Proyecto 2. Creación e implementación de un protocolo de la Capa de Aplicación

José Nieves Morán Silva
Juan Antonio López Rivera

16 de diciembre de 2018

1 Objetivo

Pokemon es usa serie de juegos altamente popular, generando billones con sus juegos y mercancía al año. Asimismo tiene la ventaja de necesitar cambios mínimos a la fórmula establecida para cada entrada de la franquicia, por lo que diseñar un nuevo juego es pan comido!

Presentamos aquí el núcleo de una implementación del protocolo web del próximo lanzamiento en la franquicia: Pokemon Let's Go Bidoof! ¿Cansado de atrapar siempre a los mismos pokemones OP año tras año? ¡Obtén una experiencia refrescante capturando pokemones ~~ridículos~~ menos comunes como Ducklett, Klefki y Magikarp!

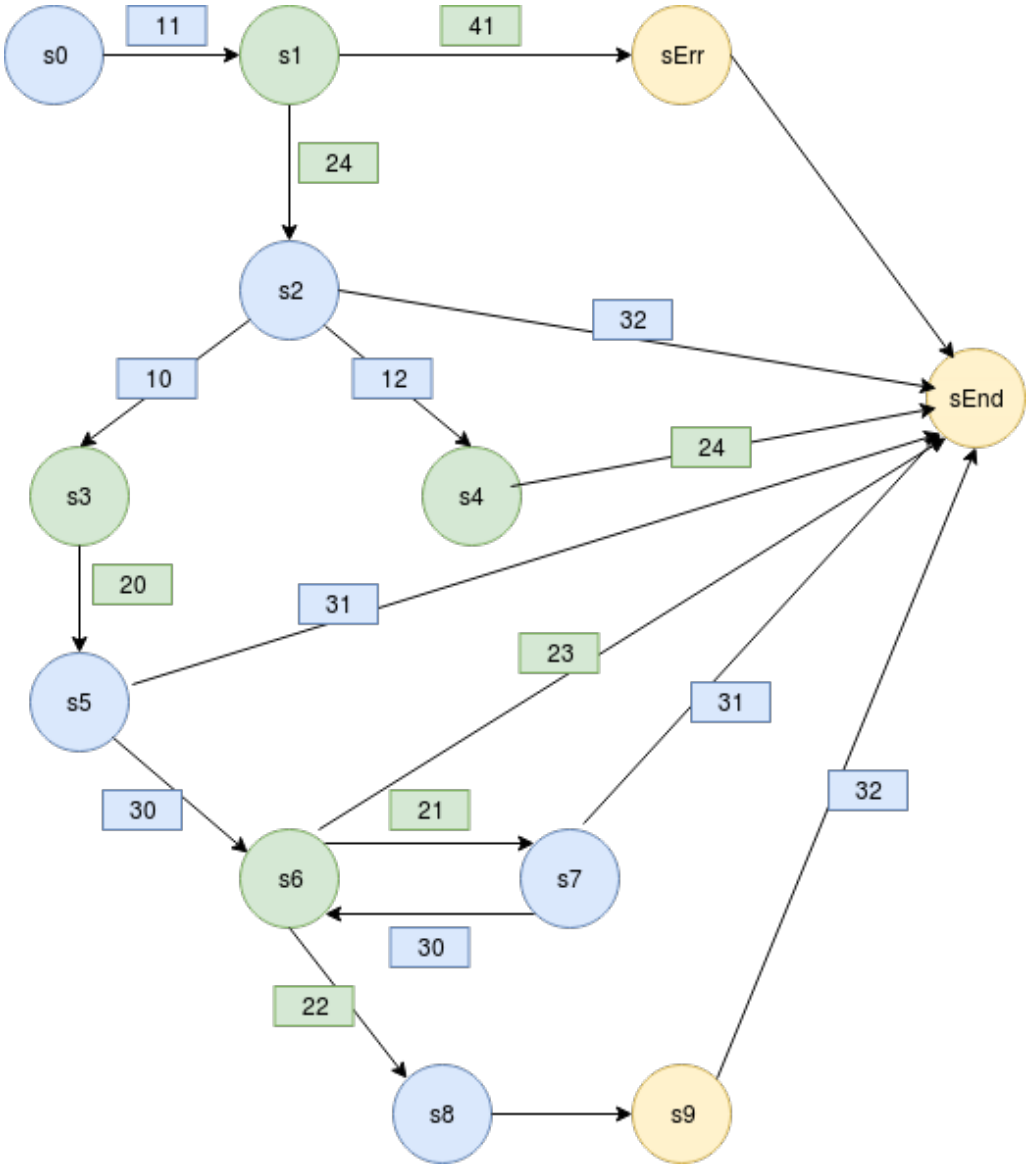
A pesar de que no se implementan batallas pokemon, interacción con otros entrenadores, o siquiera un modo historia sencillo, ésto no impidió que Pokemon Go! fuera masivamente popular (durante unas 2 semanas) y creemos que tampoco lo impedirá ahora. ¿Qué funcionalidades sí tiene? ¡Conexiones concurrentes de usuarios con un servidor web, el cual cuenta con una base de datos con una Pokedex de los pokemones de la región; asimismo almacena en ella perfiles de usuario y sus pokemones capturados!

2 Diseño del protocolo

2.1 Tabla de Estados

Nombre	Descripción
s0	Estado inicial.
s1	Validando usuario.
s2	Esperando selección de opción (capturar pokemon, ver pokemones capturados o salir).
s3	Obteniendo pokemon aleatorio.
s4	Obteniendo lista de pokemones capturados.
s5	Esperando que el usuario indique si se quiere capturar el pokemon o no.
s6	Inicia con N=5. Se selecciona aleatoriamente una de las siguientes: o bien se capturó exitosamente el pokemon, o falló la captura y N = N - 1.
s7	Esperando que el usuario indique si desea lanzar otra pokebola.
s8	Se recibe imagen de pokemon capturado.
s9	Cerrando conexión.
sEnd	Estado final, terminando la sesión.
sErr	Estado de error, terminando la sesión.

2.2 Diagrama de la Máquina de Estados Finita (FSM)



Los estados de color verde corresponden al servidor, y los azules son del cliente. Asimismo los recuadros son mensajes enviados con el código indicado, los verdes son mensajes del servidor y los azules del cliente. Los estados amarillos son mutuos.

2.3 Códigos de mensajes

Código	Descripción	Composición
10	Solicitar un pokemon para capturar.	Número de código (1 byte).
11	Solicitar validación de usuario.	Número de código (1 byte), ID de usuario (1 byte).
12	Solicitar lista de pokemones capturados.	Número de código (1 byte).
20	Enviar pokemon seleccionado, preguntar ¿Desea capturar?	Número de código (1 byte), Nombre Pokemon (N bytes).
21	Captura fallida pero aún quedan intentos. ¿Intentar de nuevo?	Número de código (1 byte), ID Pokemon (1 byte), Intentos restantes (1 byte).
22	Captura exitosa, enviando imagen.	Número de código (1 byte), Tamaño imagen (4 bytes), Imagen (N bytes).
23	Intentos de captura agotados.	Número de código (1 byte).
24	Enviando lista de pokemones capturados, serializada como cadena separada por comas.	Número de código (1 byte), Pokemones capturados (N bytes).
25	ID de usuario autenticado exitosamente.	Número de código (1 byte), Nombre usuario (N bytes).
30	Sí.	Número de código (1 byte).
31	No.	Número de código (1 byte).
32	Terminando sesión.	Número de código (1 byte).
40	Error no especificado.	Número de código (1 byte).
41	Error al autenticar usuario.	Número de código (1 byte).

3 Implementación

Se puede descargar todo el proyecto en la siguiente URL:

```
https://github.com/stunnedbud/bidoof
```

Escogimos programarla en Python 2.7, con SQLite 3 como motor de la base de datos. No requiere compilación, ejecutar la aplicación es extremadamente sencillo, y la base de datos se almacena toda en un archivo de SQLite. Por éstos motivos consideramos redundante la adición de un archivo Makefile.

Para iniciar la ejecución del servidor simplemente hay que ejecutar el siguiente comando (en la carpeta base del proyecto):

```
> python serverPokemon.py
```

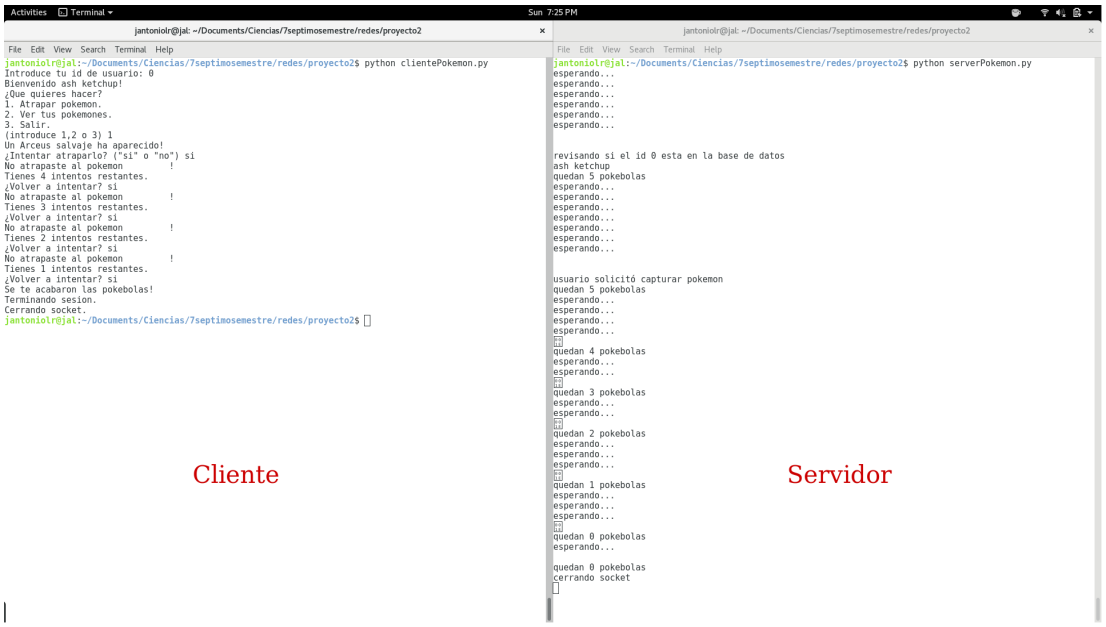
Éste escucha en localhost en el puerto 9999, como estaba indicado. Se puede terminar su ejecución presionando las teclas Ctrl + C.

Análogamente, para iniciar un cliente:

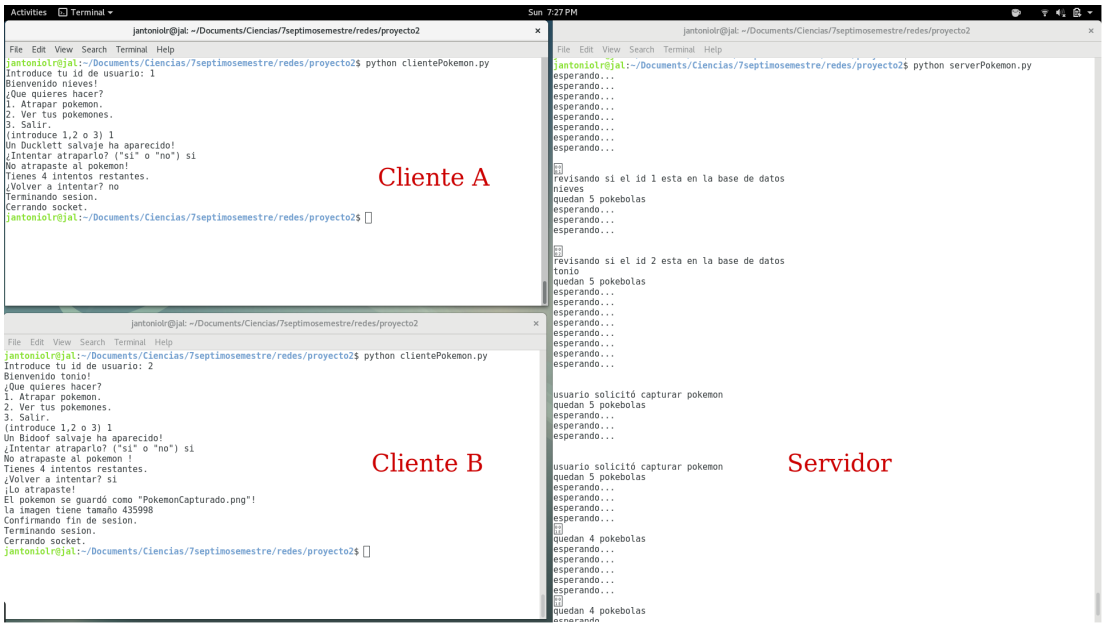
```
> python clientePokemon.py
```

La base de datos se encuentra ya montada y lista en el archivo *pokedex.db*. Existen en ella los usuarios con id 0, 1 y 2, para pruebas. Asimismo se incluye el archivo *setupDB.py* el cual puede ejecutarse para crear de nuevo la base e inicializarla con datos de ejemplo.

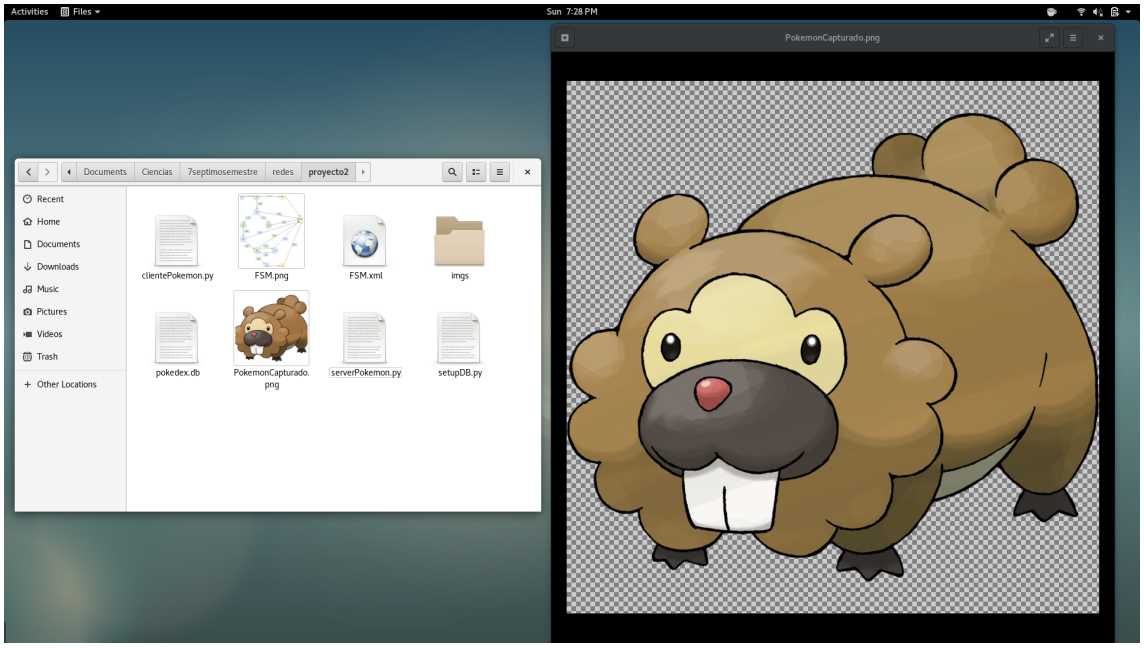
A continuación se incluyen capturas de pantalla que demuestran el funcionamiento correcto de la aplicación.



Intento de captura (fallido) de un Arceus por el usuario con id 0.



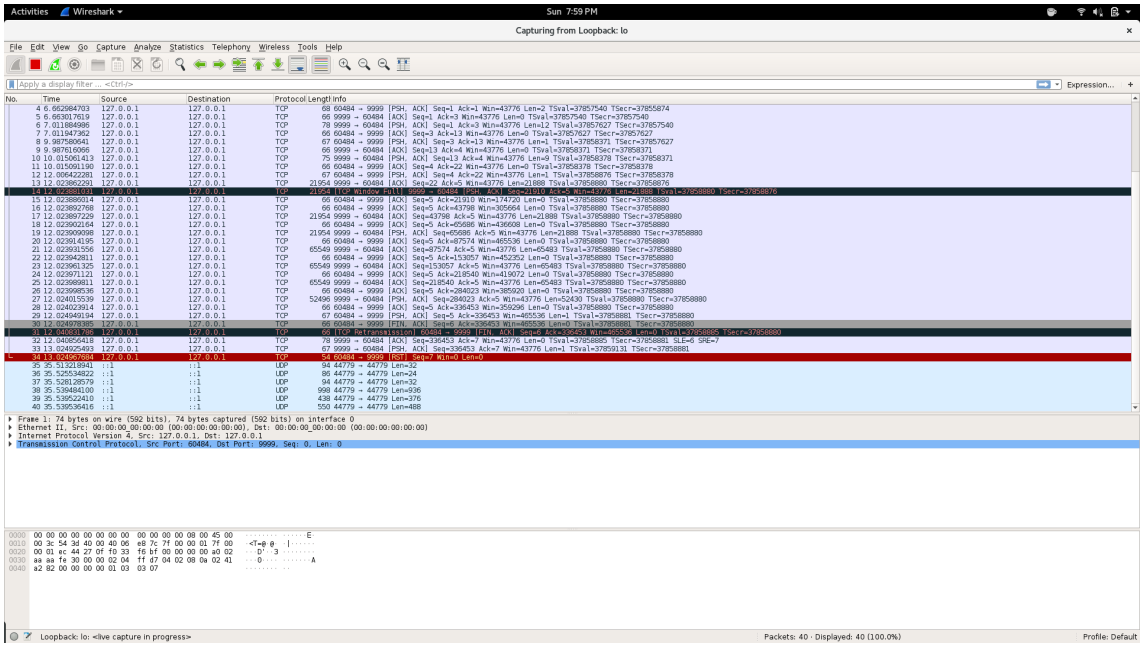
Funcionamiento concurrente por medio de hilos de ejecución en el servidor.



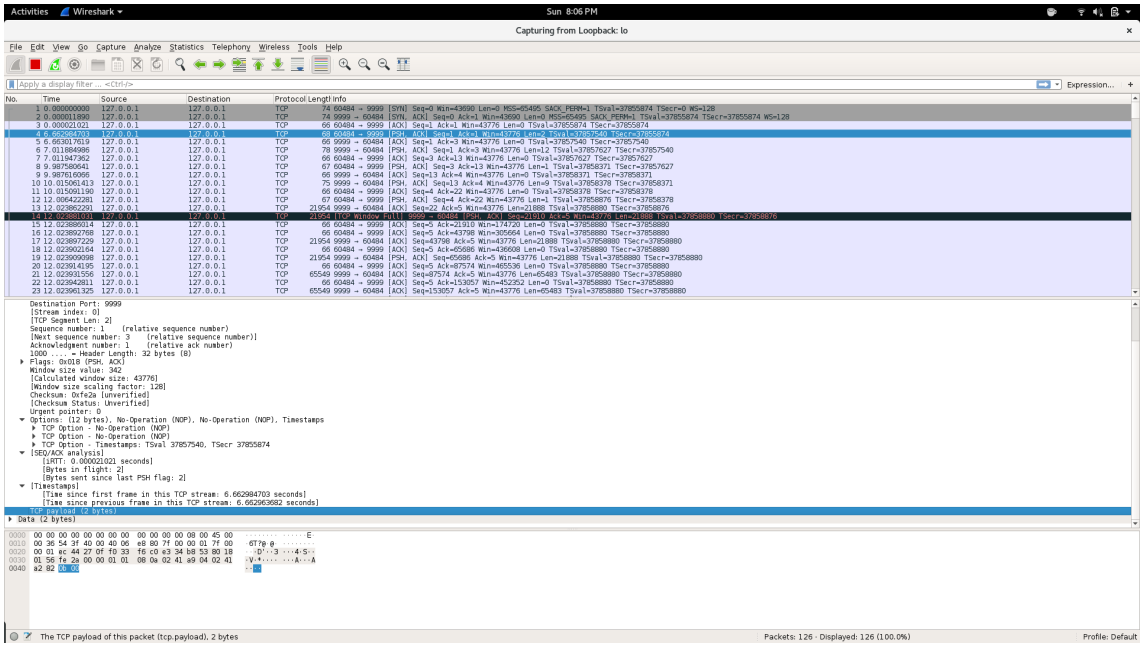
Envío correcto de imagen del pokemon, capturado por el Cliente B en la imagen previa.

```
jantoniolr@jal:~/Documents/Ciencias/7septimosemestre/redes/proyecto2$ python clientePokemon.py
Introduce tu id de usuario: 2
Bienvenido tonio!
¿Que quieres hacer?
1. Atrapar pokemon.
2. Ver tus pokemones.
3. Salir.
(introduce 1,2 o 3) 2
Pokemones capturados: Pidgey, Smeargle, Klefki, Rattata, Bidoof
Terminando sesion.
Cerrando socket.
jantoniolr@jal:~/Documents/Ciencias/7septimosemestre/redes/proyecto2$
```

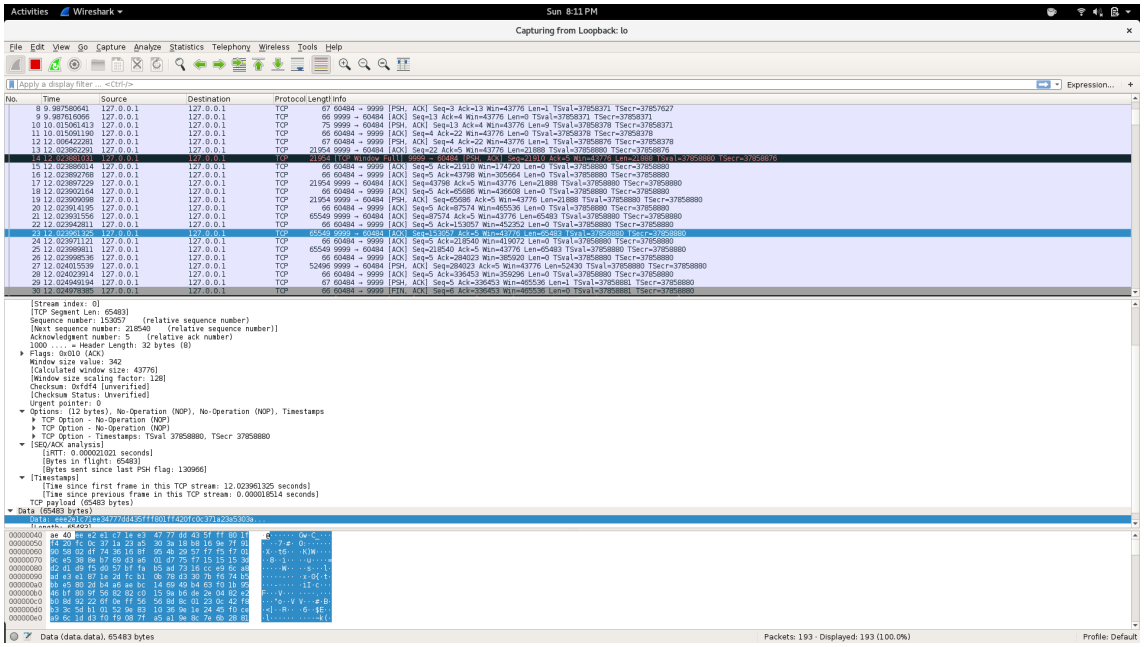
Actualización correcta de pokemones capturados.



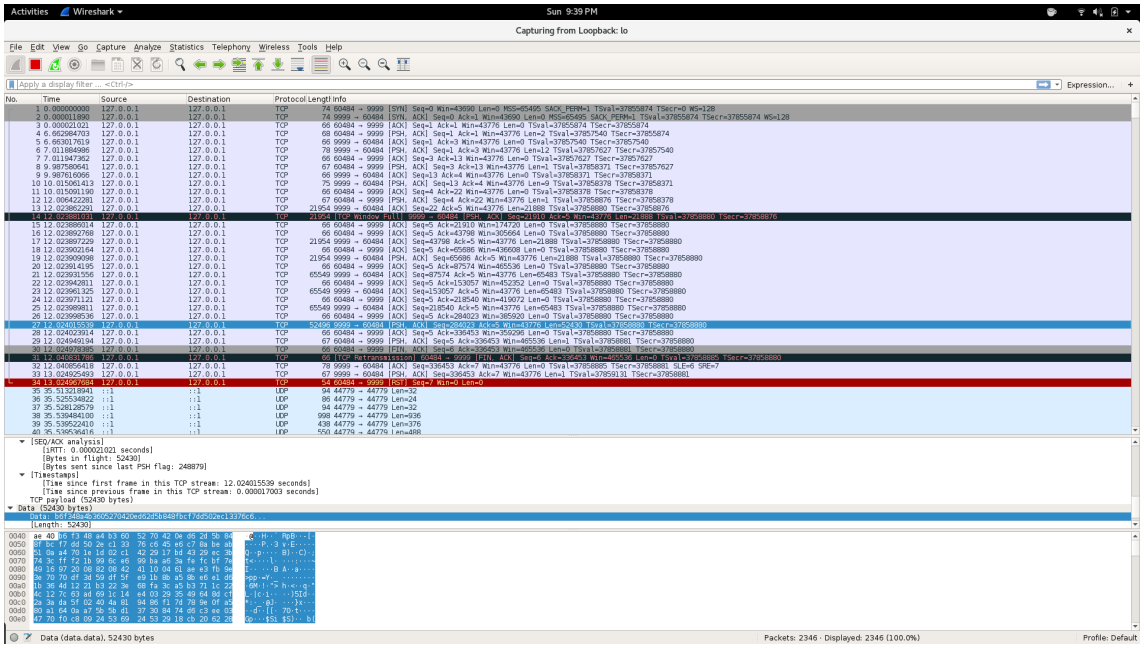
Paquetes enviados en loopback (lo) durante una captura de pokemon (del frame 1 al 34).



Ejemplo de un mensaje del cliente. Nótese hasta abajo que el TCP Payload se encuentra seleccionado y consta de 2 bytes. Ésto es porque éste mensaje corresponde al código 11 (solicitar validación de usuario); se envió el código de mensaje en el primer byte y el id de usuario en el segundo.



Ejemplo de un mensaje del servidor. Éste corresponde al código 22, donde se capturó un pokemon exitosamente y se envía la imagen. Del TCP Payload el primer byte corresponde al código de mensaje, los siguientes 4 al tamaño de la imagen, y el resto es el inicio de ésta. La imagen se mandó por partes en varios paquetes (frames 21, 23, 25 y 27).



Último paquete que contiene la imagen enviada. En éste caso todos los bytes del payload corresponden al fragmento final de la imagen del pokemon.

4 Documentación de funciones

4.1 Cliente

- Nombre: procesar_respuesta
  - Parámetros: respuesta (cadena), socket
  - Descripción: Implementación de los estados de la FSM correspondientes al cliente. Recibe una cadena de texto la cual es una respuesta del servidor, realiza las acciones correspondientes al estado actual y devuelve una cadena con la respuesta a enviar al servidor.

Es la única función de la implementación del cliente. Todo lo demás se realiza en el cuerpo principal del código.

4.2 Servidor

- Nombre: manejar_conexion
  - Parámetros: conexión (objeto connection dado por sock.accept())
  - Descripción: Una vez que se acepta una conexión entrante de un cliente, se inicia un nuevo hilo de ejecución corriendo ésta función. Se encarga de inicializar las variables de la sesión, esperar un mensaje entrante, y cerrar limpiamente la conexión una vez que se finalice la sesión. Asimismo lleva el conteo para hacer timeout después de 15 segundos.
- Nombre: procesar_datos
  - Parámetros: conexión, data (cadena de texto), vars (lista variables de sesión)
  - Descripción: Implementa los estados de la FSM correspondientes al servidor. Recibe una cadena de texto la cual es un mensaje del cliente, realiza las acciones correspondientes al estado actual y devuelve una cadena con la respuesta a enviar al cliente.
- Nombre: create_connection
  - Parámetros: db_file (nombre de archivo de la base de datos)
  - Descripción: Inicializa una conexión SQLite en la base de datos especificada.
- Nombre: execute_query
  - Parámetros: conn (conexión SQLite), sql_query (consulta a ejecutar)
  - Descripción: Ejecuta una operación (que no sea INSERT o UPDATE) en la base de datos.
- Nombre: seleccionar_pokemon_aleatorio
  - Parámetros: Ninguno
  - Descripción: Selecciona un Pokemon aleatoriamente entre todos los que se encuentran en la tabla "pokemones" de la base de datos.
- Nombre: obtener_nombre_pokemon
  - Parámetros: id_pokemon
  - Descripción: Devuelve el nombre del pokemon con id especificada.
- Nombre: obtener_nombre_usuario
  - Parámetros: id_usuario

  - Descripción: Devuelve el nombre del usuario con id especificada.

- Nombre: insertar_captura_pokemon
  - Parámetros: id_usuario, id_pokemon
  - Descripción: Actualiza la tabla de pokemones capturados. Si el usuario no ha capturado ese pokemon aún, agrega el renglón a la tabla. Si ya lo habia capturado, agrega 1 al contador de pokemones de esa especie capturados (la columna "cantidad").

- Nombre: obtener_pokemones_capturados
  - Parámetros: id_usuario
  - Descripción: Devuelve una lista con los nombres de pokemones capturados por el usuario dado.