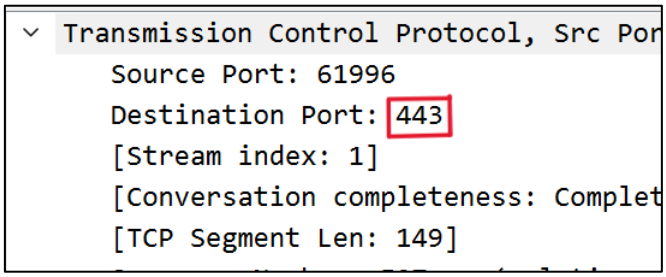
Práctica 6

Alumno: Alcausa Luque, Juan Carlos

Titulación: Grado de Ingeniería Informática + Matemáticas

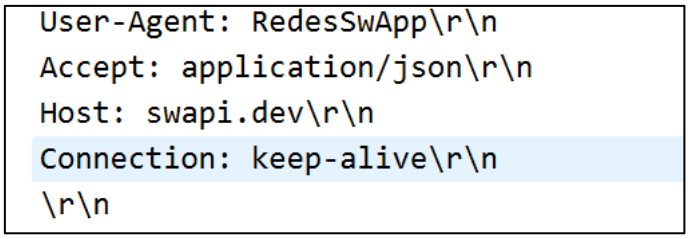
PC de la práctica

Ejercicio 1. ¿Cuál es el puerto utilizado por el servidor? ¿Es el normal de HTTP (80)? ¿Por qué?
Usa el 443 que es el puerto de HTTPS, el puerto por defecto para la versión segura de HTTP.



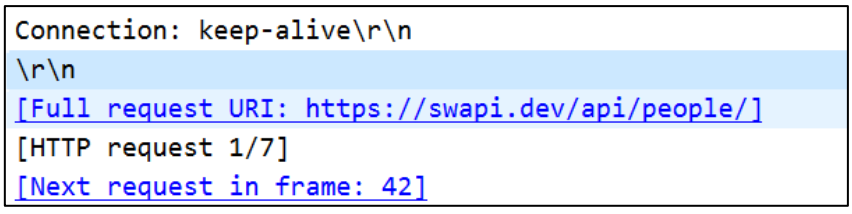
Ejercicio 2. Observe el número de conexiones realizadas. ¿Cuántas hace? ¿Usa una conexión permanente (en la misma conexión hace varias peticiones) o no permanente (solo realiza una por conexión)? En caso de ser permanente, ¿qué cabecera de la petición indica que queremos que sea permanente?

Se realiza una única conexión, en la trama 35 por ejemplo, podemos ver cómo en el campo de HTTP Connection se especifica keep-alive.



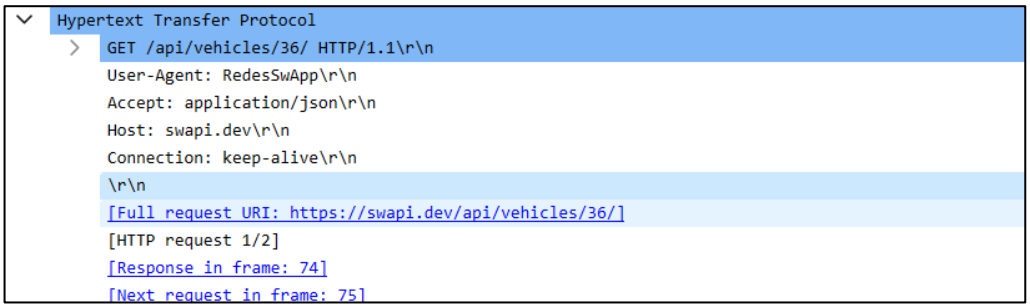
Ejercicio 3. Observe una respuesta, ¿cómo se identifica dónde acaban las cabeceras HTTP y empieza el recurso?

Hay una línea en blanco entre el final de las cabeceras y el inicio del recurso:



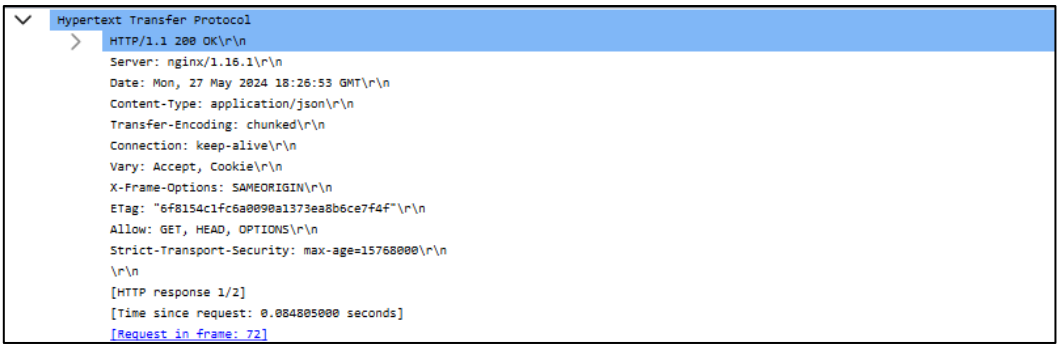
Ejercicio 4. Describa el significado de las cabeceras de una petición y una respuesta (sin incluir las que empiecen por `X-`).
Veamos la petición 72:
Tenemos la primera línea que es la línea de estado, nos informa de que el método empleado en la petición es GET, la URL que se pide y qué versión de HTTP se está usando. Después se tienen cabeceras como Cache-Control que es en relación con la eficiencia. Connection nos indica

que la conexión va a cerrarse y el resto de las cabeceras con información adicional sobre la conexión como. Tras ello viene un salto de línea que es y después el mensaje.



Veamos su respuesta, la trama 74.

En su respuesta, la trama 335 tenemos también la línea de estado que nos indica el protocolo utilizado, el código de la respuesta y el mensaje de dicha respuesta. Después tenemos más cabeceras con información adicional como los rangos de bytes aceptados, la última modificación del recurso, el tipo o formato del contenido y de nuevo el salto de línea y el mensaje de la respuesta.



Práctica 7

Alumno: Alcausa Luque, Juan Carlos

Titulación: Grado en Ingeniería Informática + Matemáticas
PC de la práctica: portátil

Ejercicio 1. Explique cada una de las instrucciones enviadas por el cliente, indicando para qué se usa. ¿Cómo determina el servidor cuándo termina el cuerpo del correo?

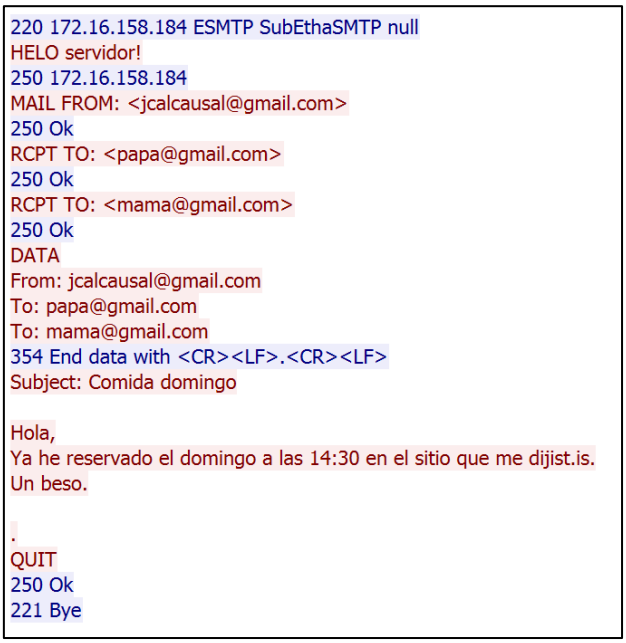
- HELO RySD.2.0.24: para iniciar la sesión y crear la conexión
- MAIL FROM:<profesor@rysd.es>: indica el remitente del correo.
- RCPT TO:<alumnado@rysd.es>: indica uno de los destinatarios.
- RCPT TO:<profesorado@rysd.es>: indica otro destinatario.
- RCPT TO:<root@rysd.es>: indica el último de los destinatarios.
- DATA: se utiliza para indicar el inicio de la transferencia del cuerpo del mensaje de correo.
- Se envía el mensaje
- Se envía un QUIT para finalizar.

Ejercicio 2. ¿Por qué el servidor tras el comando DATA envía un código de tipo 3xx?
Porque indica que lo ha recibido correctamente y está a la espera de más datos (del contenido de correo)

Ejercicio 3. ¿Por qué hay tres envíos de comando RCPT TO?
Porque hay tres destinatarios en la traza que estamos analizando, uno es alumnado@rysd.es, otro profesorado@rysd.es y el último root@rysd.es

Ejercicio 4. Si observa la imagen previa los destinatarios son enviados en los campos Para, CC (Carbon Copy) y BCC (Blind Carbon Copy). ¿Qué diferencia a nivel de comando SMTP y contenido del correo en sí mismo tiene que un destinatario sea indicado en un campo u otro?
En cuanto a comando todos son RCPT, que indica los receptores, es después de enviarse DATA cuando se indica qué tipo queremos que sea cada uno.

Ejercicio 5. Use la opción **Follow TCP Stream** de Wireshark para observar el diálogo completo que han mantenido el cliente de correo y el servidor. Adjunte una captura de pantalla donde se observe dicho diálogo.



Ejercicio 6. ¿En cuales mensajes se usa *piggybacking*? ¿Por qué? En los que no usen esa estrategia, los mensajes de datos ¿confirman algo? ¿El qué?

Se usa en todos menos en los de datos, porque mientras se están enviando las líneas el servidor sólo responde con ACK, por lo que no hay nuevos datos que confirmar del servidor y el número de ACK permanece constante en estos mensajes en 88

Ejercicio 7. Si observa el interfaz gráfico de FakeSMTP, verá que este correo lo ha recibido varias veces. ¿Por qué?

Porque aparece uno por cada destinatario.

Ejercicio 8. Sabría indicar (quizás mediante un uso "inteligente" del cliente desarrollado), si el servidor FakeSMTP es iterativo o concurrente. Justifique la respuesta y añada capturas de pantalla para apoyar su contestación.

Es un servidor concurrente ya que permite establecer varias conexiones a la vez. Si ejecutamos varias veces Main podemos tener varios procesos activos al mismo tiempo.

DESCRIPCIÓN DEL CÓDIGO:

```
import java.util.ArrayList;
import java.util.List;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.Socket;

public class Main {

    // Socket para las comunicaciones
    static Socket socket = null;
    // Streams para el envío y recepción
    static InputStream in = null;
    static OutputStream out = null;

    // Esta función se conecta al servidor de SMTP
    static void conectar(){
      try {
            // TODO: conecte el socket al servidor de correo
    (máquina local en puerto 25)
            socket = new Socket ("127.0.0.1", 25);
            // Se abren los flujos de entrada y salida
            in = socket.getInputStream();
            out = socket.getOutputStream();
      } catch (IOException e) {
            System.out.println("Error al conectar al servidor: "
    + e.getMessage());
            System.exit(-1);
      }
    }

    // Esta función cierra el socket
```

```
static void desconectar(){
  try {
        // TODO: Cierre el socket y los flujos asociados
        socket.close();
        in.close();
        out.close();
  } catch (IOException e) {
        System.out.println("Error al cerrar la conexión con
el servidor: " + e.getMessage());
        System.exit(-1);
  }
}

// Envío de mensajes
static void enviar(String mensaje){
  // Mostramos por consola el mensaje a enviar
     System.out.println("C: " + mensaje);

     try {
        // TODO: Enviar usando write
        // TODO: Recordar en añadir en cada mensaje el \r\n
        mensaje = mensaje+"\r\n";
        // TODO: Convertir a array de bytes con getBytes (no
hace falta indicar el charset ni nada)
        out.write(mensaje.getBytes());

         // Usamos flush para forzar que el envio se haga en
este momento
         out.flush();
    } catch (IOException e) {
        System.out.println("Error al enviar: " +
e.getMessage());
        System.exit(-1);
  }
}

// Recepción de mensajes
static void recibir(){
     byte [] buffer = new byte[5000];
     try {
         // TODO: Recibir el mensaje con read
        in.read(buffer);
  } catch (IOException e) {
        System.out.println("Error al recibir: " +
e.getMessage());
        System.exit(-1);
  }
     // Convertimos el mensaje recibido a String ...
     String recv = new String(buffer,0,buffer.length);
```

```
        // ... eliminamos el \r\n final ...
        recv = recv.substring(0,recv.lastIndexOf('\r'));
        // ... y lo mostramos por pantalla
        System.out.println("S: "+ recv);
        // TODO: Validar si el código obtenido es correcto (1xx,
2xx, 3xx) o incorrecto (4xx, 5xx)
        if((int) recv.charAt(0) == 4 || (int) recv.charAt(0) ==
5){
           // TODO: Si es incorrecto enviar al servidor un RSET
(use el método enviar)
           enviar("RSET");
           // TODO: Envíe luego el comando QUIT
           enviar ("QUIT");
           // Desconectamos del servidor
           desconectar();
           System.exit(0);
        }
    }

    public static void main(String[] args) throws IOException {
      // Flujo de lectura de teclado:
      BufferedReader stdIn = new BufferedReader(new
InputStreamReader(System.in));

      // Nos conectamos al servidor
      System.out.print("Conectado al servidor...");
      conectar();
      System.out.println("conectado!");

      // ESQUEMA DEL PROTOCOLO SMTP

      // Recibimos el saludo inicial del servidor
      recibir();

      // Enviamos el HELO y recibimos su respuesta
      enviar("HELO servidor!");
      recibir();

      // Leemos el origen:
      String origen = "";
      while(origen.equals("")){
           System.out.println("Dime el correo del emisor: ");
           origen = stdIn.readLine();
      }
      // TODO: enviar origen del mensaje y recibir su respuesta
      enviar("MAIL FROM: <" + origen +  ">");
      recibir();

      // Ahora los destinos
```

```
  List<String> destinos = new ArrayList<String>();
  String destino = "";
  while(destinos.size() == 0 || !destino.equals("")){
        System.out.println("Dime el correo del destino (linea
en blanco para acabar): ");
        destino = stdIn.readLine();
        if(!destino.equals("")){
              destinos.add(destino);
              // TODO: enviar el destino del mensaje y su
recibir su respuesta
              enviar("RCPT TO: <" + destino + ">");
              recibir();
        }
  }

  // Ahora enviamos el correo: cabeceras + cuerpo
  // TODO: Enviamos el DATA y recibimos la respuesta
  enviar ("DATA");
  // Cabeceras:
  // TODO: Enviar la cabecera From: (no hay que recibir
respuesta)
  enviar ("From: " + origen);
  // TODO: Enviar las cabeceras To: (no hay que recibir
respuesta)
  for (String d : destinos) {
        enviar ("To: " + d);
  }

  // Leemos el asunto:
  String asunto = "";
  while(asunto.equals("")){
        System.out.println("Dime el asunto del correo: ");
        asunto = stdIn.readLine();
  }
  // TODO: enviar la cabecera Subject: (no hay que recibir
respuesta)
  enviar ("Subject: " + asunto);

  // Enviamos una línea en blanco para separar las cabeceras
del cuerpo
  enviar("");

  // Ahora el cuerpo que son muchas líneas
  String cuerpo = "";
  System.out.println("Dime el mensaje (linea en blanco para
acabar): ");
  do{
        cuerpo = stdIn.readLine();
        if(!cuerpo.equals("")){
```

```
                // TODO: enviar la línea (no hay que recibir
respuesta)
                enviar(cuerpo);
            }
        }while(!cuerpo.equals(""));
        // TODO: Enviar una línea en blanco y luego un punto para
acabar y recibir la respuesta
        enviar("\r\n.");
        recibir();

        // TODO: Enviar QUIT para acabar y recibir la respuesta
        enviar ("QUIT");
        recibir();

        // Nos desconectamos del servidor
        System.out.print("Desconectándonos del servidor...");
        desconectar();
        System.out.println("Desconectado!");
    }
```