

# Proyecto 2

Carlos Gerardo Acosta Hernández  
Andrea Itzel González Vargas  
Luis Pablo Mayo Vega

Redes de Computadoras  
Facultad de Ciencias, UNAM

## Índice

<b>1. Especificación de requerimientos</b>	<b>2</b>	<b>3. Implementación del protocolo</b>	<b>6</b>
1.1. Enunciado del problema . . . . .	2	3.1. Especificación del ambiente de desarrollo . . . . .	6
1.2. Objetivo de la aplicación . . . . .	2	3.2. Repositorio en línea . . . . .	6
<b>2. Diseño del protocolo</b>	<b>4</b>	3.3. Estructura del proyecto . . . . .	6
2.1. Máquina de Estados Finita . . . . .	4	<b>4. Uso y pruebas del protocolo</b>	<b>8</b>
2.2. Descripción de los estados . . . . .	4	4.1. Manual de uso . . . . .	8
2.3. Descripción de los mensajes en la comunicación cliente-servidor . . . . .	5	4.2. Demostración del funcionamiento	10
2.4. Diseño de la base de datos . . . . .	5	4.3. Demostración del funcionamiento	10

# 1. Especificación de requerimientos

## 1.1. Enunciado del problema

De acuerdo con el modelo TCP/IP, la *Capa de Aplicación* es la de mayor importancia y en la que se sustenta todo el desarrollo de redes de computadoras pues está compuesta por los protocolos y demás servicios encargados de manejar, intercambiar o decodificar los datos que los usuarios se envían a través de distintos hosts para comunicarse. En otras palabras, se busca que dos o más procesos en distintas computadoras puedan ser capaces de intercambiar información y de esta manera otorgar una mayor capacidad de procesamiento y mejor rendimiento del que se tendría con un único host.

Sin embargo, la *Capa de Aplicación* no puede trabajar sola, siendo mediante un proceso de encapsulación en cada una de las capas, que los datos(PDU) se envían a las capas inferiores, añadiendo cada una de las capas información que le concierne para que sus protocolos puedan manejarlos.

Como sabemos, protocolos como HTTP, DNS, FTP, SMTP o DHCP tienen cada uno una estructura distinta ya bien definida y estandarizada en los artículos de *Request for Comments* publicados, pero eso no significa que sean los únicos protocolos disponibles en la *Capa de Aplicación*. La gran ventaja de esta capa es su adaptabilidad para que se desarrollen protocolos de acuerdo a las necesidades de la aplicación sobre la que se usarán.

## 1.2. Objetivo de la aplicación

Esta aplicación fue creada con el objetivo de implementar un protocolo de la capa de aplicación en el que el usuario, conectado del lado del cliente, solicite un pokémon a capturar. El servidor elegirá aleatoriamente alguno de los pokémon que estén en su base de datos, se lo ofrecerá al usuario y, si éste acepta capturarlo, también aleatoriamente se indicará si logró capturarlo o no.

El usuario también podrá ser capaz de consultar su Pokédex, con los nombres de los pokémon que ha capturado y la imagen de cada pokémon que seleccione.

El protocolo que hemos diseñado se enfocará en las acciones del usuario y el servidor que involucren una comunicación entre ambos, cada una con un tipo de mensaje específico. Es decir, si un usuario quisiera capturar un pokémon, solo tendría que enviar un mensaje con el código que el servidor entienda como "quiero capturar un pokémon" sin considerar otros aspectos como el nombre del pokémon o el tamaño de la imagen que contiene a tal pokémon.

En ese sentido, diseñar nuestro propio protocolo nos permite tener mayor control sobre la tasa de transferencia de los datos dentro de la aplicación, disminuyendo los costos de la comunicación y mejorando el desempeño del programa.

### 1.2.1. Casos de uso

Actor	Caso de uso	Descripción
Pokentrenador	Iniciar sesión	El usuario se conecta con el servidor y se le muestra el menú principal de la aplicación
Pokentrenador	Capturar pokémon	El usuario solicita al servidor que le muestre un pokémon y decide si intenta capturarlo (con un número finito de intentos) o no
Pokentrenador	Consultar pokedex	El usuario hace una consulta para buscar un pokémon en su pokédex
Pokentrenador	Cerrar sesión	Cierra la conexión con el servidor

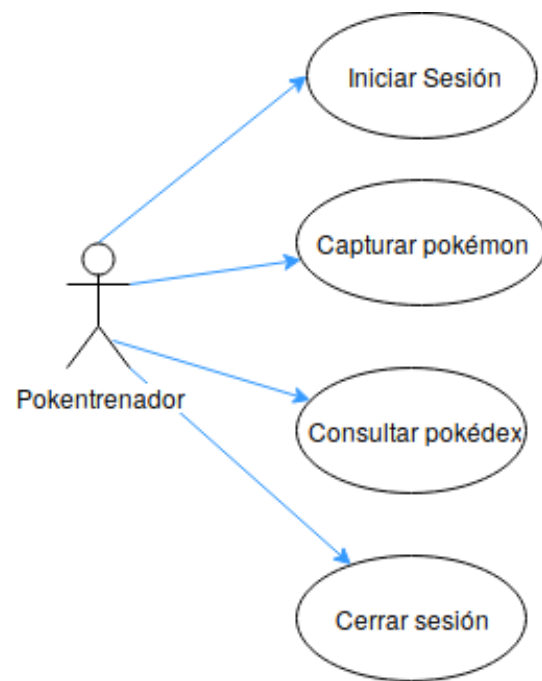
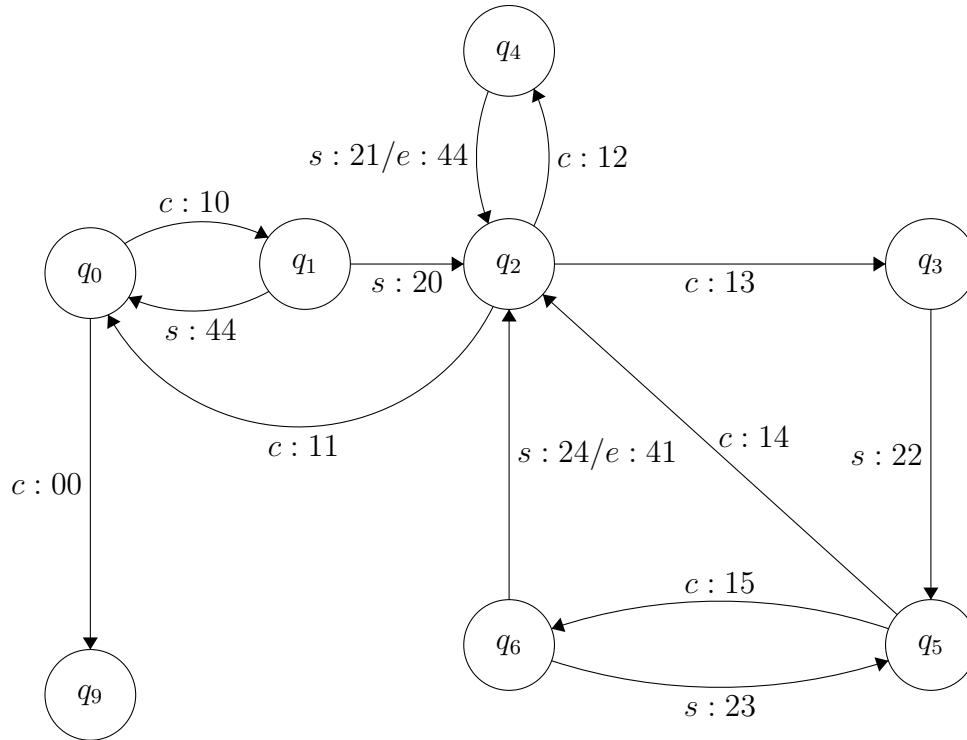


Figura 1: Diagrama de casos de uso de la aplicación

## 2. Diseño del protocolo

### 2.1. Máquina de Estados Finita

Máquina de Estados Finita para el protocolo de la capa de aplicación.



### 2.2. Descripción de los estados

Estado	Descripción
$q_0$	Conexión establecida, inicio de aplicación.
$q_1$	Inicio de sesión.
$q_2$	Menú de juego.
$q_3$	Solicitud de captura de <i>póke</i> mon.
$q_4$	Búsqueda de un <i>póke</i> mon en la <i>póke</i> dex.
$q_5$	Aparición de un <i>Póke</i> mon salvaje.
$q_6$	Intento de captura de <i>póke</i> mon.
$q_9$	Cierre de conexión.

## 2.3. Descripción de los mensajes en la comunicación cliente-servidor

Código	Segmento	Descripción
00	long code	Termina la conexión para un cliente.
10	long code name	Solicitud de inicio de sesión del cliente. El parámetro <b>name</b> se refiere al nombre de usuario del cliente.
11	long code	Solicitud de cierre de sesión del cliente.
12	long code name	Acceso y consulta a la <i>Pokédex</i> mediante el nombre de un pokémon.
13	long code	Acceso a la opción de captura.
14	long code	El usuario rechaza el pokémon salvaje ofrecido por el servidor.
15	long code attempts name	Intento de captura del pokémon salvaje ofrecido.
20	long code	Inicio de sesión del cliente exitoso.
21	long code long_n name img	Resultado exitoso de la Pokédex con imagen del pokémon.
22	long code attempts name	Selección aleatoria de un pokémon salvaje para el cliente. Incluye el máximo número de intentos.
23	long code attempts name	Pokémon no capturado, disminución del número de intentos.
24	long code long_n name img	Pokémon capturado con imagen incluida.
41	long code	Error: Máximo número de intentos de captura alcanzados.
44	long code	Error: Consulta infructuosa.

## 2.4. Diseño de la base de datos

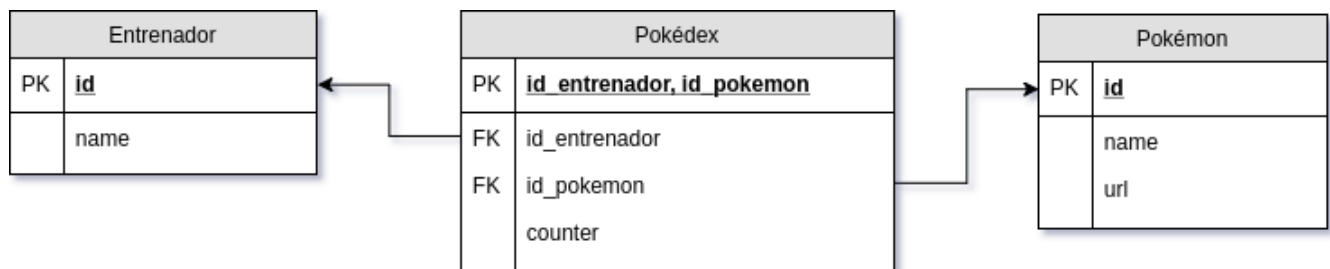


Figura 2: Diagrama de la base de datos relacional.

## 3. Implementación del protocolo

### 3.1. Especificación del ambiente de desarrollo

Lenguaje de programación:	Java	<b>versión:</b>	1.8
Herramienta de construcción de software:	Apache Ant	<b>versión:</b>	1.9.9
Sistema Manejador de Base de Datos:	sqlite	<b>versión:</b>	2.8.17
Driver de conectividad JDBC:	sqlite-jdbc	<b>versión:</b>	3.16.1
Sistema manejador de versiones:	GIT	<b>versión:</b>	2.14.1
Pruebas Unitarias:	JUnit y Hamcrest	<b>versión:</b>	4

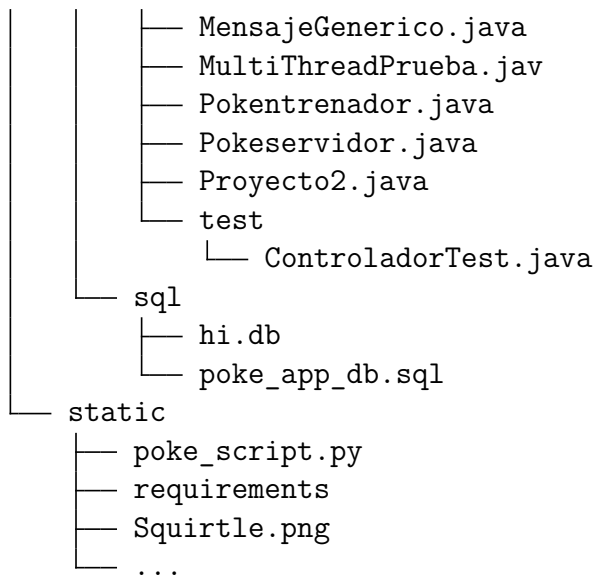
### 3.2. Repositorio en línea

GitHub - Proyecto 2

### 3.3. Estructura del proyecto

#### 3.3.1. Árbol del directorio (preinstalación)

```
p02
├── build.xml
├── documentos
│   ├── graphic
│   │   ├── casosdeuso.png
│   │   └── ...
│   ├── proyecto2.pdf
│   └── proyecto2.tex
├── lib
│   ├── hamcrest-core.jar
│   ├── junit.jar
│   └── sqlite-jdbc-3.16.1.jar
├── man
│   └── proyecto2.1
└── src
    └── redes
        ├── ClienteHilo.java
        ├── ConexionBD.java
        ├── Controlador.java
        ├── Estado.java
        ├── FabricaMensaje.java
        ├── Imagen.java
        ├── Mensaje10.java
        ├── Mensaje12.java
        ├── Mensaje15.java
        ├── Mensaje21.java
        ├── Mensaje22.java
        ├── Mensaje23.java
        └── Mensaje24.java
```



### 3.3.2. build.xml

Este archivo contiene las directivas de operación para el sistema de construcción ant. Es mediante su lectura que la estructura del proyecto da lugar a la compilación del código y el enlace entre sus dependencias.

### 3.3.3. src

En este directorio encontraremos dos subdirectorios o, más propiamente paquetes. En *redes/* se encuentra el código fuente principal del protocolo de aplicación, mientras que en *test/* se encontrarán los archivos de las pruebas unitarias contempladas para algunas de los métodos del proyecto.

### 3.3.4. static

Aquí deben encontrarse forzosamente las imágenes de los pokémon que estarán disponibles en la dinámica de la aplicación, con un nombre de archivo dado por el formato: <pokémon>.png. Además vienen incluidos, un *script* en python que permite volver a descargar en ese directorio las imágenes de los pokémons de la primera generación y su archivo de requerimientos para funcionar (que puede utilizarse mediante la introducción del comando: `$ pip install -r requirements`, previa a la ejecución del *script*, ya sea en un ambiente virtual o directamente sobre el sistema).

### 3.3.5. documentos

El documento que ahora mismo está leyendo debe encontrarse en este directorio, junto con el código fuente en *L<sup>A</sup>T<sub>E</sub>X* que lo genera. Además, el subdirectorio *graphic/* contiene las imágenes utilizadas en el código mencionado para la generación del documento *PDF*.

### 3.3.6. lib

Como ya se mencionó en la sección 3.1, se utilizaron bibliotecas externas tanto para las pruebas unitarias como para el controlador del *SMBD*. Tales recursos deben encontrarse en este directorio.

### 3.3.7. man

En este directorio se encuentra el archivo que contiene la información que será agregada al *man pages* (manual) de la distribución de *Linux* durante la instalación.

En la siguiente sección será más evidente la importancia del listado anterior, pues se describirá brevemente su interacción en el proceso de instalación y ejecución del proyecto. Adicionalmente, si se desea explorar la conformación del código a mayor detalle, recomendamos ver la sección 4.1.3.

## 4. Uso y pruebas del protocolo

### 4.1. Manual de uso

#### 4.1.1. Compilación

Para compilar el proyecto y generar un recopilado de ejecutables de la *JVM*, un archivo *JAR*. Basta con ejecutar *Apache-Ant* sin argumentos, pues se ha especificado el *target* descrito por defecto:

```
[user@host p02]$ ant
```

#### 4.1.2. Linux man pages

Si es preferible que las instrucciones generales de uso sean agregadas al *Linux man pages* para así mantener accesible la forma de uso del proyecto, preparamos el siguiente *target* en *ant* para este proposito:

```
[user@host p02]$ ant man
```

Cuya ejecución solicitará permisos de superusuario para agregar la página de manual de este proyecto. Una vez ingresada la contraseña de *sudo* será posible visitar el manual desde cualquier ruta mediante el siguiente comando:

```
[user@host ~]$ man proyecto2
```

#### 4.1.3. Generación de documentación

Para explorar la organización programática del proyecto sin necesidad de visitar todos los códigos fuente involucrados en la construcción del proyecto, recomendamos generar la documentación de código *html*, para así hallar la explicación de cada método implementado, sus entradas y salidas y el papel que desempeñan en el protocolo. Para ello, basta efectuar:

```
[user@host p02]$ ant doc
```

Esto generará un directorio *doc/*, que contiene un conjunto de documentos en formato *html* descriptivos de cada clase en el código, relacionados unos con otros a través de hipervínculos que permiten una navegación fluida que no necesariamente es posible en la lectura directa del código.



#### 4.1.4. Precondiciones

Antes de comenzar la ejecución es importante que de las secciones anteriores se haya efectuado cuando menos la 4.1.1. En caso de querer ahorrar los pasos anteriores y simplemente convenimos que se ejecuten todos, basta con el siguiente *target* de *ant*:

```
[user@host p02]$ ant all
```

También es importante que el archivo de la base de datos de *SQLite* se encuentre bajo la ruta: *src/sql/hi.db*. De no hallarse ahí, pero sí el esquema de la base de datos en el archivo *src/sql/poke\_app\_db.sql*, es posible generar la base de datos ejecutando:

```
[user@host p02/src/sql]$ sqlite3 hi.db < poke_app_db.sql
```

Finalmente, sólo resta verificar que las imágenes a utilizar por la aplicación se encuentren en la ruta adecuada. Por ejemplo, para el pokémon **charmander**, debe existir la imagen *static/charmander.png*. En caso de no hayarse ninguno, a continuación damos una serie de instrucciones<sup>1</sup> para que el *script* incluido en ese directorio descargue de internet nuevamente las imágenes que la aplicación mostrará.

```
[user@host p02/static]$ virtualenv py_env
[user@host p02/static]$ source py_env/bin/activate
(py_env)[user@host p02/static]$ pip install -r requirements
(py_env)[user@host p02/static]$ python3 poke_script.py
```

Una vez asegurandonos de esto, podemos proceder a la ejecución de la aplicación.

#### 4.1.5. Ejecución de pruebas unitarias

Si se desea incluir nuevas pruebas en el directorio *src/test/* para la verificación de alguna función del protocolo o simplemente correr las ya existentes, ejecutamos lo siguiente:

```
[user@host p02]$ ant test
```

Que compilará y ejecutará tanto el código de las pruebas como el de la aplicación y listará los tiempos, errores y salidas sin errores de cada una.

#### 4.1.6. Ejecución

La ejecución del programa como viene especificado en el manual incluido (ver 4.1.2) se realiza mediante el archivo *JAR* resultado de la compilación y posee una cantidad de argumentos tanto para un cliente como para un servidor. Es importante señalar que decidimos manejar este archivo como uno solo, es decir se utiliza tanto para la ejecución de un cliente como la de un servidor, diferenciando el comportamiento del programa mediante los argumentos.

Para ejecutar la aplicación con sus valores por defecto para cada parte de la comunicación, efectuamos al menos uno<sup>2</sup> de cada uno<sup>3</sup> de los siguientes comandos:

#### Servidor

---

<sup>1</sup>Éstas sugieren el uso de un entorno virtual, pero que no es fundamentalmente necesario.

<sup>2</sup>La aplicación soporta la conexión de múltiples clientes.

<sup>3</sup>De preferencia con un servidor ejecutándose previo a la ejecución de cualquier cliente.

```
[user@host p02]$ java -jar proyecto2.jar
```

## **Cliente**

```
[user@host p02]$ java -jar proyecto2.jar -c
```

## **4.2. Operación general**

Ya se ha visto cómo realizar la ejecución más sencilla del proyecto (4.1.6). En la presente sección mostramos una posible navegación por el programa y sus resultados, apreciables en capturas de pantalla de la aplicación.

### **4.2.1. Inicio de sesión**

Una vez establecida la conexión entre cliente y servidor, el servidor le pedirá al usuario que ingrese su nombre para poder iniciar sesión o cerrar la conexión entre ambos hosts. Si el nombre de usuario ingresado se encuentra registrado en la base de datos de la aplicación, el servidor le mostrará al usuario con sesión iniciada su menú principal de juego.

### **4.2.2. Uso dentro de la aplicación**

Una vez iniciada la sesión del usuario, éste tendrá acceso a un menú principal en el que se muestre las opciones que le otorga la aplicación, como capturar un pokémon, consultar su pokédex o cerrar sesión.

Si el usuario quiere capturar un pokémon, el servidor le mostrará aleatoriamente alguno de los nombres de pokémon registrados y el usuario decidirá si desea o no capturarlo. Si acepta el desafío, tendrá que lograr capturarlo antes de llegar al límite de intentos permitido.

Para consultar su pokédex, el usuario deberá ingresar el nombre del pokémon que quiera buscar. Si el pokémon ya fue capturado, se le mostrará la imagen de tal pokémon y recibirá un mensaje de error si el nombre del pokémon no es el correcto o el usuario no lo ha capturado.

Si el usuario decide cerrar su sesión, regresará a la pantalla de inicio de la aplicación y se le preguntará si desea iniciar sesión de nuevo o cerrar la conexión entre los hosts.

## **4.3. Demostración del funcionamiento**

A continuación se ilustra la experiencia de un cliente al ejecutar el programa, junto con los datos capturados por Wireshark de la comunicación entre cliente y servidor.

El cliente se conectó desde la misma máquina que el servidor, hacia el puerto 9999.

### **4.3.1. Inicio de sesión**

Se muestra el menú de inicio de sesión, junto con los datos ingresados por el cliente.

```

[chepe@chepe p02]$ java -jar proyecto2.jar -c
Selecciona una opción
1. Iniciar sesión
2. Cerrar conexión
1
Ingresa tu nombre de usuario
chepe
Nombre de usuario no identificado.
Selecciona una opción
1. Iniciar sesión
2. Cerrar conexión
1
Ingresa tu nombre de usuario
paulo
Inicio de sesión exitoso
Selecciona una opción
1. Utilizar pokedex
2. Capturar un pokémon
3. Cerrar sesión

```

Figura 3: Menú de inicio de sesión

En la interacción mostrada en la Figura 3, se le pidió al usuario que escogiera entre iniciar sesión y cerrar la conexión. El cliente pidió iniciar sesión con el usuario **chepe** mandando el siguiente mensaje con código 10:

```

▶ Frame 136: 78 bytes on wire (624 bits), 78 bytes captured (624 bits) on interface 0
▶ Linux cooked capture
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▶ Transmission Control Protocol, Src Port: 49402, Dst Port: 9999, Seq: 1, Ack: 1, Len: 10
▼ OpenWire (KeepAliveInfo)
  Length: 6
  Command: KeepAliveInfo (10)
  Command Id: 1667786096
  Command response required: 101

```

0000	00 00 03 04 00 06 00 00 00 00 00 00 00 00 08 00	.....
0010	45 00 00 3e f3 a5 40 00 40 06 49 12 7f 00 00 01	E..>..@. @.I....
0020	7f 00 00 01 c0 fa 27 0f 46 6b cd 6c 02 26 e9 d9	.....'. Fk.l.&..
0030	80 18 01 56 fe 32 00 00 01 01 08 0a 01 67 46 34	...V.2.. .....gF4
0040	01 67 3e c8 00 00 00 06 0a 63 68 65 70 65	.g>.....chepe

Figura 4: Mensaje con código 10

El servidor contestó con el mensaje de error con código 44, indicando que tal usuario no se encuentra registrado en la base de datos.

```

▶ Frame 138: 73 bytes on wire (584 bits), 73 bytes captured (584 bits) on interface 0
▶ Linux cooked capture
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▶ Transmission Control Protocol, Src Port: 9999, Dst Port: 49402, Seq: 1, Ack: 11, Len: 5
▼ OpenWire (Unknown (0x2c))
  Length: 1
  Command: Unknown (44)

```

0000	00 00 03 04 00 06 00 00	00 00 00 00 00 00 08 00	.....
0010	45 00 00 39 d7 12 40 00	40 06 65 aa 7f 00 00 01	E..9...@. @.e.....
0020	7f 00 00 01 27 0f c0 fa	02 26 e9 d9 46 6b cd 76	....'... .&..Fk.v
0030	80 18 01 56 fe 2d 00 00	01 01 08 0a 01 67 46 63	...V.-... ..gFc
0040	01 67 46 34 00 00 00 01	2c	.gF4.... ,

Figura 5: Mensaje de error con código 44

El cliente hace un segundo intento de inicio de sesión con un usuario distinto.

```

▶ Frame 215: 78 bytes on wire (624 bits), 78 bytes captured (624 bits) on interface 0
▶ Linux cooked capture
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▶ Transmission Control Protocol, Src Port: 49402, Dst Port: 9999, Seq: 11, Ack: 6, Len: 10
▼ OpenWire (KeepAliveInfo)
  Length: 6
  Command: KeepAliveInfo (10)
  Command Id: 1885435244
  Command response required: 111

```

0000	00 00 03 04 00 06 00 00	00 00 00 00 11 78 08 00	.....x..
0010	45 00 00 3e f3 a7 40 00	40 06 49 10 7f 00 00 01	E..>..@. @.I.....
0020	7f 00 00 01 c0 fa 27 0f	46 6b cd 76 02 26 e9 de	.....'. Fk.v.&..
0030	80 18 01 56 fe 32 00 00	01 01 08 0a 01 67 54 25	...V.2... ..gT%
0040	01 67 46 63 00 00 00 06	0a 70 61 75 6c 6f	.gFc.... .paulo

Figura 6: Mensaje con código 10

El servidor responde con el código 20, indicando que se inicio sesión exitosamente.

```

▶ Frame 217: 73 bytes on wire (584 bits), 73 bytes captured (584 bits) on interface 0
▶ Linux cooked capture
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▶ Transmission Control Protocol, Src Port: 9999, Dst Port: 49402, Seq: 6, Ack: 21, Len: 5
▼ OpenWire (MessagePull)
  Length: 1
  Command: MessagePull (20)

```

0000	00 00 03 04 00 06 00 00	00 00 00 00 fa ab 08 00	.....
0010	45 00 00 39 d7 14 40 00	40 06 65 a8 7f 00 00 01	E..9...@. @.e.....
0020	7f 00 00 01 27 0f c0 fa	02 26 e9 de 46 6b cd 80	....'... .&..Fk..
0030	80 18 01 56 fe 2d 00 00	01 01 08 0a 01 67 54 26	...V.-... ..gT&
0040	01 67 54 25 00 00 00 01	14	.gT%.... .

Figura 7: Mensaje con código 20

### 4.3.2. Búsqueda de pokémon en Pokédex

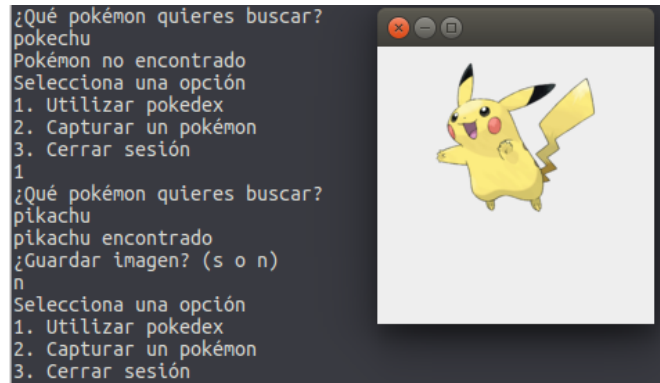


Figura 8: Búsquedas en Pokédex

El cliente indica que quiere hacer una búsqueda del pokémon pokechu en su Pokédex.

```
▶ Frame 10455: 80 bytes on wire (640 bits), 80 bytes captured (640 bits) on interface 0
▶ Linux cooked capture
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▶ Transmission Control Protocol, Src Port: 49402, Dst Port: 9999, Seq: 21, Ack: 11, Len: 12
▼ OpenWire (RemoveInfo)
  Length: 8
  Command: RemoveInfo (12)
  Command Id: 1886350181
  Command response required: 99
0000  00 00 03 04 00 06 00 00 00 00 00 00 00 06 08 00  .....
0010  45 00 00 40 f3 a9 40 00 40 06 49 0c 7f 00 00 01  E..@..@. @.I....
0020  7f 00 00 01 c0 fa 27 0f 46 6b cd 80 02 26 e9 e3  ....' . Fk...&..
0030  80 18 01 56 fe 34 00 00 01 01 08 0a 01 68 22 59  ...V.4.. ....h"Y
0040  01 67 54 26 00 00 00 08 0c 70 6f 6b 65 63 68 75  .gT&.... .pokechu
```

Figura 9: Mensaje con código 12

El servidor contesta con el código de error 44, indicando que no se encontró tal pokémon.

```
▶ Frame 10456: 73 bytes on wire (584 bits), 73 bytes captured (584 bits) on interface 0
▶ Linux cooked capture
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▶ Transmission Control Protocol, Src Port: 9999, Dst Port: 49402, Seq: 11, Ack: 33, Len: 5
▼ OpenWire (Unknown (0x2c))
  Length: 1
  Command: Unknown (44)
0000  00 00 03 04 00 06 00 00 00 00 00 00 00 00 08 00  .....
0010  45 00 00 39 d7 15 40 00 40 06 65 a7 7f 00 00 01  E..9..@. @.e....
0020  7f 00 00 01 27 0f c0 fa 02 26 e9 e3 46 6b cd 8c  ....'... .&..Fk..
0030  80 18 01 56 fe 2d 00 00 01 01 08 0a 01 68 22 5a  ...V.-... ....h"Z
0040  01 68 22 59 00 00 00 01 2c  ..... ,
```

Figura 10: Mensaje con código de error 44

El cliente pide que se realice la búsqueda de otro pokémon, en este caso pikachu.

```

▶ Frame 10464: 80 bytes on wire (640 bits), 80 bytes captured (640 bits) on interface 0
▶ Linux cooked capture
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▶ Transmission Control Protocol, Src Port: 49402, Dst Port: 9999, Seq: 33, Ack: 16, Len: 12
▼ OpenWire (RemoveInfo)
  Length: 8
  Command: RemoveInfo (12)
  Command Id: 1885956961
  Command response required: 99

```

0000	00 00 03 04 00 06 00 00	00 00 00 00 00 00 08 00	.....
0010	45 00 00 40 f3 ab 40 00	40 06 49 0a 7f 00 00 01	E..@..@. @.I.....
0020	7f 00 00 01 c0 fa 27 0f	46 6b cd 8c 02 26 e9 e8	.....'. Fk...&..
0030	80 18 01 56 fe 34 00 00	01 01 08 0a 01 68 2a b6	...V.4.. ....h*.
0040	01 68 22 5a 00 00 00 08	0c 70 69 6b 61 63 68 75	.h"Z.... .pikachu

Figura 11: Mensaje con código 12

El servidor contesta con el código de éxito 21 junto con la imagen del pokémon, la cual se despliega en pantalla y se indica si se quiere guardar o no en el equipo del cliente.

```

▶ Frame 10465: 16171 bytes on wire (129368 bits), 16171 bytes captured (129368 bits) on interface 0
▶ Linux cooked capture
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▶ Transmission Control Protocol, Src Port: 9999, Dst Port: 49402, Seq: 16, Ack: 45, Len: 16103
▼ OpenWire (MessageDispatch)
  Length: 16099
  Command: MessageDispatch (21)
  Command Id: 7
  Command response required: 112
  ConsumerId: Unknown (0x6b)

```

0040	01 68 2a b6 00 00 3e e3	15 00 00 00 07 70 69 6b	.h*..>. ....pik
0050	01 63 68 75 89 50 4e 47	0d 0a 1a 0a 00 00 00 00	achu.PNG .....
0060	49 48 44 52 00 00 00 78	00 00 00 78 08 06 00 00	IHDR...x ...x....
0070	00 39 64 36 d2 00 00 00	04 67 41 4d 41 00 00 b1	.9d6.... .gAMA...
0080	8f 0b fc 61 05 00 00 00	20 63 48 52 4d 00 00 7a	...a.... cHRM..z
0090	26 00 00 80 84 00 00 fa	00 00 00 80 e8 00 00 75	&..... ....u
00a0	30 00 00 ea 60 00 00 3a	98 00 00 17 70 9c ba 51	0...`.: ....p..Q
00b0	3c 00 00 00 06 62 4b 47	44 00 ff 00 ff 00 ff a0	<....bKG D.....
00c0	bd a7 93 00 00 00 09 70	48 59 73 00 00 2d fc 00	.....p HYS...-
00d0	00 2d fc 01 ae c3 ec 98	00 00 00 07 74 49 4d 45	.-..... ....tIME

Figura 12: Mensaje con código 21

### 4.3.3. Captura de un pokémon



Figura 13: Intentos de captura de un pokémon

El cliente pide capturar un pokémon.

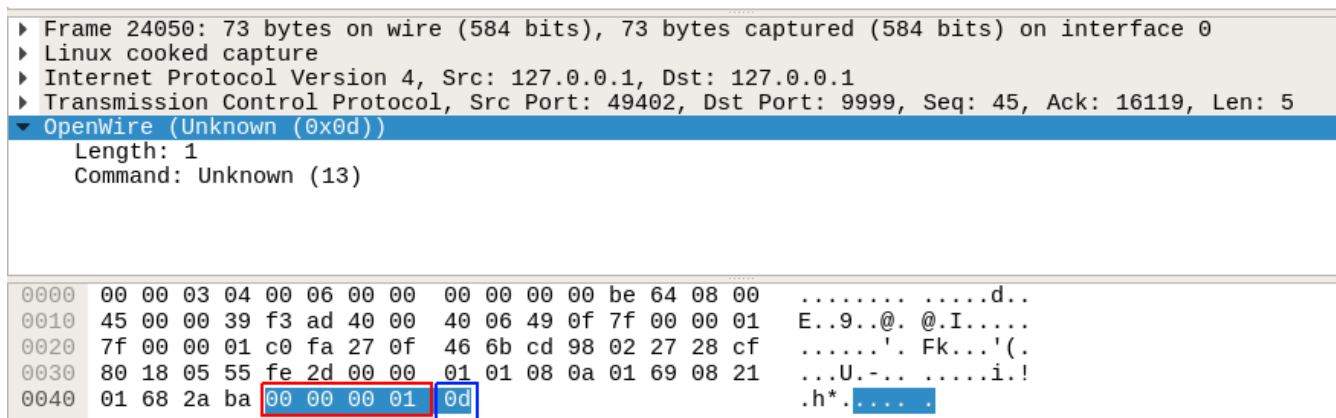


Figura 14: Mensaje con código 13

El servidor responde con un pokémon aleatorio, en este caso **Flareon**, y le da 5 intentos al cliente para capturar el pokémon.

```

▶ Frame 24051: 81 bytes on wire (648 bits), 81 bytes captured (648 bits) on interface 0
▶ Linux cooked capture
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▶ Transmission Control Protocol, Src Port: 9999, Dst Port: 49402, Seq: 16119, Ack: 50, Len: 13
▼ OpenWire (MessageAck)
  Length: 9
  Command: MessageAck (22)
  Command Id: 88501345
  Command response required: 114
  ▶ Destination: LocalTransactionId

```

0000	00 00 03 04 00 06 00 00 00 00 00 00 80 e8 08 00	.....
0010	45 00 00 41 d7 17 40 00 40 06 65 9d 7f 00 00 01	E..A..@. @.e....
0020	7f 00 00 01 27 0f c0 fa 02 27 28 cf 46 6b cd 9d	....'... .'(.Fk..
0030	80 18 01 56 fe 35 00 00 01 01 08 0a 01 69 08 22	...V.5.. ....i."
0040	01 69 08 21 00 00 00 09 16 05 46 6c 61 72 65 6f	.i.!.... ..Flareo
0050	6e	n

Figura 15: Mensaje con código 22

El cliente lanza su primer pokebola.

```

▶ Frame 24061: 81 bytes on wire (648 bits), 81 bytes captured (648 bits) on interface 0
▶ Linux cooked capture
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▶ Transmission Control Protocol, Src Port: 49402, Dst Port: 9999, Seq: 50, Ack: 16132, Len: 13
▼ OpenWire (FlushCommand)
  Length: 9
  Command: FlushCommand (15)
  Command Id: 88501345
  Command response required: 114
  ▶ [Expert Info (Note/Undecoded): OpenWire command fields unknown to Wireshark: 15]

```

0000	00 00 03 04 00 06 00 00 00 00 00 00 00 08 00	.....
0010	45 00 00 41 f3 af 40 00 40 06 49 05 7f 00 00 01	E..A..@. @.I....
0020	7f 00 00 01 c0 fa 27 0f 46 6b cd 9d 02 27 28 dc	....'... '(.Fk...'
0030	80 18 05 55 fe 35 00 00 01 01 08 0a 01 69 10 a7	...U.5.. ....i..
0040	01 69 08 22 00 00 00 09 0f 05 46 6c 61 72 65 6f	.i.".... ..Flareo
0050	6e	n

Figura 16: Mensaje con código 15; intento #5

El servidor indica que falló el intento y decrementa el contador.

```

▶ Frame 24062: 81 bytes on wire (648 bits), 81 bytes captured (648 bits) on interface 0
▶ Linux cooked capture
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▶ Transmission Control Protocol, Src Port: 9999, Dst Port: 49402, Seq: 16132, Ack: 63, Len: 13
▼ OpenWire (ActiveMQMessage)
  Length: 9
  Command: ActiveMQMessage (23)
  Command Id: 71724129
  Command response required: 114
  ▶ Object: ActiveMQMessage

```

0000	00 00 03 04 00 06 00 00 00 00 00 00 00 08 00	.....
0010	45 00 00 41 d7 18 40 00 40 06 65 9c 7f 00 00 01	E..A..@. @.e....
0020	7f 00 00 01 27 0f c0 fa 02 27 28 dc 46 6b cd aa	....'... .'(.Fk..
0030	80 18 01 56 fe 35 00 00 01 01 08 0a 01 69 10 a7	...V.5.. ....i..
0040	01 69 10 a7 00 00 00 09 17 04 46 6c 61 72 65 6f	.i.. .... ..Flareo
0050	6e	n

Figura 17: Mensaje con código 23; 4 intentos restantes



El cliente lanza otra pokebola.

```
▶ Frame 24065: 81 bytes on wire (648 bits), 81 bytes captured (648 bits) on interface 0
▶ Linux cooked capture
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▶ Transmission Control Protocol, Src Port: 49402, Dst Port: 9999, Seq: 63, Ack: 16145, Len: 13
▼ OpenWire (FlushCommand)
  Length: 9
  Command: FlushCommand (15)
  Command Id: 71724129
  Command response required: 114
▶ [Expert Info (Note/Undecoded): OpenWire command fields unknown to Wireshark: 15]
```

0000	00 00 03 04 00 06 00 00	00 00 00 00 08 00 08 00	.....
0010	45 00 00 41 f3 b1 40 00	40 06 49 03 7f 00 00 01	E..A..@. @.I.....
0020	7f 00 00 01 c0 fa 27 0f	46 6b cd aa 02 27 28 e9	.....'. Fk...'.(.
0030	80 18 05 55 fe 35 00 00	01 01 08 0a 01 69 12 9c	...U.5.. ....i..
0040	01 69 10 a7 00 00 00 09	0f 04 46 6c 61 72 65 6f	.i..... ..Flareo
0050	6e		n

Figura 18: Mensaje con código 15; intento #4

Se indica que de nuevo falló el lanzamiento, decrementa el contador.

```
▶ Frame 24066: 81 bytes on wire (648 bits), 81 bytes captured (648 bits) on interface 0
▶ Linux cooked capture
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▶ Transmission Control Protocol, Src Port: 9999, Dst Port: 49402, Seq: 16145, Ack: 76, Len: 13
▼ OpenWire (ActiveMQMessage)
  Length: 9
  Command: ActiveMQMessage (23)
  Command Id: 54946913
  Command response required: 114
▶ Object: ActiveMQMessage
```

0000	00 00 03 04 00 06 00 00	00 00 00 00 68 6f 08 00	..... ..ho..
0010	45 00 00 41 d7 19 40 00	40 06 65 9b 7f 00 00 01	E..A..@. @.e.....
0020	7f 00 00 01 27 0f c0 fa	02 27 28 e9 46 6b cd b7	....'... '(.Fk..
0030	80 18 01 56 fe 35 00 00	01 01 08 0a 01 69 12 9c	...V.5.. ....i..
0040	01 69 12 9c 00 00 00 09	17 03 46 6c 61 72 65 6f	.i..... ..Flareo
0050	6e		n

Figura 19: Mensaje con código 23; 3 intentos restantes

El cliente hace otro intento.

```
▶ Frame 24068: 81 bytes on wire (648 bits), 81 bytes captured (648 bits) on interface 0
▶ Linux cooked capture
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▶ Transmission Control Protocol, Src Port: 49402, Dst Port: 9999, Seq: 76, Ack: 16158, Len: 13
▼ OpenWire (FlushCommand)
  Length: 9
  Command: FlushCommand (15)
  Command Id: 54946913
  Command response required: 114
▶ [Expert Info (Note/Undecoded): OpenWire command fields unknown to Wireshark: 15]
```

0000	00 00 03 04 00 06 00 00	00 00 00 00 b5 16 08 00	.....
0010	45 00 00 41 f3 b3 40 00	40 06 49 01 7f 00 00 01	E..A..@. @.I.....
0020	7f 00 00 01 c0 fa 27 0f	46 6b cd b7 02 27 28 f6	.....'. Fk...'.(.
0030	80 18 05 55 fe 35 00 00	01 01 08 0a 01 69 13 e9	...U.5.. ....i..
0040	01 69 12 9c 00 00 00 09	0f 03 46 6c 61 72 65 6f	.i..... ..Flareo
0050	6e		n

Figura 20: Mensaje con código 15; intento #3

El servidor indica que esta vez sí se logró capturar el pokémon, y manda la imagen asociada a éste.

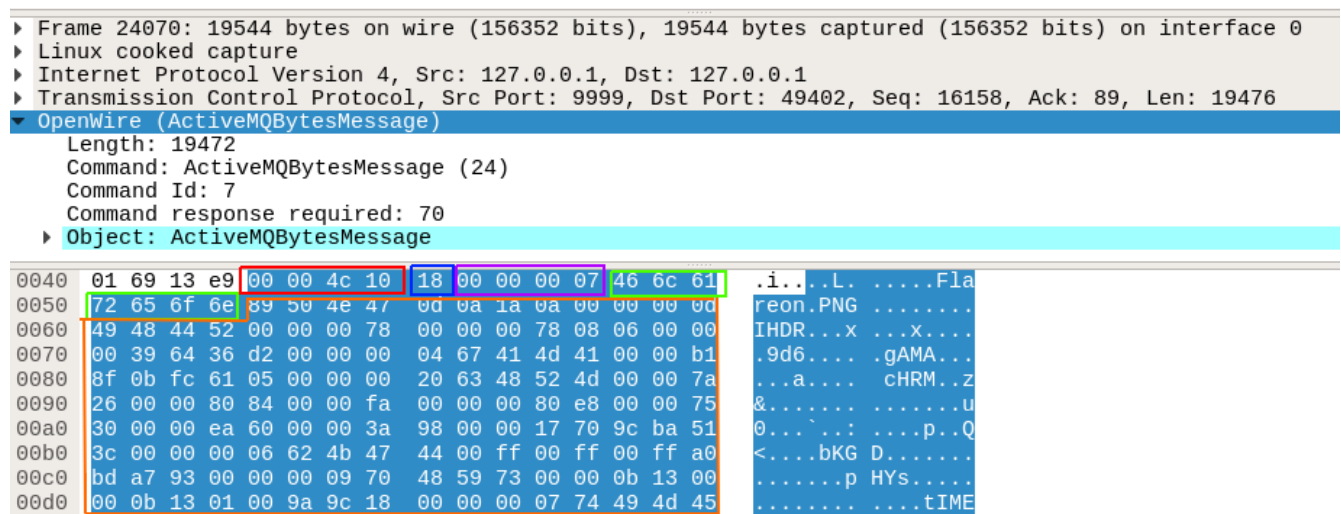


Figura 21: Mensaje con código 24

A continuación se muestra un caso en el que el cliente escoge ignorar al pokémon en vez de lanzar una pokebola.

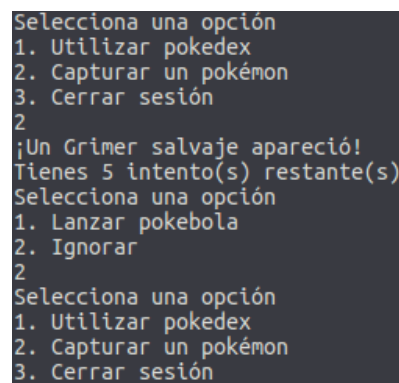


Figura 22: Se ignora al pokémon Grimer

El cliente indica que quiere capturar un pokémon

```

▶ Frame 37652: 73 bytes on wire (584 bits), 73 bytes captured (584 bits) on interface 0
▶ Linux cooked capture
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▶ Transmission Control Protocol, Src Port: 49402, Dst Port: 9999, Seq: 89, Ack: 35634, Len: 5
▼ OpenWire (Unknown (0x0d))
  Length: 1
  Command: Unknown (13)

```

0000	00 00 03 04 00 06 00 00	00 00 00 00 8a 9c 08 00	.....
0010	45 00 00 39 f3 b5 40 00	40 06 49 07 7f 00 00 01	E..9..@. @.I.....
0020	7f 00 00 01 c0 fa 27 0f	46 6b cd c4 02 27 75 0a	.....'. Fk...'u.
0030	80 18 09 54 fe 2d 00 00	01 01 08 0a 01 69 ce 8f	...T.-... ..i..
0040	01 69 14 22 00 00 00 01	0d	.i.".....

Figura 23: Mensaje con código 13

El servidor regresa al pokémon aleatorio Grimer.

```

▶ Frame 37654: 80 bytes on wire (640 bits), 80 bytes captured (640 bits) on interface 0
▶ Linux cooked capture
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▶ Transmission Control Protocol, Src Port: 9999, Dst Port: 49402, Seq: 35634, Ack: 94, Len: 12
▼ OpenWire (MessageAck)
  Length: 8
  Command: MessageAck (22)
  Command Id: 88568425
  Command response required: 109

```

0000	00 00 03 04 00 06 00 00	00 00 00 00 a1 5a 08 00	.....Z..
0010	45 00 00 40 d7 1d 40 00	40 06 65 98 7f 00 00 01	E..@..@. @.e.....
0020	7f 00 00 01 27 0f c0 fa	02 27 75 0a 46 6b cd c9	....'... 'u.Fk..
0030	80 18 01 56 fe 34 00 00	01 01 08 0a 01 69 ce 90	...V.4.. ....i..
0040	01 69 ce 8f 00 00 00 08	16 05 47 72 69 6d 65 72	.i.....Grimer

Figura 24: Mensaje con código 22

El cliente indica que no quiere capturar a tal pokémon.

```

▶ Frame 37674: 73 bytes on wire (584 bits), 73 bytes captured (584 bits) on interface 0
▶ Linux cooked capture
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▶ Transmission Control Protocol, Src Port: 49402, Dst Port: 9999, Seq: 94, Ack: 35646, Len: 5
▼ OpenWire (ControlCommand)
  Length: 1
  Command: ControlCommand (14)

```

0000	00 00 03 04 00 06 00 00	00 00 00 00 64 00 08 00	.....d...
0010	45 00 00 39 f3 b7 40 00	40 06 49 05 7f 00 00 01	E..9..@. @.I.....
0020	7f 00 00 01 c0 fa 27 0f	46 6b cd c9 02 27 75 16	.....'. Fk...'u.
0030	80 18 09 54 fe 2d 00 00	01 01 08 0a 01 69 d3 37	...T.-... ..i.7
0040	01 69 ce 90 00 00 00 01	0e	.i.....

Figura 25: Mensaje con código 14

En el siguiente caso el cliente falla en el último intento de captura del pokémon Magnemite.

```

Fallaste en capturar al Magnemite salvaje
Tienes 1 intento(s) restante(s)
Selecciona una opción
1. Lanzar pokebola
2. Ignorar
1
!El pokémon se escapó!
Selecciona una opción
1. Utilizar pokedex
2. Capturar un pokémon
3. Cerrar sesión

```

Figura 26

El cliente realiza el último intento para capturar a Magnemite.

► Frame 38157: 83 bytes on wire (664 bits), 83 bytes captured (664 bits) on interface 0  
 ► Linux cooked capture  
 ► Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1  
 ► Transmission Control Protocol, Src Port: 9999, Dst Port: 49402, Seq: 353, Ack: 128015, Len: 15

▼ OpenWire (FlushCommand)  
 Length: 11  
 Command: FlushCommand (15)  
 Command Id: 21848423  
 Command response required: 110  
 ► [Expert Info (Note/Undecoded): OpenWire command fields unknown to Wireshark: 15]

0000	00 00 03 04 00 06 00 00 00 00 00 00 00 06 08 00	.....
0010	45 00 00 43 f3 e6 40 00 40 06 48 cc 7f 00 00 01	E..C..@. @.H....
0020	7f 00 00 01 c0 fa 27 0f 46 6b ce cc 02 28 dd e7	.....'. Fk...(..
0030	80 18 0e 35 fe 37 00 00 01 01 08 0a 01 6a 5a 14	...5.7.. .....jZ.
0040	01 6a 59 67 00 00 00 0b 0f 01 4d 61 67 6e 65 6d	.jYg.... ..Magnem
0050	69 74 65	ite

Figura 27: Mensaje con código 15; intento #1

El servidor contesta con el código de error 41, indicando que el último intento de captura del cliente falló y ya no le quedan más intentos.

► Frame 38158: 73 bytes on wire (584 bits), 73 bytes captured (584 bits) on interface 0  
 ► Linux cooked capture  
 ► Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1  
 ► Transmission Control Protocol, Src Port: 9999, Dst Port: 49402, Seq: 128015, Ack: 368, Len: 5

▼ OpenWire (Unknown (0x29))  
 Length: 1  
 Command: Unknown (41)

0000	00 00 03 04 00 06 00 00 00 00 00 00 45 9a 08 00	..... ....E...
0010	45 00 00 39 d7 41 40 00 40 06 65 7b 7f 00 00 01	E..9.A@. @.e{....
0020	7f 00 00 01 27 0f c0 fa 02 28 dd e7 46 6b ce db	.....'... ..Fk..
0030	80 18 01 56 fe 2d 00 00 01 01 08 0a 01 6a 5a 14	...V.-... .....jZ.
0040	01 6a 5a 14 00 00 00 01 29	.jZ. .... )

Figura 28: Mensaje con código de error 41

#### 4.3.4. Cierre de sesión y conexión

```
Selecciona una opción
1. Utilizar pokedex
2. Capturar un pokémon
3. Cerrar sesión
3
Selecciona una opción
1. Iniciar sesión
2. Cerrar conexión
2
[chepe@chepe p02]$ _
```

Figura 29: El cliente cierra la sesión y se desconecta del servidor

El cliente manda un mensaje indicando que quiere cerrar sesión.

```
► Frame 52761: 73 bytes on wire (584 bits), 73 bytes captured (584 bits) on interface 0
► Linux cooked capture
► Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
► Transmission Control Protocol, Src Port: 49402, Dst Port: 9999, Seq: 368, Ack: 128020, Len: 5
▼ OpenWire (ShutdownInfo)
  Length: 1
  Command: ShutdownInfo (11)
```

0000	00 00 03 04 00 06 00 00	00 00 00 00 91 53 08 00	.....S..
0010	45 00 00 39 f3 e8 40 00	40 06 48 d4 7f 00 00 01	E..9..@. @.H....
0020	7f 00 00 01 c0 fa 27 0f	46 6b ce db 02 28 dd ec	.....'. Fk...(..
0030	80 18 0e 35 fe 2d 00 00	01 01 08 0a 01 6a cf ab	...5.-.. ....j..
0040	01 6a 5a 14 00 00 00 01	0b	.jZ.....

Figura 30: Mensaje con código 11

Posteriormente indica que también quiere cerrar la conexión con el servidor.

```
► Frame 52787: 73 bytes on wire (584 bits), 73 bytes captured (584 bits) on interface 0
► Linux cooked capture
► Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
► Transmission Control Protocol, Src Port: 49402, Dst Port: 9999, Seq: 373, Ack: 128020, Len: 5
▼ OpenWire (Unknown (0x00))
  Length: 1
  Command: Unknown (0)
```

0000	00 00 03 04 00 06 00 00	00 00 00 00 6d 06 08 00	.....m...
0010	45 00 00 39 f3 e9 40 00	40 06 48 d3 7f 00 00 01	E..9..@. @.H....
0020	7f 00 00 01 c0 fa 27 0f	46 6b ce e0 02 28 dd ec	.....'. Fk...(..
0030	80 18 0e 35 fe 2d 00 00	01 01 08 0a 01 6a d7 f2	...5.-.. ....j..
0040	01 6a cf b5 00 00 00 01	00	.j.....

Figura 31: Mensaje con código 0