1º parcial de Redes - ETSINF - 25 de enero de 2011

Apellidos, Nombre: **Solución**

Grupo de matrícula:_____

1. ***(0,5 puntos)*** a)¿Por qué se considera que el protocolo *telnet* no es seguro? b)¿Qué protocolo/aplicación ha reemplazado a *telnet* por su mayor seguridad?

a) Porque transmite todos los datos, y entre ellos el nombre de usuario y la contraseña, sin cifrar a través de la red.

b) SSH (*Secure Shell*)

2. ***(0,5 puntos)*** Indica el parámetro que hay pasarle al método *getByName* de la clase *InetAddress* y qué devuelve dicho método.

Se le pasa como parámetro un *string* conteniendo el nombre de dominio de un ordenador o una dirección IP.

El método devuelve la dirección IP correspondiente en formato *InetAddress* si ha podido resolverla o genera una excepción *UnknownHostException*.

3. ***(1 punto)*** ¿Cómo es posible enviar varios objetos MIME distintos dentro del mismo correo electrónico? Explica el mecanismo para diferenciar los contenidos de los mismos.

Empleando en las cabeceras de comienzo del mensaje una cabecera "*Content-Type: Multipart ...*". En esta cabecera el parámetro *boundary* indicará la secuencia ASCII que se utilizará para separar cada uno de los objetos. Después, cada uno de los objetos irá identificado también al comienzo por una cabecera *Content-Type* que indicará el tipo del objeto (*Text*, *Application*, etc.)

4. ***(0,5 puntos)*** Escribe la petición que se generaría cuando un usuario introduce el URL "http://www.upv.es/" en la barra de direcciones de un navegador que emplea el protocolo HTTP versión 1.1. Indica explícitamente los finales de línea y/o líneas en blanco que aparezcan.

```
GET / HTTP/1.1<CR><LF>
Host: www.upv.es<CR><LF>
<CR><LF>
```

La primera línea podría ser también: `GET http://www.upv.es/ HTTP/1.1<CR><LF>`

Donde la secuencia `<CR><LF>` representa los caracteres de retorno de carro y alimentación de línea, respectivamente. Al tratarse de la versión 1.1 de HTTP la cabecera `Host` es obligatoria. La tercera línea es la línea en blanco que identifica el final de las cabeceras.

5. ***(0,5 puntos)*** Dado el código siguiente perteneciente al método *main* del servidor web concurrente de la práctica 5, y suponiendo que **PeticionHTTP** es la clase definida para atender la petición del cliente:

```
ServerSocket serv = new ServerSocket(puerto);
while(true) {
    Socket s = serv.accept();
      …
}
```

indica las líneas de código adecuadas para completar el bucle *while*.

A continuación de la línea "`Socket s = ...`":

```
PeticionHTTP atiendePet = new PeticionHTTP(s);
atiendePet.start();
```

6. ***(2 puntos)*** Escribe un programa en Java, que implemente un cliente TCP con las siguientes características:

   a) El programa requiere dos parámetros de entrada. El primero es el nombre del servidor destino. Asumiremos que este parámetro se proporciona siempre al invocar el programa y contiene un nombre de servidor válido, no comprobaremos los posibles errores. El segundo parámetro es el nombre de un fichero almacenado en el cliente y que se desea enviar al servidor. Si el fichero no existe, el cliente visualizará por pantalla un mensaje con la cadena "*Error: nombre de fichero incorrecto*" y terminará. En caso contrario, empleará **dos** conexiones distintas para transferir el fichero al servidor.

   b) La primera conexión se establece al puerto 20.000 del servidor, y se utiliza para enviar una cadena con el nombre del fichero (por ejemplo, *prueba.pdf*). Dicha cadena acaba con la secuencia de fin de línea "**\r\n**".

   c) Tras recibir el nombre del fichero, el servidor creará un nuevo *socket* servidor al que el cliente debe conectarse. El **puerto** asociado a este *socket* será **aleatorio** (elegido por el sistema operativo). Para que el cliente pueda saber qué puerto es, el servidor se lo enviará como una cadena de texto por la conexión que ambos tienen ya establecida, después cerrará la conexión.

   d) Tras recibir el número de puerto donde escucha el servidor, el cliente se conectará a ese puerto, enviará el fichero al servidor y cerrará la conexión. Se supone que los ficheros transmitidos son siempre binarios.

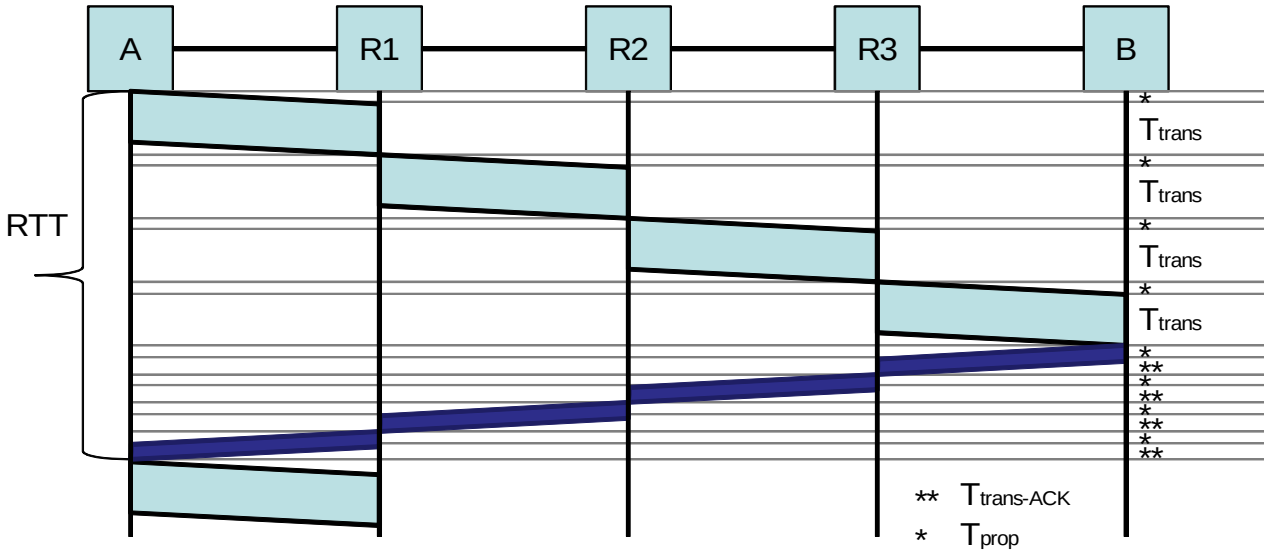
```
import java.net.*;
import java.io.*;
import java.util.*;

class ClienteFicheroTCP {
  public static void main(String args[]) {
     int bytes, port;
     byte[] buffer = new byte[1024];
     String nombre = null;
     FileInputStream fich = null;
       try{
          Socket s = new Socket(args[0],20000);
          try{
              nombre = args[1];
              fich = new FileInputStream(nombre);
          }
          catch(ArrayIndexOutOfBoundsException e) {
            System.out.println("Error: falta el nombre del fichero. Uso:
                                              ClienteFicheroTCP nombre_fichero");
            System.exit(-1);
          }
         catch(FileNotFoundException e) {
            System.out.println("Error: nombre de fichero incorrecto");
            System.exit(-2);
         }
         PrintWriter esc=new PrintWriter(s.getOutputStream());
         Scanner lee = new Scanner(s.getInputStream());
         esc.print(nombre + "\r\n");
         esc.flush();   // nombre + fin de linea
         try{
             port = Integer.parseInt(lee.nextLine());
             Socket s2 = new Socket(args[0],port);
             while ((bytes = fich.read(buffer))!= -1){
                s2.getOutputStream().write(buffer, 0, bytes);
             }
             s.getOutputStream().flush();
          }
          catch(NumberFormatException e) {
             System.out.println("Error numero de puerto
                                             enviado por el servidor incorrecto");
             System.exit(-3);
          }
          fich.close();
          s.close();
       }
       catch (Exception e) {System.err.println(e);}
}    //main
}    //class
```

AVISO: El único tratamiento de excepciones que se solicitaba en el examen es el correspondiente a la excepción *FileNotFoundException* para comprobar si el fichero existe. El resto podían lanzarse de forma general mediante "*throws Exception*" en la cabecera del método *main*.

7. ***(2 puntos)*** Un computador A debe transmitir un fichero de 20000 B a través de una red de conmutación de paquete que incluye tres *routers* (con tiempo de proceso despreciable), y cuatro líneas iguales de 100m con V_{prop}=2 x 10^8 m/s y V_{trans}=100 Mbps hasta otro computador B. El tamaño máximo de paquete es 1000 B, incluyendo una cabecera que consideramos despreciable. Cada vez que B recibe un paquete transmitirá un paquete de reconocimiento de 20B.
   a) ¿Cuánto tiempo pasaría desde el comienzo de la comunicación hasta que A recibe el reconocimiento de su último paquete si se emplea una técnica de parada y espera (*Stop&Wait*)?
   b) ¿Cuánto si se emplea una ventana deslizante con tamaño máximo de 5000B?

Para la transmisión del fichero de 20000 B en paquetes de 1000 B son necesarios 20 paquetes. El tiempo de transmisión de uno de ellos será T_{trans}= L / V_{trans} = 8 x 1000 / 10^8 = 80 x 10^{-6} segundos. El tiempo de propagación en cada línea será de T_{prop} = 100 / 2 x 10^8 = 50 x 10^{-8}= 0,5 x 10^{-6} segundos.

a) Parada y espera: Cada paquete debe ser reconocido individualmente. Hasta que no llegue el reconocimiento de un paquete, no se permite enviar el siguiente. Gráficamente:



Como puede apreciarse, el tiempo que se tarda en recibir el ACK de un paquete (al que denominaremos RTT), puede calcularse mediante la expresión:

$$RTT = 4 \text{ saltos} \times (T_{trans} + T_{prop}) + 4 \text{ saltos} \times (T_{trans\text{-}ACK} + T_{prop}) = 4 \times T_{trans} + 4 \times T_{trans\text{-}ACK} + 8 \times T_{prop}$$

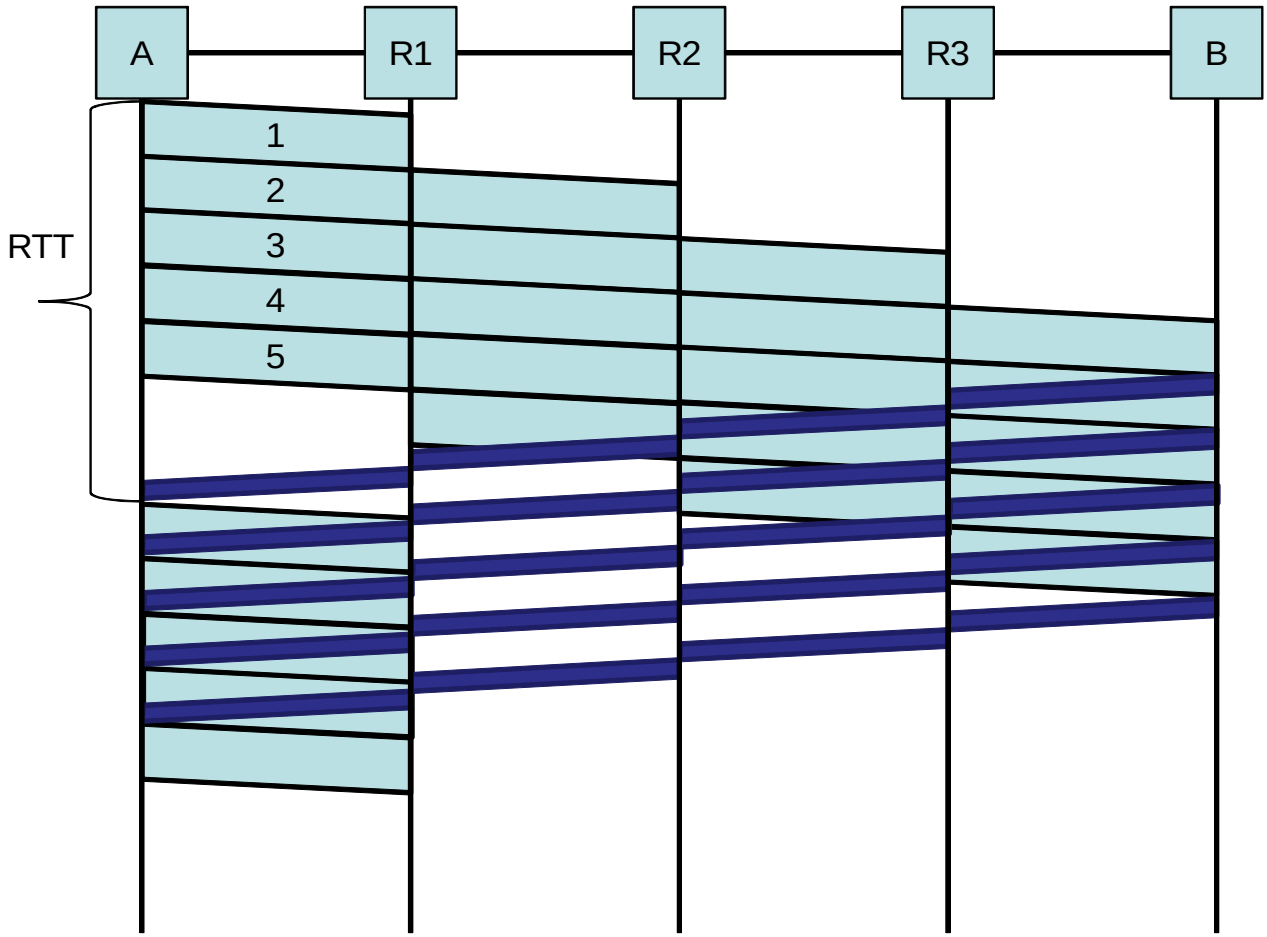
Donde $T_{trans\text{-}ACK}$ = 8 x 20 / V_{trans} = 160 / 10^8 = 1,6 x 10^{-6} segundos.

Luego RTT = 320 x 10^{-6} + 6,4 x 10^{-6} + 4 x 10^{-6} = **330,4 x 10^{-6} segundos**

Puesto que cada paquete ha de transmitirse después de recibido el ACK anterior, serán necesarios 20 x RTT segundos para recibir el ACK del último paquete, luego:

$$T_{total} = 20 \times RTT = \textbf{6608 x } \mathbf{10^{-6}} \textbf{ segundos}$$

b) Puesto que la ventana de transmisión es de 5000B, podremos transmitir cinco paquetes antes de necesitar un reconocimiento. Gráficamente:



El primer aspecto a considerar sería estudiar si se produce la condición de envío continuo, es decir, si el ACK del primer paquete llega antes de finalizar la transmisión del quinto. Para ello debe cumplirse que RTT < 5 x T_{trans}, situación que **sí** se produce (RTT = 330,4 x 10^{-6}, 5 x T_{trans} = 5 x 80 x 10^{-6} = 400 x 10^{-6}).

Esto significa que no dejamos de transmitir paquetes de forma consecutiva: después del quinto se transmite el sexto puesto que ya ha llegado el ACK del primero, y así sucesivamente.

Por tanto, tardaremos 19 x T_{trans} en llegar al último paquete (el 20), y a continuación, un RTT más tarde, llegará el último reconocimiento:

$$T_{total} = 19 \times T_{trans} + RTT = (1520 + 330,4) \times 10^{-6} = \mathbf{1850,4 \times 10^{-6}\ segundos}$$

8. ***(1.5 puntos)*** Indica qué violaciones del protocolo TCP se producen en la siguiente traza que corresponde a una conexión entre los computadores A y B, en la que win(A)=win(B)=500 y MSS=100. Una vez detectado un error, analiza el resto de la traza como si el error no se hubiese producido.

Línea	Origen	Número de secuencia	Flags	Número ACK	Datos (primer … último byte)
1	A	1000	SYN	---	---
2	B	5000	SYN, ACK	1000	---
3	A	1000	ACK	5000	---
4	A	1000	ACK	5000	1000...1100
5	A	1101	ACK	5000	1101...1199
6	B	5000	ACK	1199	---
7	A	1200	ACK	5000	1200...1299
8	A	1300	ACK	5000	1300...1399
9	A	1400	ACK	5000	1400...1499
10	A	1500	ACK	5000	1500...1599
11	A	1600	ACK	5000	1600...1649
12	B	5000	ACK	1650	---
13	A	1651	FIN, ACK	5000	---
14	B	5000	FIN, ACK	1651	---
15	A	1651	ACK	5001	---

Línea 2: El ACK debe ser 1001 (protocolo de establecimiento de la conexión TCP)

Línea 3: El ACK debe ser 5001 (protocolo de establecimiento de la conexión TCP)

El número de secuencia debe ser 1001, pero este error es consecuencia de un error ya reportado. Por tanto, no hace falta indicarlo de nuevo. En lo sucesivo solo se reporta el error en la primera línea dónde aparece.

Línea 4: Se envían 101 bytes de datos y el MSS es de 100

Línea 6: El ACK debe ser 1200 ya que el último byte recibido es el 1199

Líneas 7 a 11: Se envían 5 segmentos de tamaño máximo. El control de congestión no lo permite (estamos en fase de *Slow start* y sólo se ha recibido un ACK). Además, deberían reconocerse al menos los segmentos 8 y 10, suponiendo reconocimientos retardados, o todos si no se suponen.

Línea 13: El número de secuencia debe ser 1650 ya que el último byte enviado es el 1649

Línea 14: El ACK debe ser 1652 (protocolo de cierre de la conexión TCP)

Línea 15: Falta un segmento procedente de A reconociendo el segmento FIN enviado por B (protocolo de cierre de la conexión TCP)

9. ***(1.5 puntos)*** Tras ejecutar la orden **nslookup -debug www.facebook.com** se obtiene el resultado siguiente (la opción **-debug** es para que muestre más información en la respuesta):

```
------------
    QUESTIONS:
        www.facebook.com, type = A, class = IN
    ANSWERS:
    ->  www.facebook.com
        internet address = 69.63.189.34
        ttl = 51
    AUTHORITY RECORDS:
    ->  www.facebook.com
        nameserver = glb2.facebook.com.
        ttl = 467
    ->  www.facebook.com
        nameserver = glb1.facebook.com.
        ttl = 467
    ADDITIONAL RECORDS:
    ->  glb2.facebook.com
        internet address = 69.171.255.10
        ttl = 2484
    ->  glb1.facebook.com
        internet address = 69.171.239.10
        ttl = 2484
------------
Non-authoritative answer:
Name:   www.facebook.com
Address: 69.63.189.34
```

a) ¿Qué información está intentando obtener el cliente?¿Qué información ha enviado al servidor?

Información solicitada por el cliente: dirección IP asociada al nombre **`www.facebook.com`** (véase el campo **`QUESTIONS`**. **`type = A`** significa que se pregunta por una dirección IP).
Al servidor se enviá una petición DNS únicamente con la clausula **`QUESTIONS`**).

b) Interpreta el contenido del campo **`AUTHORITY RECORDS.`**

En este campo aparecen los nombres de los servidores de nombres autorizados para resolver el nombre **`www.facebook.com.`** Son: **`glb2.facebook.com`** y **`glb1.facebook.com .`**
Son registros DNS de tipo NS.

c) Interpreta el contenido del campo **`ADDITIONAL RECORDS`**, incluyendo el significado del parámetro **TTL**.

Este campo indica las direcciones IP de los servidores de nombres listados en el apartado anterior. Concretamente, **`69.171.255.10`** es la dirección IP de **`glb2.facebook.com`** y **`69.171.239.10`** la de **`glb1.facebook.com`** . Son registros DNS de tipo A.
El parámetro **TTL** indica el tiempo de vida de la información asociada.
Los servidores de nombres intermedios que reciban esta respuesta y/o el computador que la solicitó, pueden almacenar la información contenida en este respuesta en su caché, de forma que se aceleren las respuestas a futuras peticiones sobre la misma cuestión. Cada uno de los datos (registros DNS) incluidos en la respuesta tienen un **TTL** que indica durante cuánto tiempo se puede mantener ese dato (registro) en la caché, es decir, el tiempo de caducidad de ese dato (registro).
El **`TTL=2484`** asociado al registro de tipo A que vincula el nombre **`glb2.facebook.com`** con la dirección IP **`69.171.255.10`** indica que la validez de esta vinculación es de 2484 segundos.