP_HISTORY_REQUEST:  
    List<String> historyRequest = roomManager.getHistory(user);
```

- 1.1. Para ello se le pide una copia de la historia al roomManager

```
public List<String> getHistory(User u) {  
  
    if(!users.containsKey(u)){  
        return new ArrayList<String>();  
    }  
  
    return Collections.unmodifiableList(history);  
}
```

## 2. Se manda la historia como una internalListMessage.

```
NCInternalListMessage listMessage = (NCInternalListMessage) NCMessage.makeInternalListMessage(NCMessage.OP_HISTORY_RESPONSE,
historyRequest);
rawResponse = listMessage.toEncodedString();
dos.writeUTF(rawResponse);
break;
```

### Lado del cliente

Se inicia la petición con el comando history:

```
case NCCommands.COM_HISTORY:
    getAndShowHistory();
    break;
```

Se pide al servidor la historia a través de NCConector y se imprime la historia

```
private void getAndShowHistory() {
    String history = null;
    try {
        history = ncConnector.getHistory();
    } catch (IOException e) {
        e.printStackTrace();
    }

    System.out.println("The history of messages for this room and this minute is:");
    System.out.println("\n*****\n");
    System.out.println(history);
    System.out.println("\n*****\n");
}
```

NCConector pide la historia al servidor y da cierto formato a la misma antes de retornarla al NCControler.

```
public String getHistory() throws IOException {
    NCMessage message = (NCInternalMessage) NCMessage.makeInternalMessage(NCMessage.OP_HISTORY_REQUEST);
    String rawMessage = message.toEncodedString();
    dos.writeUTF(rawMessage);
    NCInternalListMessage response = (NCInternalListMessage) NCMessage.readMessageFromSocket(dis);

    if(response.getOpcode() != NCMessage.OP_HISTORY_RESPONSE)
        throw new IllegalStateException("Expected OP_HISTORY_RESPONSE, but got " + response.getOpcode());

    List<String> history = (List<String>) response.getList();
    StringBuilder sb = new StringBuilder();

    for(String s : history) {
        sb.append(s);
        sb.append("\n");
    }

    return sb.toString();
}
```

Definir administradores de sala. Que los administradores de sala puedan echar o ascender a otros usuarios

Lado de servidor

Esta mejora es quizás la más compleja ya que tiene asociados más de un comando: makeAdmin, removeAdmin, kick (echar a un usuario).

La implementación de administradores en la sala se hace simplemente manteniendo un listado de administradores en cada sala:

```
Set<User> admins;
```

Echar a un usuario supone banearlo permanentemente de la sala, el baneo se realiza por nick y ip. Esto es así por motivos relacionados con el testeo en local, pues el baneo por Ip también es igualmente sencillo y no haría falta modificar casi nada de código. Se mantiene un conjunto de usuarios baneados:

```
HashSet<User> blocked;
```

---

Se impide el registro de usuarios baneados en la sala:

```
public boolean registerUser(User u, Socket s) {

    if(blocked.contains(u)){
        return false;
    }

    boolean result = false;
    try {
        result = users.putIfAbsent(u, new DataOutputStream(s.getOutputStream())) == null;
    } catch (IOException e) {
        e.printStackTrace();
    }
    return result;
}
```

Después de este vistazo general vayamos por partes:

Definir administradores nuevos tras una petición asociada a ellos consiste en una verificación de permisos (para definir administradores nuevos has de ser administrador):

```
case NCMessage.OP_MAKE_ROOM_ADMIN:
    success = false;
    if(roomManager.isAdmin(user)) {
        roomMessage = (NCRoomMessage) message;
        success = roomManager.addAdmin(serverManager.getUser(roomMessage.getValue()));
    }

    if(success) {
        NCInternalMessage response = (NCInternalMessage) NCMessage.makeInternalMessage(NCMessage.OP_OK);
        rawResponse = response.toEncodedString();
        dos.writeUTF(rawResponse);
    } else {
        NCInternalMessage response = (NCInternalMessage) NCMessage.makeInternalMessage(NCMessage.OP_PERMISSION_INVALID);
        rawResponse = response.toEncodedString();
        dos.writeUTF(rawResponse);
    }
    break;
```



Si se es administrador **se agrega** a la lista de administradores, **verificando** que el usuario exista en la sala:

```
public boolean addAdmin(User u) {  
    return users.keySet().contains(u) && admins.add(u);  
    //if the user is not in the "users" returns false, else, returns whether admin was added or not (if the user is already an admin, it returns false)  
}
```

Más tarde se informa al cliente del resultado de la petición.

**También es posible agregar un administrador aleatoriamente:**

```
public void addRandomAdmin() {  
    if (users.size() > 0) {  
        List<User> valuesList = new ArrayList<User>(users.keySet());  
        int randomIndex = new Random().nextInt(valuesList.size());  
        User randomUser = valuesList.get(randomIndex);  
        addAdmin(randomUser);  
    }  
}
```

Esto sucederá automáticamente si la sala se queda sin administradores y no es una sala creada por el sistema, para más información de este proceso ver **mecanismo de gestión de salas : salida**.

Para quitar un administrador el proceso es exactamente igual, aunque en esta ocasión **se elimina de la lista de administradores**:

```
case NCMessage.OP_DELETE_ROOM_ADMIN:  
    success = false;  
    if(roomManager.isAdmin(user)) {  
        roomMessage = (NCRoomMessage) message;  
        success = roomManager.removeAdmin(serverManager.getUser(roomMessage.getValue()));  
    }  
  
    if(success) {  
        NCInternalMessage response = (NCInternalMessage) NCMessage.makeInternalMessage(NCMessage.OP_OK);  
        rawResponse = response.toEncodedString();  
        dos.writeUTF(rawResponse);  
    } else {  
        NCInternalMessage response = (NCInternalMessage) NCMessage.makeInternalMessage(NCMessage.OP_PERMISSION_INVALID);  
        rawResponse = response.toEncodedString();  
        dos.writeUTF(rawResponse);  
    }  
    break;
```

**Se abre así una vía para dejar sin administradores una sala creada por usuarios, el creador deberá eliminarse su permiso de administrador.**

Para el caso de baneo de usuario es necesario conocer como se encuentra implementado la clase User, en concreto bastaría con conocer que un User tiene en la implementación actual dos atributos:

```
public class User {  
  
    private final String name;  
    private final InetAddress address;  
    //private final Set<String> blocked;  
  
    public User(String name, InetAddress address) {  
        this.name = name;  
        this.address = address;  
        //this.blocked = new HashSet<>();  
    }  
}
```

Nombre y una representación de la dirección IP (InetAddress). Dos usuarios son iguales si ambos atributos coinciden:

@Override

```
public int hashCode() {  
    final int prime = 31;  
    int result = 1;  
    result = prime * result + ((address == null) ? 0 : address.hashCode());  
    result = prime * result + ((name == null) ? 0 : name.hashCode());  
    return result;  
}
```

@Override

```
public boolean equals(Object obj) {  
    if (this == obj)  
        return true;  
    if (obj == null)  
        return false;  
    if (getClass() != obj.getClass())  
        return false;  
    User other = (User) obj;  
    if (address == null) {  
        if (other.address != null)  
            return false;  
    } else if (!address.equals(other.address))  
        return false;  
    if (name == null) {  
        if (other.name != null)  
            return false;  
    } else if (!name.equals(other.name))  
        return false;  
    return true;  
}
```

A la hora de banear un usuario al igual que en los casos anteriores se recibe la petición y se verifican en el primer paso los permisos:

```
NCInternalMessage response = null;
roomMessage = (NCRoomMessage) message;
User toBanUser = serverManager.getUser(roomMessage.getValue());

if(!roomManager.isAdmin(user) || roomManager.isAdmin(toBanUser) ) {
    response = (NCInternalMessage) NCMessage.makeInternalMessage(NCMessage.OP_PERMISSION_INVALID);
    rawResponse = response.toEncodedString();
    dos.writeUTF(rawResponse);
}
```

Si no son suficientes se responde que no hay permisos, no son suficientes si: El usuario que solicita la expulsión de otro no es administrados, o bien, el usuario a eliminar es administrador.

Si se cumplen se verifica que el usuario a expulsar es usuario de la sala en cuestión, y se avisa a todos los usuarios del baneo, así como se lanza una advertencia especial hacia el usuario baneado que será interpretada por el cliente:

```
else if (roomManager.isUser(toBanUser)) {
    NCInternalMessage internalMessage = (NCInternalMessage) NCMessage.makeInternalMessage(NCMessage.OP_GOT_BANNED);
    rawResponse = internalMessage.toEncodedString();
    roomManager.sendPrivateMessage(toBanUser, rawResponse);

    roomManager.blockUser(toBanUser);
    response = (NCInternalMessage) NCMessage.makeInternalMessage(NCMessage.OP_OK);
    rawResponse = response.toEncodedString();
    dos.writeUTF(rawResponse);

    String exitWarning = toBanUser.getName() + " ha sido baneado del chat por el admin " + user.getName();
    roomMessage = (NCRoomMessage) NCMessage.makeRoomMessage(NCMessage.OP_SYSTEM_MESSAGE, exitWarning);
    roomManager.broadcastSystemMessage(roomMessage);
}
```

En caso de que el usuario a banear no fuera miembro de la sala se notifica al solicitante:

```
} else{
    response = (NCInternalMessage) NCMessage.makeInternalMessage(NCMessage.OP_NICK_INVALID);
    rawResponse = response.toEncodedString();
    dos.writeUTF(rawResponse);
}
```

### Lado del cliente

Desde el lado del cliente el procesamiento de las solicitudes es análogo a todos (makeAdmin, kick, removeAdmin). Se lee el comando y se procesa en su caso:

```
case NCCommands.COM_MAKE_ADMIN:
    processMakeAdminRequest();
    break;

case NCCommands.COM_REMOVE_ADMIN:
    processRemoveAdminRequest();
    break;

case NCCommands.COM_KICK:
    processKickRequest();
    break;
```

Para procesarlo se establece comunicación a través de NCconector como es habitual:

```
private void processKickRequest() {
    try{
        ncConnector.kickUser(otherParameter);
    }
    catch(IOException e){
        System.err.println("There was an error kicking the user");
        e.printStackTrace();
    }
}

private void processRemoveAdminRequest() {
    boolean result = false;
    try {
        result = ncConnector.removeAdmin(otherParameter);
    } catch (IOException e) {
        e.printStackTrace();
    }

    if(result){
        System.out.println("Petition fulfilled");
    } else{
        System.out.println("You are not an admin");
    }
}

private void processMakeAdminRequest() {
    boolean result = false;
    try {
        result = ncConnector.makeAdmin(otherParameter);
    } catch (IOException e) {
        e.printStackTrace();
    }

    if(result){
        System.out.println("Petition fulfilled");
    } else{
        System.out.println("You are not an admin");
    }
}
```

Ncconector lanza las peticiones oportunas de manera análoga:

```
public boolean kickUser(String nickname) throws IOException {
    NCRoomMessage message = (NCRoomMessage) NCMessage.makeRoomMessage(NCMessage.OP_KICK_USER, nickname);
    String rawMessage = message.toEncodedString();
    dos.writeUTF(rawMessage);
    NCInternalMessage response = (NCInternalMessage) NCMessage.readMessageFromSocket(dis);
    switch (response.getOpcode()) {
        case NCMessage.OP_OK:
            System.out.println("The user " + nickname + " has been kicked");
            return true;
        case NCMessage.OP_NICK_INVALID:
            System.out.println("The user " + nickname + " is not in the room");
            return false;
        case NCMessage.OP_PERMISSION_INVALID:
            System.out.println("Either you are not an admin or the user " + nickname + " is not admin");
            return false;
        default:
            throw new IllegalStateException("Unexpected opcode: " + response.getOpcode());
    }
}
```

```

public boolean removeAdmin(String nickname) throws IOException {
    NCRoomMessage message = (NCRoomMessage) NCMessage.makeRoomMessage(NCMessage.OP_DELETE_ROOM_ADMIN, nickname);
    String rawMessage = message.toEncodedString();
    dos.writeUTF(rawMessage);
    NCInternalMessage response = (NCInternalMessage) NCMessage.readMessageFromSocket(dis);
    switch (response.getOpcode()) {
        case NCMessage.OP_OK:
            return true;
        case NCMessage.OP_NICK_INVALID:
            System.out.println("The user " + nickname + " is not in the room");
            return false;
        case NCMessage.OP_PERMISSION_INVALID:
            System.out.println("Either you are not an admin or the user " + nickname + " is not admin");
            return false;
        default:
            throw new IllegalStateException("Unexpected opcode: " + response.getOpcode());
    }
}

public boolean makeAdmin(String nickname) throws IOException {
    NCRoomMessage message = (NCRoomMessage) NCMessage.makeRoomMessage(NCMessage.OP_MAKE_ROOM_ADMIN, nickname);
    String rawMessage = message.toEncodedString();
    dos.writeUTF(rawMessage);
    NCInternalMessage response = (NCInternalMessage) NCMessage.readMessageFromSocket(dis);
    switch (response.getOpcode()) {
        case NCMessage.OP_OK:
            return true;
        case NCMessage.OP_NICK_INVALID:
            System.out.println("The user " + nickname + " is not in the room");
            return false;
        case NCMessage.OP_PERMISSION_INVALID:
            System.out.println("Either you are not an admin or the user " + nickname + " is already an admin");
            return false;
        default:
            throw new IllegalStateException("Unexpected opcode: " + response.getOpcode());
    }
}

```

Por último, si se produce un baneo el usuario debe ser informado, y tiene la necesidad de informar al servidor. Si el cliente no informa al servidor:

[ver pagina siguiente]

```

public String getIncomingMessage() throws IOException {
    String text = null;;
    NCMessage message = NCMessage.readMessageFromSocket(dis);
    NCRoomMessage roomMessage = null;
    NCChatMessage chatMessage = null;

    switch(message.getOpcode()) {
        case NCMessage.OP_CHAT_MESSAGE:
            chatMessage = (NCChatMessage) message;
            text = chatMessage.toPrintableString();
            break;
        case NCMessage.OP_SYSTEM_MESSAGE:
            roomMessage = (NCRoomMessage) message;
            text = "System: " + roomMessage.getValue();
            break;
        case NCMessage.OP_GOT_BANNED:
            text = "You got banned";
            break;
        case NCMessage.OP_ENTERED_USER:
            roomMessage = (NCRoomMessage) message;
            text = "User " + roomMessage.getValue() + " entered the room";
            break;
        case NCMessage.OP_LEFT_USER:
            roomMessage = (NCRoomMessage) message;
            text = "User " + roomMessage.getValue() + " left the room";
            break;
        case NCMessage.OP_PRIVATE_CHAT_MESSAGE:
            chatMessage = (NCChatMessage) message;
            text = "Private message: " + chatMessage.toPrintableString();
            break;
        default:
            throw new IllegalStateException("Unexpected opcode: " + message.getOpcode());
    }

    return text;
}

```

El cliente detecta un mensaje de salida mediante la recepción de un mensaje desde el servidor.

Ncconector informa al controler de que ha sido baneado y debe salir:

```

//Método para procesar los mensajes recibidos del servidor mientras que el shell estaba esperando un comando de usuario
private void processIncomingMessage() {
    String text = "";
    try {
        text = ncConnector.getIncomingMessage();
    } catch (IOException e) {
        e.printStackTrace();
    }

    System.out.println(text);

    if(text.equals("You got banned")){
        clientStatus = REGISTERED;
        try {
            ncConnector.leaveRoom(room);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
}

```

### **Un método diferente**

Otra forma sería añadir un tercer atributo a la clase User, que contenga el un `DataOutputStream` hacia el servidor del usuario. De esta forma el propio servidor que banea podría informar al servidor (o hilo del servidor) que atiende al usuario baneado, sin la necesidad de establecer una relación de confianza ni verificación de baneo del cliente.

# Ejemplo de una conversación completa con Wirshark

No se muestran los mensajes auxiliares de TCP (handshake, confirmación, etc.)

No.	Time	Source	Destination	Protocol	Length	Info
201	2.917233	127.0.0.1	127.0.0.1	TCP	68	51430 → 6969 [SYN] Seq=0 Win=65535 Len=0 MSS=16344 WS=64 TSval=71371483 TSecr=0 SA
202	2.917328	127.0.0.1	127.0.0.1	TCP	68	6969 → 51430 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=16344 WS=64 TSval=71371483
203	2.917343	127.0.0.1	127.0.0.1	TCP	56	51430 → 6969 [ACK] Seq=1 Ack=1 Win=408256 Len=0 TSval=71371483 TSecr=71371483
204	2.917357	127.0.0.1	127.0.0.1	TCP	56	[TCP Window Update] 6969 → 51430 [ACK] Seq=1 Ack=1 Win=408256 Len=0 TSval=71371483
281	21.938527	127.0.0.1	127.0.0.1	TCP	90	51430 → 6969 [PSH, ACK] Seq=1 Ack=1 Win=408256 Len=34 TSval=71390454 TSecr=7137148
282	21.938554	127.0.0.1	127.0.0.1	TCP	56	6969 → 51430 [ACK] Seq=1 Ack=35 Win=408256 Len=0 TSval=71390454 TSecr=71390454
283	21.932185	127.0.0.1	127.0.0.1	TCP	75	6969 → 51430 [PSH, ACK] Seq=1 Ack=35 Win=408256 Len=19 TSval=71390455 TSecr=713904
284	21.932236	127.0.0.1	127.0.0.1	TCP	56	51430 → 6969 [ACK] Seq=35 Ack=20 Win=408256 Len=0 TSval=71390455 TSecr=71390455
285	35.153238	127.0.0.1	127.0.0.1	TCP	82	51430 → 6969 [PSH, ACK] Seq=35 Ack=20 Win=408256 Len=26 TSval=71403664 TSecr=71390
286	35.153270	127.0.0.1	127.0.0.1	TCP	56	6969 → 51430 [ACK] Seq=20 Ack=61 Win=408192 Len=0 TSval=71403664 TSecr=71403664

Frame 281: 90 bytes on wire (720 bits), 90 bytes captured (720 bits) on interface lo0, id 0

Null/Loopback

Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

Transmission Control Protocol, Src Port: 51430, Dst Port: 6969, Seq: 1, Ack: 1, Len: 34

Data (34 bytes)

0000 02 00 00 00 45 00 00 56 00 00 40 00 40 06 00 00 .....E..V...@...  
0010 7f 00 00 01 7f 00 00 01 c8 e6 1b 39 2a 80 2a 95 .....9\*...  
0020 5d d0 45 c4 80 18 18 eb fe 4a 00 00 01 01 00 0a ]..E.....J.....  
0030 04 41 54 f6 04 41 0a db 00 20 6f 70 65 72 61 74 .AT..A...operat  
0040 69 6f 6e 3a 4f 50 5f 4e 49 43 4b 0a 76 61 6c 75 ion:OP\_N\_ICK\_valu  
0050 65 3a 68 65 72 6e 61 6e 0a 0a e:herman..

wireshark\_lo0GII4L1.pcapng

Packets: 294 · Displayed: 12 (4.1%)

Profile: Default

## NICK

No.	Time	Source	Destination	Protocol	Length	Info
201	2.917233	127.0.0.1	127.0.0.1	TCP	68	51430 → 6969 [SYN] Seq=0 Win=65535 Len=0 MSS=16344 WS=64 TSval=71371483 TSecr=0 SA
202	2.917328	127.0.0.1	127.0.0.1	TCP	68	6969 → 51430 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=16344 WS=64 TSval=71371483
203	2.917343	127.0.0.1	127.0.0.1	TCP	56	51430 → 6969 [ACK] Seq=1 Ack=1 Win=408256 Len=0 TSval=71371483 TSecr=71371483
204	2.917357	127.0.0.1	127.0.0.1	TCP	56	[TCP Window Update] 6969 → 51430 [ACK] Seq=1 Ack=1 Win=408256 Len=0 TSval=71371483
281	21.938527	127.0.0.1	127.0.0.1	TCP	90	51430 → 6969 [PSH, ACK] Seq=1 Ack=1 Win=408256 Len=34 TSval=71390454 TSecr=7137148
282	21.938554	127.0.0.1	127.0.0.1	TCP	56	6969 → 51430 [ACK] Seq=1 Ack=35 Win=408256 Len=0 TSval=71390454 TSecr=71390454
283	21.932185	127.0.0.1	127.0.0.1	TCP	75	6969 → 51430 [PSH, ACK] Seq=1 Ack=35 Win=408256 Len=19 TSval=71390455 TSecr=713904
284	21.932236	127.0.0.1	127.0.0.1	TCP	56	51430 → 6969 [ACK] Seq=35 Ack=20 Win=408256 Len=0 TSval=71390455 TSecr=71390455
285	35.153238	127.0.0.1	127.0.0.1	TCP	82	51430 → 6969 [PSH, ACK] Seq=35 Ack=20 Win=408256 Len=26 TSval=71403664 TSecr=71390
286	35.153270	127.0.0.1	127.0.0.1	TCP	56	6969 → 51430 [ACK] Seq=20 Ack=61 Win=408192 Len=0 TSval=71403664 TSecr=71403664

Frame 283: 75 bytes on wire (600 bits), 75 bytes captured (600 bits) on interface lo0, id 0

Null/Loopback

Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

Transmission Control Protocol, Src Port: 6969, Dst Port: 51430, Seq: 1, Ack: 35, Len: 19

Data (19 bytes)

0000 02 00 00 00 45 00 00 47 00 00 40 00 40 06 00 00 .....E..G...@...  
0010 7f 00 00 01 7f 00 00 01 1b 39 c8 e6 5d d0 45 c4 .....9...].E..  
0020 2a 80 2a b7 80 18 18 eb fe 3b 00 00 01 01 00 0a \*...;.....  
0030 04 41 54 f7 04 41 54 f6 00 11 6f 70 65 72 61 74 .AT..AT...operat  
0040 69 6f 6e 3a 4f 50 5f 4f 4b 0a 0a ion:OP\_O\_K...

wireshark\_lo0GII4L1.pcapng

Packets: 296 · Displayed: 12 (4.1%)

Profile: Default

## NICK\_OK



No.	Time	Source	Destination	Protocol	Length	Info
201	2.917233	127.0.0.1	127.0.0.1	TCP	68	51430 → 6969 [SYN] Seq=0 Win=65535 Len=0 MSS=16344 WS=64 TSval=71371483 TSecr=0 SA
202	2.917328	127.0.0.1	127.0.0.1	TCP	68	6969 → 51430 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=16344 WS=64 TSval=71371483
203	2.917343	127.0.0.1	127.0.0.1	TCP	56	51430 → 6969 [ACK] Seq=1 Ack=1 Win=408256 Len=0 TSval=71371483 TSecr=71371483
204	2.917357	127.0.0.1	127.0.0.1	TCP	56	[TCP Window Update] 6969 → 51430 [ACK] Seq=1 Ack=1 Win=408256 Len=0 TSval=71371483
205	2.917357	127.0.0.1	127.0.0.1	TCP	90	51430 → 6969 [PSH, ACK] Seq=1 Ack=1 Win=408256 Len=34 TSval=71390454 TSecr=7137148
206	2.917357	127.0.0.1	127.0.0.1	TCP	56	6969 → 51430 [ACK] Seq=1 Ack=35 Win=408256 Len=0 TSval=71390454 TSecr=71390454
207	2.917357	127.0.0.1	127.0.0.1	TCP	75	6969 → 51430 [PSH, ACK] Seq=1 Ack=35 Win=408256 Len=19 TSval=71390455 TSecr=713904
208	2.917357	127.0.0.1	127.0.0.1	TCP	56	51430 → 6969 [ACK] Seq=35 Ack=20 Win=408256 Len=0 TSval=71390455 TSecr=71390455
209	2.917357	127.0.0.1	127.0.0.1	TCP	82	51430 → 6969 [PSH, ACK] Seq=35 Ack=20 Win=408256 Len=26 TSval=71403664 TSecr=71390
210	2.917357	127.0.0.1	127.0.0.1	TCP	56	6969 → 51430 [ACK] Seq=20 Ack=61 Win=408192 Len=0 TSval=71403664 TSecr=71403664

▶ Frame 285: 82 bytes on wire (656 bits), 82 bytes captured (656 bits) on interface lo0, id 0  
 ▶ Null/Loopback  
 ▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1  
 ▶ Transmission Control Protocol, Src Port: 51430, Dst Port: 6969, Seq: 35, Ack: 20, Len: 26  
 ▶ Data (26 bytes)

```

0000  02 00 00 00 45 00 00 4e 00 00 40 00 06 00 00  ....E..N...@...
0010  7f 00 00 01 7f 00 00 01 c8 e6 1b 39 2a 80 2a b7  .....9...*
0020  5d d0 45 d7 80 18 18 eb fe 42 00 00 01 01 08 0a  ].E.....B.....
0030  04 41 88 90 04 41 54 f7 00 18 6f 70 65 72 61 74  .A...AT...operat
0040  69 6f 6e 3a 4f 50 5f 47 45 54 5f 52 4f 4f 4d 53  ion:OP_G ET_ROOMS
0050  0a 0a  ..
  
```

## GET ROOMS

No.	Time	Source	Destination	Protocol	Length	Info
202	2.917328	127.0.0.1	127.0.0.1	TCP	68	6969 → 51430 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=16344 WS=64 TSval=71371483
203	2.917343	127.0.0.1	127.0.0.1	TCP	56	51430 → 6969 [ACK] Seq=1 Ack=1 Win=408256 Len=0 TSval=71371483 TSecr=71371483
204	2.917357	127.0.0.1	127.0.0.1	TCP	56	[TCP Window Update] 6969 → 51430 [ACK] Seq=1 Ack=1 Win=408256 Len=0 TSval=71371483
205	2.917357	127.0.0.1	127.0.0.1	TCP	90	51430 → 6969 [PSH, ACK] Seq=1 Ack=1 Win=408256 Len=34 TSval=71390454 TSecr=7137148
206	2.917357	127.0.0.1	127.0.0.1	TCP	56	6969 → 51430 [ACK] Seq=1 Ack=35 Win=408256 Len=0 TSval=71390454 TSecr=71390454
207	2.917357	127.0.0.1	127.0.0.1	TCP	75	6969 → 51430 [PSH, ACK] Seq=1 Ack=35 Win=408256 Len=19 TSval=71390455 TSecr=713904
208	2.917357	127.0.0.1	127.0.0.1	TCP	56	51430 → 6969 [ACK] Seq=35 Ack=20 Win=408256 Len=0 TSval=71390455 TSecr=71390455
209	2.917357	127.0.0.1	127.0.0.1	TCP	82	51430 → 6969 [PSH, ACK] Seq=35 Ack=20 Win=408256 Len=26 TSval=71403664 TSecr=71390
210	2.917357	127.0.0.1	127.0.0.1	TCP	56	6969 → 51430 [ACK] Seq=20 Ack=61 Win=408192 Len=0 TSval=71403664 TSecr=71403664
211	3.173914	127.0.0.1	127.0.0.1	TCP	186	6969 → 51430 [PSH, ACK] Seq=20 Ack=61 Win=408192 Len=130 TSval=71403683 TSecr=7140

▶ Frame 287: 186 bytes on wire (1488 bits), 186 bytes captured (1488 bits) on interface lo0, id 0  
 ▶ Null/Loopback  
 ▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1  
 ▶ Transmission Control Protocol, Src Port: 6969, Dst Port: 51430, Seq: 20, Ack: 61, Len: 130  
 ▶ Data (130 bytes)

```

0000  02 00 00 00 45 00 00 b6 00 00 40 00 06 00 00  ....E...@...
0010  7f 00 00 01 7f 00 00 01 1b 39 c8 e6 5d d0 45 d7  .....9...].E
0020  2a 80 2a d1 80 18 18 ea fe aa 00 00 01 01 08 0a  *.....
0030  04 41 88 a3 04 41 88 90 00 00 6f 70 65 72 61 74  .A...A...operat
0040  69 6f 6e 3a 4f 50 5f 52 4f 4f 4d 5f 4c 49 53 54  ion:OP_ROOM_LIST
0050  0a 65 6c 65 6d 65 6e 74 73 3a 45 73 70 61 c3 b1  .element s:Espa
0060  61 40 40 40 31 36 35 32 30 32 35 36 32 35 33 31  a@@@1652 02562531
0070  37 2c 41 6c 65 6d 61 6c 69 61 40 40 40 31 36 35  7,Alenan 19@@@165
0080  32 30 32 35 36 32 35 33 31 37 2c 49 74 61 6c 69  20256253 17,ttat1
0090  61 40 40 40 31 36 35 32 30 32 35 36 32 35 33 31  a@@@1652 02562531
00a0  37 2c 47 6c 6f 62 61 6c 40 40 40 31 36 35 32 30  7,Global @@@16520
00b0  32 35 36 32 35 33 31 37 0a 0a 25625317 ..
  
```

## Get romos response (roomList)

No.	Time	Source	Destination	Protocol	Length	Info
285	35.153238	127.0.0.1	127.0.0.1	TCP	82	51430 → 6969 [PSH, ACK] Seq=35 Ack=20 Win=408256 Len=26 TSval=71403664 TSecr=71390
286	35.153270	127.0.0.1	127.0.0.1	TCP	56	6969 → 51430 [ACK] Seq=20 Ack=61 Win=408192 Len=0 TSval=71403664 TSecr=71403664
287	35.173914	127.0.0.1	127.0.0.1	TCP	186	6969 → 51430 [PSH, ACK] Seq=20 Ack=61 Win=408192 Len=130 TSval=71403683 TSecr=7140
288	35.173976	127.0.0.1	127.0.0.1	TCP	56	51430 → 6969 [ACK] Seq=61 Ack=150 Win=408128 Len=0 TSval=71403683 TSecr=71403683
306	500.951069	127.0.0.1	127.0.0.1	TCP	102	51430 → 6969 [PSH, ACK] Seq=61 Ack=150 Win=408128 Len=46 TSval=71868309 TSecr=7140
307	500.951122	127.0.0.1	127.0.0.1	TCP	56	6969 → 51430 [ACK] Seq=150 Ack=107 Win=408192 Len=0 TSval=71868309 TSecr=71868309
308	500.952624	127.0.0.1	127.0.0.1	TCP	75	6969 → 51430 [PSH, ACK] Seq=150 Ack=107 Win=408192 Len=19 TSval=71868310 TSecr=718
309	500.952660	127.0.0.1	127.0.0.1	TCP	56	51430 → 6969 [ACK] Seq=107 Ack=169 Win=408128 Len=0 TSval=71868310 TSecr=71868310
310	500.953301	127.0.0.1	127.0.0.1	TCP	98	6969 → 51430 [PSH, ACK] Seq=169 Ack=107 Win=408192 Len=42 TSval=71868311 TSecr=718
311	500.953324	127.0.0.1	127.0.0.1	TCP	56	51430 → 6969 [ACK] Seq=107 Ack=211 Win=408064 Len=0 TSval=71868311 TSecr=71868311

▶ Frame 386: 102 bytes on wire (816 bits), 102 bytes captured (816 bits) on interface lo0, id 0  
 ▶ Null/Loopback  
 ▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1  
 ▶ Transmission Control Protocol, Src Port: 51430, Dst Port: 6969, Seq: 61, Ack: 150, Len: 46  
 ▶ Data (46 bytes)

```

0000  02 00 00 00 45 00 00 62 00 00 40 00 40 06 00 00  ....E..b...@...
0010  7f 00 00 01 7f 00 00 01 c8 e6 1b 39 2a 80 2a d1  ....9*...
0020  5d d0 46 59 80 18 18 e9 fe 56 00 00 01 01 08 0a  ].FY.....V.....
0030  04 48 9f 95 04 41 88 a3 00 2c 6f 70 65 72 61 74  .H..A...operat
0040  69 6f 6e 3a 4f 50 5f 43 52 45 41 54 45 5f 52 4f  ionOP_C REATE RO
0050  4f 4d 0a 76 61 6c 75 65 3a 65 78 61 6d 70 6c 65  OM value :example
0060  52 6f 6f 6d 0a 0a                                Room..
  
```

wireshark\_lo0GII4LI.pcapng      Packets: 313 - Displayed: 18 (5.8%)      Profile: Default

Create room

No.	Time	Source	Destination	Protocol	Length	Info
285	35.153238	127.0.0.1	127.0.0.1	TCP	82	51430 → 6969 [PSH, ACK] Seq=35 Ack=20 Win=408256 Len=26 TSval=71403664 TSecr=71390
286	35.153270	127.0.0.1	127.0.0.1	TCP	56	6969 → 51430 [ACK] Seq=20 Ack=61 Win=408192 Len=0 TSval=71403664 TSecr=71403664
287	35.173914	127.0.0.1	127.0.0.1	TCP	186	6969 → 51430 [PSH, ACK] Seq=20 Ack=61 Win=408192 Len=130 TSval=71403683 TSecr=7140
288	35.173976	127.0.0.1	127.0.0.1	TCP	56	51430 → 6969 [ACK] Seq=61 Ack=150 Win=408128 Len=0 TSval=71403683 TSecr=71403683
306	500.951069	127.0.0.1	127.0.0.1	TCP	102	51430 → 6969 [PSH, ACK] Seq=61 Ack=150 Win=408128 Len=46 TSval=71868309 TSecr=7140
307	500.951122	127.0.0.1	127.0.0.1	TCP	56	6969 → 51430 [ACK] Seq=150 Ack=107 Win=408192 Len=0 TSval=71868309 TSecr=71868309
308	500.952624	127.0.0.1	127.0.0.1	TCP	75	6969 → 51430 [PSH, ACK] Seq=150 Ack=107 Win=408192 Len=19 TSval=71868310 TSecr=718
309	500.952660	127.0.0.1	127.0.0.1	TCP	56	51430 → 6969 [ACK] Seq=107 Ack=169 Win=408128 Len=0 TSval=71868310 TSecr=71868310
310	500.953301	127.0.0.1	127.0.0.1	TCP	98	6969 → 51430 [PSH, ACK] Seq=169 Ack=107 Win=408192 Len=42 TSval=71868311 TSecr=718
311	500.953324	127.0.0.1	127.0.0.1	TCP	56	51430 → 6969 [ACK] Seq=107 Ack=211 Win=408064 Len=0 TSval=71868311 TSecr=71868311

▶ Frame 388: 75 bytes on wire (600 bits), 75 bytes captured (600 bits) on interface lo0, id 0  
 ▶ Null/Loopback  
 ▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1  
 ▶ Transmission Control Protocol, Src Port: 6969, Dst Port: 51430, Seq: 150, Ack: 107, Len: 19  
 ▶ Data (19 bytes)

```

0000  02 00 00 00 45 00 00 47 00 00 40 00 40 06 00 00  ....E..G...@...
0010  7f 00 00 01 7f 00 00 01 1b 39 c8 e6 5d d0 46 59  ....9...].FY
0020  2a 80 2a ff 00 18 18 ea fe 3b 00 00 01 01 08 0a  *.~.....;.....
0030  04 48 9f 96 04 48 9f 95 00 11 6f 70 65 72 61 74  .H..H...operat
0040  69 6f 6e 3a 4f 50 5f 4f 4b 0a 0a                  ion:OP_0 K..
  
```

wireshark\_lo0GII4LI.pcapng      Packets: 313 - Displayed: 18 (5.8%)      Profile: Default

Ok (response) [create\_room\_ok response]

No.	Time	Source	Destination	Protocol	Length	Info
285	35.153238	127.0.0.1	127.0.0.1	TCP	82	51430 → 6969 [PSH, ACK] Seq=35 Ack=20 Win=408256 Len=26 TSval=71403664 TSecr=71390
286	35.153270	127.0.0.1	127.0.0.1	TCP	56	6969 → 51430 [ACK] Seq=20 Ack=61 Win=408192 Len=0 TSval=71403664 TSecr=71403664
287	35.173914	127.0.0.1	127.0.0.1	TCP	186	6969 → 51430 [PSH, ACK] Seq=20 Ack=61 Win=408192 Len=130 TSval=71403683 TSecr=7140
288	35.173976	127.0.0.1	127.0.0.1	TCP	56	51430 → 6969 [ACK] Seq=61 Ack=150 Win=408128 Len=0 TSval=71403683 TSecr=71403683
306	500.951069	127.0.0.1	127.0.0.1	TCP	102	51430 → 6969 [PSH, ACK] Seq=61 Ack=150 Win=408128 Len=46 TSval=71868309 TSecr=7140
307	500.951122	127.0.0.1	127.0.0.1	TCP	56	6969 → 51430 [ACK] Seq=150 Ack=107 Win=408192 Len=0 TSval=71868309 TSecr=71868309
308	500.952624	127.0.0.1	127.0.0.1	TCP	75	6969 → 51430 [PSH, ACK] Seq=150 Ack=107 Win=408192 Len=19 TSval=71868310 TSecr=718
309	500.952660	127.0.0.1	127.0.0.1	TCP	56	51430 → 6969 [ACK] Seq=107 Ack=169 Win=408128 Len=0 TSval=71868310 TSecr=71868310
310	500.953301	127.0.0.1	127.0.0.1	TCP	98	6969 → 51430 [PSH, ACK] Seq=169 Ack=107 Win=408192 Len=42 TSval=71868311 TSecr=718
311	500.953324	127.0.0.1	127.0.0.1	TCP	56	51430 → 6969 [ACK] Seq=107 Ack=211 Win=408064 Len=0 TSval=71868311 TSecr=71868311

▶ Frame 310: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface lo0, id 0  
 ▶ Null/Loopback  
 ▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1  
 ▶ Transmission Control Protocol, Src Port: 6969, Dst Port: 51430, Seq: 169, Ack: 107, Len: 42  
 ▶ Data (42 bytes)

```

0000  02 00 00 00 45 00 00 5e 00 00 40 00 06 00 00  ....E...@...
0010  7f 00 00 01 7f 00 00 01 1b 39 c8 e6 5d d0 46 6c  .....9...].F.
0020  2a 80 2a ff 80 18 18 ea fe 52 00 00 01 01 08 0a  *...R.....
0030  04 48 9f 97 04 48 9f 96 00 28 6f 70 65 72 61 74  .H...H...operat
0040  69 6f 6e 3a 4f 50 5f 45 4e 54 45 52 45 44 5f 55  ion:OP_E NTERED_U
0050  53 45 52 0a 76 61 6c 75 65 3a 68 65 72 6e 61 6e  SER valu e:hernan
0060  0a 0a
  
```

wireshark\_lo0GII4L1.pcapng      Packets: 319 · Displayed: 18 (5.6%)      Profile: Default

7 server: user\_entered\_by\_creating\_room

No.	Time	Source	Destination	Protocol	Length	Info
642	1008.270760	127.0.0.1	127.0.0.1	TCP	95	6969 → 51561 [PSH, ACK] Seq=39 Ack=77 Win=408192 Len=39 TSval=72374619 TSecr=72374
643	1008.270780	127.0.0.1	127.0.0.1	TCP	56	51561 → 6969 [ACK] Seq=77 Ack=78 Win=408192 Len=0 TSval=72374619 TSecr=72374619
644	1008.270863	127.0.0.1	127.0.0.1	TCP	95	6969 → 51430 [PSH, ACK] Seq=211 Ack=107 Win=408192 Len=39 TSval=72374620 TSecr=718
645	1008.270882	127.0.0.1	127.0.0.1	TCP	56	51430 → 6969 [ACK] Seq=107 Ack=250 Win=408000 Len=0 TSval=72374620 TSecr=72374620
646	1040.152554	127.0.0.1	127.0.0.1	TCP	97	51430 → 6969 [PSH, ACK] Seq=107 Ack=250 Win=408000 Len=41 TSval=72406472 TSecr=723
647	1040.152638	127.0.0.1	127.0.0.1	TCP	56	6969 → 51430 [ACK] Seq=250 Ack=148 Win=408128 Len=0 TSval=72406472 TSecr=72406472
648	1040.154144	127.0.0.1	127.0.0.1	TCP	111	6969 → 51561 [PSH, ACK] Seq=78 Ack=77 Win=408192 Len=55 TSval=72406473 TSecr=72374
649	1040.154184	127.0.0.1	127.0.0.1	TCP	56	51561 → 6969 [ACK] Seq=77 Ack=133 Win=408128 Len=0 TSval=72406473 TSecr=72406473
650	1040.154297	127.0.0.1	127.0.0.1	TCP	111	6969 → 51430 [PSH, ACK] Seq=250 Ack=148 Win=408128 Len=55 TSval=72406473 TSecr=724
651	1040.154319	127.0.0.1	127.0.0.1	TCP	56	51430 → 6969 [ACK] Seq=148 Ack=385 Win=407936 Len=0 TSval=72406473 TSecr=72406473

▶ Frame 646: 97 bytes on wire (776 bits), 97 bytes captured (776 bits) on interface lo0, id 0  
 ▶ Null/Loopback  
 ▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1  
 ▶ Transmission Control Protocol, Src Port: 51430, Dst Port: 6969, Seq: 107, Ack: 250, Len: 41  
 ▶ Data (41 bytes)

```

0000  02 00 00 00 45 00 00 5d 00 00 40 00 06 00 00  ....E...@...
0010  7f 00 00 01 7f 00 00 01 c8 e6 1b 39 2a 80 2a ff  .....9*...
0020  5d 00 46 bd 00 18 18 e7 fe 51 00 00 01 01 08 0a  ].F.....Q....
0030  04 50 d5 c8 04 50 59 5c 00 27 6f 70 65 72 61 74  .P...PY...operat
0040  69 6f 6e 3a 4f 50 5f 53 45 4e 44 5f 4d 45 53 53  ion:OP_S END_MESS
0050  41 47 45 0a 76 61 6c 75 65 3a 68 6f 6c 61 20 0a  AGE valu e:hola
0060  0a
  
```

wireshark\_lo0GII4L1.pcapng      Packets: 654 · Displayed: 40 (6.1%)      Profile: Default

8 user: send hola



No.	Time	Source	Destination	Protocol	Length	Info
642	1008.270760	127.0.0.1	127.0.0.1	TCP	95	6969 → 51561 [PSH, ACK] Seq=39 Ack=77 Win=408192 Len=39 TSval=72374619 TSecr=72374619
643	1008.270780	127.0.0.1	127.0.0.1	TCP	56	51561 → 6969 [ACK] Seq=77 Ack=78 Win=408192 Len=0 TSval=72374619 TSecr=72374619
644	1008.270863	127.0.0.1	127.0.0.1	TCP	95	6969 → 51430 [PSH, ACK] Seq=211 Ack=107 Win=408192 Len=39 TSval=72374620 TSecr=718
645	1008.270882	127.0.0.1	127.0.0.1	TCP	56	51430 → 6969 [ACK] Seq=107 Ack=250 Win=408000 Len=0 TSval=72374620 TSecr=72374620
646	1040.152554	127.0.0.1	127.0.0.1	TCP	97	51430 → 6969 [PSH, ACK] Seq=107 Ack=250 Win=408000 Len=41 TSval=72406472 TSecr=723
647	1040.152638	127.0.0.1	127.0.0.1	TCP	56	6969 → 51430 [ACK] Seq=250 Ack=148 Win=408128 Len=0 TSval=72406472 TSecr=72406472
648	1040.154144	127.0.0.1	127.0.0.1	TCP	111	6969 → 51561 [PSH, ACK] Seq=78 Ack=77 Win=408192 Len=55 TSval=72406473 TSecr=72374
649	1040.154184	127.0.0.1	127.0.0.1	TCP	56	51561 → 6969 [ACK] Seq=77 Ack=133 Win=408128 Len=0 TSval=72406473 TSecr=72406473
650	1040.154297	127.0.0.1	127.0.0.1	TCP	111	6969 → 51430 [PSH, ACK] Seq=250 Ack=148 Win=408128 Len=55 TSval=72406473 TSecr=724
651	1040.154319	127.0.0.1	127.0.0.1	TCP	56	51430 → 6969 [ACK] Seq=148 Ack=305 Win=407936 Len=0 TSval=72406473 TSecr=72406473

▶ Frame 648: 111 bytes on wire (888 bits), 111 bytes captured (888 bits) on interface lo0, id 0  
 ▶ Null/Loopback  
 ▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1  
 ▶ Transmission Control Protocol, Src Port: 6969, Dst Port: 51561, Seq: 78, Ack: 77, Len: 55  
 ▶ Data (55 bytes)

```

0000 02 00 00 00 45 00 00 6b 00 00 40 00 40 06 00 00  ....E..k...@...
0010 7f 00 00 01 7f 00 00 01 1b 39 c9 69 de 63 b8 34  ....9..i.c.4
0020 f1 db f3 7d 80 18 10 ea fe 5f 00 00 01 01 00 0a  ....-}.....-....
0030 04 50 d5 c9 04 50 59 5b 00 35 6f 70 65 72 61 74  .P...PY[  Soperat
0040 69 6f 6e 3a 4f 50 5f 43 48 41 54 5f 4d 45 53 53  ion:OP_C HAT_MESS
0050 41 47 45 0a 76 61 6c 75 65 3a 68 65 72 6e 61 6e  AGE.valu e:hernan
0060 0a 6d 65 73 73 61 67 65 3a 68 6f 6c 61 0a 0a    .message :hola..
  
```

wireshark\_lo0GII4L1.pcapng      Packets: 656 · Displayed: 40 (6.1%)      Profile: Default

9 server broadcasted message, ejemplo de que le ha hecho broadcast a un usuario de la sala

No.	Time	Source	Destination	Protocol	Length	Info
669	1203.073730	127.0.0.1	127.0.0.1	TCP	98	51430 → 6969 [PSH, ACK] Seq=148 Ack=305 Win=407936 Len=42 TSval=72568965 TSecr=724
670	1203.073814	127.0.0.1	127.0.0.1	TCP	56	6969 → 51430 [ACK] Seq=305 Ack=190 Win=408064 Len=0 TSval=72568965 TSecr=72568965
671	1203.074178	127.0.0.1	127.0.0.1	TCP	112	6969 → 51561 [PSH, ACK] Seq=133 Ack=77 Win=408192 Len=56 TSval=72568965 TSecr=7240
672	1203.074199	127.0.0.1	127.0.0.1	TCP	56	51561 → 6969 [ACK] Seq=77 Ack=189 Win=408064 Len=0 TSval=72568965 TSecr=72568965
673	1203.074332	127.0.0.1	127.0.0.1	TCP	112	6969 → 51430 [PSH, ACK] Seq=305 Ack=190 Win=408064 Len=56 TSval=72568965 TSecr=725
674	1203.074351	127.0.0.1	127.0.0.1	TCP	56	51430 → 6969 [ACK] Seq=190 Ack=361 Win=407936 Len=0 TSval=72568965 TSecr=72568965
675	1216.948703	127.0.0.1	127.0.0.1	TCP	82	51430 → 6969 [PSH, ACK] Seq=190 Ack=361 Win=407936 Len=26 TSval=72582832 TSecr=725
676	1216.948780	127.0.0.1	127.0.0.1	TCP	56	6969 → 51430 [ACK] Seq=361 Ack=216 Win=408064 Len=0 TSval=72582832 TSecr=72582832
677	1216.949473	127.0.0.1	127.0.0.1	TCP	97	6969 → 51561 [PSH, ACK] Seq=189 Ack=77 Win=408192 Len=41 TSval=72582832 TSecr=7256
678	1216.949505	127.0.0.1	127.0.0.1	TCP	56	51561 → 6969 [ACK] Seq=77 Ack=230 Win=408064 Len=0 TSval=72582832 TSecr=72582832

▶ Frame 677: 97 bytes on wire (776 bits), 97 bytes captured (776 bits) on interface lo0, id 0  
 ▶ Null/Loopback  
 ▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1  
 ▶ Transmission Control Protocol, Src Port: 6969, Dst Port: 51561, Seq: 189, Ack: 77, Len: 41  
 ▶ Data (41 bytes)

```

0000 02 00 00 00 45 00 00 5d 00 00 40 00 40 06 00 00  ....E..]...@...
0010 7f 00 00 01 7f 00 00 01 1b 39 c9 69 de 63 b8 a3  ....9..i.c...
0020 f1 db f3 7d 80 18 10 ea fe 51 00 00 01 01 00 0a  ....-}.....0.....
0030 04 53 86 b0 04 53 50 85 00 27 6f 70 65 72 61 74  .S...SP...operat
0040 69 6f 6e 3a 4f 50 5f 45 58 49 54 45 44 5f 55 53  ion:OP_E XITED_US
0050 45 52 0a 76 61 6c 75 65 3a 68 65 72 6e 61 6e 0a  ER.value :hernan
0060 0a                                .
  
```

wireshark\_lo0GII4L1.pcapng      Packets: 678 · Displayed: 50 (7.4%)      Profile: Default

10. Un usuario se desconecta -> broadcast de aviso

No.	Time	Source	Destination	Protocol	Length	Info
677	1216.949473	127.0.0.1	127.0.0.1	TCP	97	6969 → 51561 [PSH, ACK] Seq=189 Ack=77 Win=408192 Len=41 TSval=72582832 TSecr=7256
678	1216.949505	127.0.0.1	127.0.0.1	TCP	56	51561 → 6969 [ACK] Seq=77 Ack=230 Win=408064 Len=0 TSval=72582832 TSecr=72582832
681	1285.994307	127.0.0.1	127.0.0.1	TCP	101	51430 → 6969 [PSH, ACK] Seq=216 Ack=361 Win=407936 Len=45 TSval=72651608 TSecr=725
682	1285.994345	127.0.0.1	127.0.0.1	TCP	56	6969 → 51430 [ACK] Seq=361 Ack=261 Win=408000 Len=0 TSval=72651608 TSecr=72651608
683	1285.995463	127.0.0.1	127.0.0.1	TCP	75	6969 → 51430 [PSH, ACK] Seq=361 Ack=261 Win=408000 Len=19 TSval=72651609 TSecr=726
684	1285.995499	127.0.0.1	127.0.0.1	TCP	56	51430 → 6969 [ACK] Seq=261 Ack=380 Win=407872 Len=0 TSval=72651609 TSecr=72651609
685	1285.995610	127.0.0.1	127.0.0.1	TCP	98	6969 → 51561 [PSH, ACK] Seq=230 Ack=77 Win=408192 Len=42 TSval=72651609 TSecr=7258
686	1285.995629	127.0.0.1	127.0.0.1	TCP	56	51561 → 6969 [ACK] Seq=77 Ack=272 Win=408000 Len=0 TSval=72651609 TSecr=72651609
687	1285.995694	127.0.0.1	127.0.0.1	TCP	98	6969 → 51430 [PSH, ACK] Seq=380 Ack=261 Win=408000 Len=42 TSval=72651609 TSecr=726
688	1285.995714	127.0.0.1	127.0.0.1	TCP	56	51430 → 6969 [ACK] Seq=261 Ack=422 Win=407872 Len=0 TSval=72651609 TSecr=72651609

▶ Frame 681: 101 bytes on wire (808 bits), 101 bytes captured (808 bits) on interface lo0, id 0  
 ▶ Null/Loopback  
 ▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1  
 ▶ Transmission Control Protocol, Src Port: 51430, Dst Port: 6969, Seq: 216, Ack: 361, Len: 45  
 ▶ Data (45 bytes)

```

0000 02 00 00 00 45 00 00 61 00 00 40 00 00 06 00 00  ....E..a...@...
0010 7f 00 00 01 7f 00 00 01 c8 e6 1b 39 2a 80 2b 6c  ....9...+l
0020 5d d0 47 2c 80 18 18 e6 fe 55 00 00 01 01 08 0a  ].G.....U.....
0030 04 54 93 58 04 53 86 b0 00 2b 6f 70 65 72 61 74  .T.X.S...operat
0040 69 6f 6e 3a 4f 50 5f 45 4e 54 45 52 5f 52 4f 4f  ion:OP_E NTER_R00
0050 4d 0a 76 61 6c 75 65 3a 65 78 61 6d 70 6c 65 52  M value: examplcR
0060 6f 6f 6d 0a 0a  oom..
  
```

wireshark -i 0GigabitEthernet0

Packets: 688 · Displayed: 58 (8.4%)

Profile: Default

## 11 Usuario solicita entrar a exampleRoom (Otro usuario)

No.	Time	Source	Destination	Protocol	Length	Info
683	1285.995463	127.0.0.1	127.0.0.1	TCP	75	6969 → 51430 [PSH, ACK] Seq=361 Ack=261 Win=408000 Len=19 TSval=72651609 TSecr=726
684	1285.995499	127.0.0.1	127.0.0.1	TCP	56	51430 → 6969 [ACK] Seq=261 Ack=380 Win=407872 Len=0 TSval=72651609 TSecr=72651609
685	1285.995610	127.0.0.1	127.0.0.1	TCP	98	6969 → 51561 [PSH, ACK] Seq=230 Ack=77 Win=408192 Len=42 TSval=72651609 TSecr=7258
686	1285.995629	127.0.0.1	127.0.0.1	TCP	56	51561 → 6969 [ACK] Seq=77 Ack=272 Win=408000 Len=0 TSval=72651609 TSecr=72651609
687	1285.995694	127.0.0.1	127.0.0.1	TCP	98	6969 → 51430 [PSH, ACK] Seq=380 Ack=261 Win=408000 Len=42 TSval=72651609 TSecr=726
688	1285.995714	127.0.0.1	127.0.0.1	TCP	56	51430 → 6969 [ACK] Seq=261 Ack=422 Win=407872 Len=0 TSval=72651609 TSecr=72651609
693	1385.912027	127.0.0.1	127.0.0.1	TCP	102	51561 → 6969 [PSH, ACK] Seq=77 Ack=272 Win=408000 Len=46 TSval=72751160 TSecr=7265
694	1385.912066	127.0.0.1	127.0.0.1	TCP	56	6969 → 51561 [ACK] Seq=272 Ack=123 Win=408128 Len=0 TSval=72751160 TSecr=72751160
695	1385.912524	127.0.0.1	127.0.0.1	TCP	75	6969 → 51561 [PSH, ACK] Seq=272 Ack=123 Win=408128 Len=19 TSval=72751160 TSecr=727
696	1385.912579	127.0.0.1	127.0.0.1	TCP	56	51561 → 6969 [ACK] Seq=123 Ack=291 Win=408000 Len=0 TSval=72751160 TSecr=72751160

▶ Frame 693: 102 bytes on wire (816 bits), 102 bytes captured (816 bits) on interface lo0, id 0  
 ▶ Null/Loopback  
 ▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1  
 ▶ Transmission Control Protocol, Src Port: 51561, Dst Port: 6969, Seq: 77, Ack: 272, Len: 46  
 ▶ Data (46 bytes)

```

0000 02 00 00 00 45 00 00 62 00 00 40 00 00 06 00 00  ....E..b...@...
0010 7f 00 00 01 7f 00 00 01 c9 69 1b 39 f1 db f3 7d  ....9...+l
0020 de 63 b8 f6 80 18 18 e7 fe 56 00 00 01 01 08 0a  .C.....V.....
0030 04 56 18 38 04 54 93 59 00 2c 6f 70 65 72 61 74  .V..T.V...operat
0040 69 6f 6e 3a 4f 50 5f 4d 41 4b 45 5f 52 4f 4f 4d  ion:OP_M AKEL_ROOM
0050 5f 41 44 4d 49 4e 0a 76 61 6c 75 65 3a 68 65 72  _ADMIN v alue:her
0060 6e 61 6e 20 0a 0a  nan..
  
```

## 12 Solicitud de un admin (sim) para hacer admin a otro (hernan)

No.	Time	Source	Destination	Protocol	Length	Info
683	1285.995463	127.0.0.1	127.0.0.1	TCP	75	6969 → 51430 [PSH, ACK] Seq=361 Ack=261 Win=408000 Len=19 TSval=72651609 TSecr=726
684	1285.995499	127.0.0.1	127.0.0.1	TCP	56	51430 → 6969 [ACK] Seq=261 Ack=380 Win=407872 Len=0 TSval=72651609 TSecr=72651609
685	1285.995610	127.0.0.1	127.0.0.1	TCP	98	6969 → 51561 [PSH, ACK] Seq=230 Ack=77 Win=408192 Len=42 TSval=72651609 TSecr=7258
686	1285.995629	127.0.0.1	127.0.0.1	TCP	56	51561 → 6969 [ACK] Seq=77 Ack=272 Win=408000 Len=0 TSval=72651609 TSecr=72651609
687	1285.995694	127.0.0.1	127.0.0.1	TCP	98	6969 → 51430 [PSH, ACK] Seq=380 Ack=261 Win=408000 Len=42 TSval=72651609 TSecr=726
688	1285.995714	127.0.0.1	127.0.0.1	TCP	56	51430 → 6969 [ACK] Seq=261 Ack=422 Win=407872 Len=0 TSval=72651609 TSecr=72651609
693	1385.912027	127.0.0.1	127.0.0.1	TCP	102	51561 → 6969 [PSH, ACK] Seq=77 Ack=272 Win=408000 Len=46 TSval=72751160 TSecr=7265
694	1385.912066	127.0.0.1	127.0.0.1	TCP	56	6969 → 51561 [ACK] Seq=272 Ack=123 Win=408128 Len=0 TSval=72751160 TSecr=72751160
695	1385.912524	127.0.0.1	127.0.0.1	TCP	75	6969 → 51561 [PSH, ACK] Seq=272 Ack=123 Win=408128 Len=19 TSval=72751160 TSecr=727
696	1385.912579	127.0.0.1	127.0.0.1	TCP	56	51561 → 6969 [ACK] Seq=123 Ack=291 Win=408000 Len=0 TSval=72751160 TSecr=72751160

▶ Frame 695: 75 bytes on wire (600 bits), 75 bytes captured (600 bits) on interface lo0, id 0  
 ▶ Null/Loopback  
 ▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1  
 ▶ Transmission Control Protocol, Src Port: 6969, Dst Port: 51561, Seq: 272, Ack: 123, Len: 19  
 ▶ Data (19 bytes)

```

0000 02 00 00 00 45 00 00 47 00 00 40 00 00 06 00 00  ....E..G...@...
0010 7f 00 00 01 7f 00 00 01 1b 39 c9 69 de 63 b8 f6  ....9...+l
0020 f1 db f3 ab 80 18 18 e9 fe 3b 00 00 01 01 08 0a  ....:.....
0030 04 56 18 38 04 56 18 38 00 11 6f 70 65 72 61 74  .V..V..V...operat
0040 69 6f 6e 3a 4f 50 5f 4f 4b 0a 0a  ion:OP_O K..
  
```

13 server responde la solicitud: tarea realizada

No.	Time	Source	Destination	Protocol	Length	Info
687	1285.995694	127.0.0.1	127.0.0.1	TCP	98	6969 → 51430 [PSH, ACK] Seq=380 Ack=261 Win=408000 Len=42 TSval=72651609 TSecr=72651609
688	1285.995714	127.0.0.1	127.0.0.1	TCP	56	51430 → 6969 [ACK] Seq=261 Ack=422 Win=407872 Len=0 TSval=72651609 TSecr=72651609
693	1385.912027	127.0.0.1	127.0.0.1	TCP	102	51561 → 6969 [PSH, ACK] Seq=77 Ack=272 Win=408000 Len=46 TSval=72751160 TSecr=72651609
694	1385.912066	127.0.0.1	127.0.0.1	TCP	56	6969 → 51561 [ACK] Seq=272 Ack=123 Win=408128 Len=0 TSval=72751160 TSecr=72751160
695	1385.912524	127.0.0.1	127.0.0.1	TCP	75	6969 → 51561 [PSH, ACK] Seq=272 Ack=123 Win=408128 Len=19 TSval=72751160 TSecr=72751160
696	1385.912579	127.0.0.1	127.0.0.1	TCP	56	51561 → 6969 [ACK] Seq=123 Ack=291 Win=408000 Len=0 TSval=72751160 TSecr=72751160
705	1467.450282	127.0.0.1	127.0.0.1	TCP	82	51430 → 6969 [PSH, ACK] Seq=261 Ack=422 Win=407872 Len=26 TSval=72832338 TSecr=72651609
706	1467.450385	127.0.0.1	127.0.0.1	TCP	56	6969 → 51430 [ACK] Seq=422 Ack=287 Win=408000 Len=0 TSval=72832338 TSecr=72832338
707	1467.450983	127.0.0.1	127.0.0.1	TCP	147	6969 → 51430 [PSH, ACK] Seq=422 Ack=287 Win=408000 Len=91 TSval=72832338 TSecr=72832338
708	1467.451004	127.0.0.1	127.0.0.1	TCP	56	51430 → 6969 [ACK] Seq=287 Ack=513 Win=407744 Len=0 TSval=72832338 TSecr=72832338

▶ Frame 705: 82 bytes on wire (656 bits), 82 bytes captured (656 bits) on interface lo0, id 0

▶ Null/Loopback

▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

▶ Transmission Control Protocol, Src Port: 51430, Dst Port: 6969, Seq: 261, Ack: 422, Len: 26

▶ Data (26 bytes)

0000 02 00 00 00 45 00 00 4c 00 00 40 00 40 06 00 00 .....E..N...@...  
0010 7f 00 00 01 7f 00 00 01 c8 e6 1b 39 2a 80 2b 99 .....9...+...  
0020 5d d0 47 69 80 18 18 e5 fe 42 00 00 01 01 08 0a ]G1....B.....  
0030 04 57 55 52 04 54 93 59 00 18 6f 70 65 72 61 74 WUR.T.Y...operat  
0040 69 6f 6e 3a 4f 50 5f 52 4f 4f 4d 5f 49 4e 46 4f ion:OP\_R\_OOM\_INFO  
0050 8a 0a

14 user solicita información de la sala

No.	Time	Source	Destination	Protocol	Length	Info
687	1285.995694	127.0.0.1	127.0.0.1	TCP	98	6969 → 51430 [PSH, ACK] Seq=380 Ack=261 Win=408000 Len=42 TSval=72651609 TSecr=72651609
688	1285.995714	127.0.0.1	127.0.0.1	TCP	56	51430 → 6969 [ACK] Seq=261 Ack=422 Win=407872 Len=0 TSval=72651609 TSecr=72651609
693	1385.912027	127.0.0.1	127.0.0.1	TCP	102	51561 → 6969 [PSH, ACK] Seq=77 Ack=272 Win=408000 Len=46 TSval=72751160 TSecr=72651609
694	1385.912066	127.0.0.1	127.0.0.1	TCP	56	6969 → 51561 [ACK] Seq=272 Ack=123 Win=408128 Len=0 TSval=72751160 TSecr=72751160
695	1385.912524	127.0.0.1	127.0.0.1	TCP	75	6969 → 51561 [PSH, ACK] Seq=272 Ack=123 Win=408128 Len=19 TSval=72751160 TSecr=72751160
696	1385.912579	127.0.0.1	127.0.0.1	TCP	56	51561 → 6969 [ACK] Seq=123 Ack=291 Win=408000 Len=0 TSval=72751160 TSecr=72751160
705	1467.450282	127.0.0.1	127.0.0.1	TCP	82	51430 → 6969 [PSH, ACK] Seq=261 Ack=422 Win=407872 Len=26 TSval=72832338 TSecr=72651609
706	1467.450385	127.0.0.1	127.0.0.1	TCP	56	6969 → 51430 [ACK] Seq=422 Ack=287 Win=408000 Len=0 TSval=72832338 TSecr=72832338
707	1467.450983	127.0.0.1	127.0.0.1	TCP	147	6969 → 51430 [PSH, ACK] Seq=422 Ack=287 Win=408000 Len=91 TSval=72832338 TSecr=72832338
708	1467.451004	127.0.0.1	127.0.0.1	TCP	56	51430 → 6969 [ACK] Seq=287 Ack=513 Win=407744 Len=0 TSval=72832338 TSecr=72832338

▶ Frame 707: 147 bytes on wire (1176 bits), 147 bytes captured (1176 bits) on interface lo0, id 0

▶ Null/Loopback

▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

▶ Transmission Control Protocol, Src Port: 6969, Dst Port: 51430, Seq: 422, Ack: 287, Len: 91

▶ Data (91 bytes)

0000 02 00 00 00 45 00 00 8f 00 00 40 00 40 06 00 00 .....E...@...  
0010 7f 00 00 01 7f 00 00 01 1b 39 c8 e6 5d d0 47 69 .....9...]G1  
0020 2a 80 2b b3 80 18 18 e7 fe 83 00 00 01 01 08 0a \*+.....  
0030 04 57 55 52 04 57 55 52 00 59 6f 70 65 72 61 74 WUR.WUR.Yoperat  
0040 69 6f 6e 3a 4f 50 5f 52 4f 4f 4d 5f 49 4e 46 4f ion:OP\_R\_OOM\_INFO  
0050 5f 52 45 53 50 4f 4e 53 45 0a 76 61 6c 75 65 3a \_RESPONS:E:valu:  
0060 65 78 61 6d 70 6c 65 52 6f 6f 6d 40 73 69 6d 3a exampleR\_oomsin:  
0070 68 65 72 6e 61 6e 3a 40 73 69 6d 3a 68 65 72 6e hernani:0sim:hern  
0080 61 6e 3a 40 31 36 35 32 30 32 36 38 34 30 38 32 an:@1652 02684082  
0090 39 0a 0a 9:

15 room\_info\_response



No.	Time	Source	Destination	Protocol	Length	Info
695	1385.912524	127.0.0.1	127.0.0.1	TCP	75	6969 → 51561 [PSH, ACK] Seq=272 Ack=123 Win=408128 Len=19 TSval=72751160 TSecr=727
696	1385.912579	127.0.0.1	127.0.0.1	TCP	56	51561 → 6969 [ACK] Seq=123 Ack=291 Win=408000 Len=0 TSval=72751160 TSecr=72751160
705	1467.450282	127.0.0.1	127.0.0.1	TCP	82	51430 → 6969 [PSH, ACK] Seq=261 Ack=422 Win=407872 Len=26 TSval=72832338 TSecr=726
706	1467.450385	127.0.0.1	127.0.0.1	TCP	56	6969 → 51430 [ACK] Seq=422 Ack=287 Win=408000 Len=0 TSval=72832338 TSecr=72832338
707	1467.450983	127.0.0.1	127.0.0.1	TCP	147	6969 → 51430 [PSH, ACK] Seq=422 Ack=287 Win=408000 Len=91 TSval=72832338 TSecr=728
708	1467.451004	127.0.0.1	127.0.0.1	TCP	56	51430 → 6969 [ACK] Seq=287 Ack=513 Win=407744 Len=0 TSval=72832338 TSecr=72832338
719	1549.674229	127.0.0.1	127.0.0.1	TCP	88	51430 → 6969 [PSH, ACK] Seq=287 Ack=513 Win=407744 Len=32 TSval=72914207 TSecr=728
720	1549.674279	127.0.0.1	127.0.0.1	TCP	56	6969 → 51430 [ACK] Seq=513 Ack=319 Win=407936 Len=0 TSval=72914207 TSecr=72914207
721	1549.675221	127.0.0.1	127.0.0.1	TCP	185	6969 → 51430 [PSH, ACK] Seq=513 Ack=319 Win=407936 Len=129 TSval=72914207 TSecr=72
722	1549.675252	127.0.0.1	127.0.0.1	TCP	56	51430 → 6969 [ACK] Seq=319 Ack=642 Win=407616 Len=0 TSval=72914208 TSecr=72914207

▶ Frame 719: 88 bytes on wire (704 bits), 88 bytes captured (704 bits) on interface lo0, id 0  
 ▶ Null/Loopback  
 ▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1  
 ▶ Transmission Control Protocol, Src Port: 51430, Dst Port: 6969, Seq: 287, Ack: 513, Len: 32  
 ▶ Data (32 bytes)

```

0000 02 00 00 00 45 00 00 54 00 00 40 00 00 06 00 00  ....E..T...@...
0010 7f 00 00 01 7f 00 00 01 c8 e6 1b 39 2a 80 2b b3  .....9...+...
0020 5d d0 47 c4 80 18 18 e3 fe 48 00 00 01 01 08 0a  }G.....H.....
0030 04 58 95 1f 04 57 55 52 00 1e 6f 70 65 72 61 74  .X..WUR..operat
0040 69 6f 6e 3a 4f 50 5f 48 49 53 54 4f 52 59 5f 52  ion:OP_HISTORY_R
0050 45 51 55 45 53 54 0a 0a                          EQUEST..
  
```

## 16 user solicita historial de chat

No.	Time	Source	Destination	Protocol	Length	Info
695	1385.912524	127.0.0.1	127.0.0.1	TCP	75	6969 → 51561 [PSH, ACK] Seq=272 Ack=123 Win=408128 Len=19 TSval=72751160 TSecr=727
696	1385.912579	127.0.0.1	127.0.0.1	TCP	56	51561 → 6969 [ACK] Seq=123 Ack=291 Win=408000 Len=0 TSval=72751160 TSecr=72751160
705	1467.450282	127.0.0.1	127.0.0.1	TCP	82	51430 → 6969 [PSH, ACK] Seq=261 Ack=422 Win=407872 Len=26 TSval=72832338 TSecr=726
706	1467.450385	127.0.0.1	127.0.0.1	TCP	56	6969 → 51430 [ACK] Seq=422 Ack=287 Win=408000 Len=0 TSval=72832338 TSecr=72832338
707	1467.450983	127.0.0.1	127.0.0.1	TCP	147	6969 → 51430 [PSH, ACK] Seq=422 Ack=287 Win=408000 Len=91 TSval=72832338 TSecr=728
708	1467.451004	127.0.0.1	127.0.0.1	TCP	56	51430 → 6969 [ACK] Seq=287 Ack=513 Win=407744 Len=0 TSval=72832338 TSecr=72832338
719	1549.674229	127.0.0.1	127.0.0.1	TCP	88	51430 → 6969 [PSH, ACK] Seq=287 Ack=513 Win=407744 Len=32 TSval=72914207 TSecr=728
720	1549.674279	127.0.0.1	127.0.0.1	TCP	56	6969 → 51430 [ACK] Seq=513 Ack=319 Win=407936 Len=0 TSval=72914207 TSecr=72914207
721	1549.675221	127.0.0.1	127.0.0.1	TCP	185	6969 → 51430 [PSH, ACK] Seq=513 Ack=319 Win=407936 Len=129 TSval=72914207 TSecr=72
722	1549.675252	127.0.0.1	127.0.0.1	TCP	56	51430 → 6969 [ACK] Seq=319 Ack=642 Win=407616 Len=0 TSval=72914208 TSecr=72914207

▶ Frame 721: 185 bytes on wire (1480 bits), 185 bytes captured (1480 bits) on interface lo0, id 0  
 ▶ Null/Loopback  
 ▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1  
 ▶ Transmission Control Protocol, Src Port: 6969, Dst Port: 51430, Seq: 513, Ack: 319, Len: 129  
 ▶ Data (129 bytes)

```

0000 02 00 00 00 45 00 00 b5 00 00 40 00 00 06 00 00  ....E.....@...
0010 7f 00 00 01 7f 00 00 01 1b 39 c8 e6 5d d0 47 c4  .....9...}G...
0020 2a 80 2b d3 80 18 18 e6 fe a9 00 00 01 01 08 0a  *.X..X..operat...
0030 04 58 95 1f 04 58 95 1f 80 7f 6f 70 65 72 61 74  .X..X..operat...
0040 69 6f 6e 3a 4f 50 5f 48 49 53 54 4f 52 59 5f 52  ion:OP_HISTORY_R
0050 45 53 50 4f 4e 53 45 0a 65 6c 65 6d 65 6e 74 73  ESPONSE elements
0060 3a 5b 32 30 32 32 d3 30 35 2d 30 38 54 31 36 3a  :[2022-05-08T16:
0070 31 37 3a 35 37 2e 39 30 39 31 39 33 5a 5d 20 68  17:57.90 9193Z] h
0080 65 72 6e 61 6e 3a 20 68 6f 6c 61 2c 5b 32 30 32  ernan: h ola,[202
0090 32 2d 30 35 2d 30 30 54 31 36 3a 32 30 3a 34 30  2-05-08T16:20:40
00a0 2e 38 32 39 32 32 37 5a 5d 20 68 65 72 6e 61 6e  .829227Z] hernan
00b0 3a 20 61 64 69 6f 73 0a 0a                          : adios .
  
```

## 17 history chat response

No.	Time	Source	Destination	Protocol	Length	Info
721	1549.675221	127.0.0.1	127.0.0.1	TCP	185	6969 → 51430 [PSH, ACK] Seq=513 Ack=319 Win=407936 Len=129 TSval=72914207 TSecr=72
722	1549.675252	127.0.0.1	127.0.0.1	TCP	56	51430 → 6969 [ACK] Seq=319 Ack=642 Win=407616 Len=0 TSval=72914208 TSecr=72914207
729	1615.262693	127.0.0.1	127.0.0.1	TCP	82	51430 → 6969 [PSH, ACK] Seq=319 Ack=642 Win=407616 Len=26 TSval=72979392 TSecr=729
730	1615.262761	127.0.0.1	127.0.0.1	TCP	56	6969 → 51430 [ACK] Seq=642 Ack=345 Win=407936 Len=0 TSval=72979392 TSecr=72979392
731	1615.263360	127.0.0.1	127.0.0.1	TCP	97	6969 → 51561 [PSH, ACK] Seq=291 Ack=123 Win=408128 Len=41 TSval=72979392 TSecr=727
732	1615.263395	127.0.0.1	127.0.0.1	TCP	56	51561 → 6969 [ACK] Seq=123 Ack=332 Win=407936 Len=0 TSval=72979392 TSecr=72979392
733	1617.135838	127.0.0.1	127.0.0.1	TCP	56	51430 → 6969 [FIN, ACK] Seq=345 Ack=642 Win=407616 Len=0 TSval=72981254 TSecr=7297
734	1617.135896	127.0.0.1	127.0.0.1	TCP	56	6969 → 51430 [ACK] Seq=642 Ack=346 Win=407936 Len=0 TSval=72981254 TSecr=72981254
735	1617.137113	127.0.0.1	127.0.0.1	TCP	56	6969 → 51430 [FIN, ACK] Seq=642 Ack=346 Win=407936 Len=0 TSval=72981255 TSecr=7298
736	1617.137157	127.0.0.1	127.0.0.1	TCP	56	51430 → 6969 [ACK] Seq=346 Ack=643 Win=407616 Len=0 TSval=72981255 TSecr=72981255

▶ Frame 729: 82 bytes on wire (656 bits), 82 bytes captured (656 bits) on interface lo0, id 0  
 ▶ Null/Loopback  
 ▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1  
 ▶ Transmission Control Protocol, Src Port: 51430, Dst Port: 6969, Seq: 319, Ack: 642, Len: 26  
 ▶ Data (26 bytes)

```

0000 02 00 00 00 45 00 00 4e 00 00 40 00 00 06 00 00  ....E..N...@...
0010 7f 00 00 01 7f 00 00 01 c8 e6 1b 39 2a 80 2b d3  .....9...+...
0020 5d d0 48 45 08 18 18 c1 fe 42 00 00 01 01 08 0a  }ME.....B.....
0030 04 59 93 c0 04 59 95 1f 80 18 6f 70 65 72 61 74  .Y..X..operat...
0040 69 6f 6e 3a 4f 50 5f 45 58 49 54 5f 52 4f 4f 4d  ion:OP_EXIT_ROOM
0050 0a 0a
  
```

## 18 usuario abandona la sala

```

tcp.stream & tcp.port == 6969
No.    Time           Source            Destination      Protocol Length Info
-----
721    1549.675221    127.0.0.1        127.0.0.1        TCP              185 6969 → 51430 [PSH, ACK] Seq=513 Ack=319 Win=407936 Len=129 TSval=72914207 TSecr=72
722    1549.675252    127.0.0.1        127.0.0.1        TCP              56 51430 → 6969 [ACK] Seq=319 Ack=642 Win=407616 Len=0 TSval=72914208 TSecr=72914207
729    1615.262693    127.0.0.1        127.0.0.1        TCP              82 51430 → 6969 [PSH, ACK] Seq=319 Ack=642 Win=407616 Len=26 TSval=72979392 TSecr=729
730    1615.262761    127.0.0.1        127.0.0.1        TCP              56 6969 → 51430 [ACK] Seq=642 Ack=345 Win=407936 Len=0 TSval=72979392 TSecr=72979392
731    1615.263360    127.0.0.1        127.0.0.1        TCP              97 6969 → 51561 [PSH, ACK] Seq=291 Ack=123 Win=408128 Len=41 TSval=72979392 TSecr=727
732    1615.263395    127.0.0.1        127.0.0.1        TCP              56 51561 → 6969 [ACK] Seq=123 Ack=332 Win=407936 Len=0 TSval=72979392 TSecr=72979392
733    1617.135838    127.0.0.1        127.0.0.1        TCP              56 51430 → 6969 [FIN, ACK] Seq=345 Ack=642 Win=407616 Len=0 TSval=72981254 TSecr=7297
734    1617.135896    127.0.0.1        127.0.0.1        TCP              56 6969 → 51430 [ACK] Seq=642 Ack=346 Win=407936 Len=0 TSval=72981254 TSecr=72981254
735    1617.137113    127.0.0.1        127.0.0.1        TCP              56 6969 → 51430 [FIN, ACK] Seq=642 Ack=346 Win=407936 Len=0 TSval=72981255 TSecr=7298
736    1617.137157    127.0.0.1        127.0.0.1        TCP              56 51430 → 6969 [ACK] Seq=346 Ack=643 Win=407616 Len=0 TSval=72981255 TSecr=72981255

Frame 731: 97 bytes on wire (776 bits), 97 bytes captured (776 bits) on interface lo0, id 0
Null/Loopback
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 6969, Dst Port: 51561, Seq: 291, Ack: 123, Len: 41
Data (41 bytes)

0000 02 00 00 00 45 00 00 5d 00 00 40 00 40 06 00 00  ....E...@...
0010 7f 00 00 01 7f 00 00 01 1b 39 c9 69 de 63 b9 09  ....9...i...
0020 f1 db f3 ab 80 18 18 e9 fe 51 00 00 01 01 08 0a  ....Q.....
0030 04 59 93 c0 04 56 19 38 00 27 6f 70 65 72 61 74  ....Y...V...operat
0040 69 6f 6e 3a 4f 50 5f 45 58 49 54 45 44 5f 55 53  ....ign:OP,E XITED_US
0050 45 52 0a 76 61 6c 75 65 3a 68 65 72 6e 61 6e 0a  ER-value :hernan
0060 0a

```

19 broadcast al resto de usuarios de que el usuario ha abandonado la sala

```

tcp.stream & tcp.port == 6969
No.    Time           Source            Destination      Protocol Length Info
-----
721    1549.675221    127.0.0.1        127.0.0.1        TCP              185 6969 → 51430 [PSH, ACK] Seq=513 Ack=319 Win=407936 Len=129 TSval=72914207 TSecr=72
722    1549.675252    127.0.0.1        127.0.0.1        TCP              56 51430 → 6969 [ACK] Seq=319 Ack=642 Win=407616 Len=0 TSval=72914208 TSecr=72914207
729    1615.262693    127.0.0.1        127.0.0.1        TCP              82 51430 → 6969 [PSH, ACK] Seq=319 Ack=642 Win=407616 Len=26 TSval=72979392 TSecr=729
730    1615.262761    127.0.0.1        127.0.0.1        TCP              56 6969 → 51430 [ACK] Seq=642 Ack=345 Win=407936 Len=0 TSval=72979392 TSecr=72979392
731    1615.263360    127.0.0.1        127.0.0.1        TCP              97 6969 → 51561 [PSH, ACK] Seq=291 Ack=123 Win=408128 Len=41 TSval=72979392 TSecr=727
732    1615.263395    127.0.0.1        127.0.0.1        TCP              56 51561 → 6969 [ACK] Seq=123 Ack=332 Win=407936 Len=0 TSval=72979392 TSecr=72979392
733    1617.135838    127.0.0.1        127.0.0.1        TCP              56 51430 → 6969 [FIN, ACK] Seq=345 Ack=642 Win=407616 Len=0 TSval=72981254 TSecr=7297
734    1617.135896    127.0.0.1        127.0.0.1        TCP              56 6969 → 51430 [ACK] Seq=642 Ack=346 Win=407936 Len=0 TSval=72981254 TSecr=72981254
735    1617.137113    127.0.0.1        127.0.0.1        TCP              56 6969 → 51430 [FIN, ACK] Seq=642 Ack=346 Win=407936 Len=0 TSval=72981255 TSecr=7298
736    1617.137157    127.0.0.1        127.0.0.1        TCP              56 51430 → 6969 [ACK] Seq=346 Ack=643 Win=407616 Len=0 TSval=72981255 TSecr=72981255

Frame 733: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface lo0, id 0
Null/Loopback
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 51430, Dst Port: 6969, Seq: 345, Ack: 642, Len: 0

0000 02 00 00 00 45 00 00 3d 00 00 40 00 40 06 00 00  ....E...@...
0010 7f 00 00 01 7f 00 00 01 c8 e6 1b 39 2a 80 2b ed  ....9...e...
0020 5d d0 48 45 80 11 18 e1 fe 28 00 00 01 01 08 0a  ....].HE.....
0030 04 59 9b 06 04 59 93 c0  ....Y...Y...

"6969ok response" is not a valid number
Packets: 773 - Displayed: 78 (10.1%)
Profile: Default

```

20 finalmente se cierra la conexión TCP con el servidor