REDES
1º Parcial
(2020)

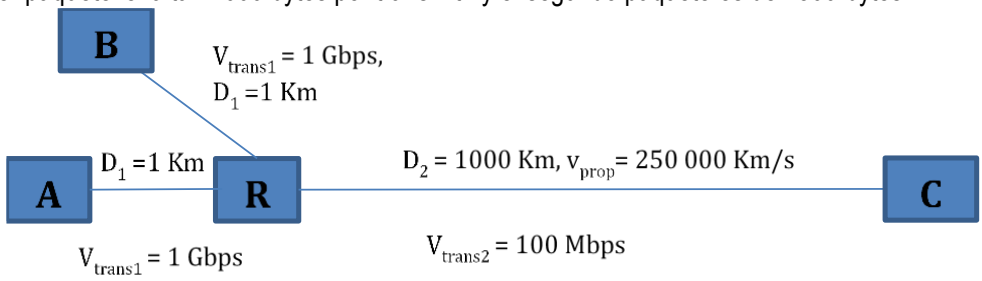
EN MAYÚSCULAS

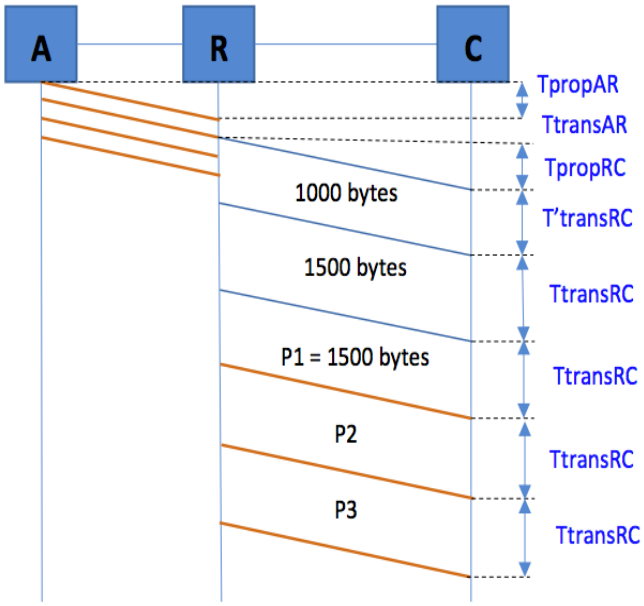
APELLIDOS:_____NOMBRE: _____

grupo

1. (1,5 ptos) El host A envía tres paquetes de 1500 bytes cada uno (P1, P2 y P3) al host C usando la red de conmutación de paquetes de la figura. El primer paquete de A (P1) encuentra dos paquetes, que habían sido enviados previamente por B, en la cola de salida del router R. De los paquetes ya encolados, al primer paquete le faltan 1000 bytes por transmitir y el segundo paquete es de 1500 bytes.

Calcula el tiempo total transcurrido desde que se inicia la transmisión de P1 en A hasta que el paquete P3 se recibe en C. Se supone que los tiempos de procesamiento en los hosts y el router son despreciables. La velocidad de propagación en todos los enlaces es de $2{,}5x10_8$ m/s.





(Todos los tiempos finales en micro-segundos)

Sobre el enlace AR:

$t_{propAR} = 10^3/(2{,}5x10^8) = 4 \times 10^{-6}$ s = 4μS

$t_{transAR} = 1500 \times 8 / 10^9 = 12 \times 10^{-6}$ s = 12μS

Sobre el enlace RC:

$t_{propRC} = 10^6/(2{,}5x10^8) = 4 \times 10^{-3} = 4.000 \times 10^{-6}$ s = 4.000 μS

$t'_{transRC} = 1000x8/10^8 = 80x10^{-6}$ s = 80 μS (resto del paquete que ya está saliendo del router, de 1000 bytes)

$t_{transRC} = 1500x8/10^8 = 120x10^{-6}$ s = 120 μS (cada paquete de 1500 bytes)

$T_{total} = (t_{propAR} + t_{transAR}) + (t_{propRC} + t'_{transRC} + 4\ t_{transRC})$

T_{total} = (4 μS + 12 μS) + (4000 μS + 80 μS + 4x 120 μS) = **4576 μS** = 4,576 mS

2. (1,5 ptos) Indica la información que proporcionan las siguientes cabeceras HTTP y si la cabecera puede figurar en mensajes de petición (C), de respuesta (S) o en ambos (A).

Cabecera	Información que proporciona	C/S/A
Host: www.upv.es	Obligatoria a partir de HTTP 1.1 Un mismo servidor puede alojar distintos **"espacios-web" (servidores virtuales)**. "Host" indica cual es el que se solicita.	C
Last Modified: Fri, 16 Jun 2008 16:49:46 GMT;	Fecha última modificación del recurso en el servidor.	S
Connection: Keep-alive	Si cliente → cliente solicita conexión persistente. Si servidor → servidor decide que es persistente.	A
If-modified-since: Wed, 13 Sep 2009 22:51:57 GMT;	Para la petición condicional de un objeto. Objetivo: evitar que el servidor envíe el objeto de nuevo si la versión en la caché del cliente coincide con la del servidor.	C

3. (1,5 ptos) Indica la secuencia total de peticiones y respuestas DNS necesarias para que el servidor de correo smtp.upv.es obtenga la IP del servidor SMTP del dominio gmail.com. El servidor DNS de la UPV (dns.upv.es) tiene en caché los servidores TLD necesarios (y sus IPs) para las consultas que tenga que realizar. Las cachés DNS de los demás equipos que intervienen están vacías. Suponemos que los TLD's conocen todos los servidores de nombres autorizados de su dominio, y que los servidores autorizados de un dominio se nombran como dns.dominio. Por ejemplo, el servidor DNS del dominio gmail.com será dns.gmail.com.

Muestra el resultado en la siguiente tabla de tres columnas: origen, destino, registro de la consulta o respuesta (nombre, valor y tipo). Si en el registro de la consulta algún campo no contiene valor, indícalo con una línea, por ejemplo, una consulta que no contiene valor en el campo "valor", sería: (p1.yahoo.es, ___ , AAAA). En caso necesario pueden indicarse varios registros por intercambio.

Origen	Destino	Información del registro (nombre, valor, tipo)
smtp.upv.es	dns.upv.es	gmail.com, ----, MX
dns.upv.es	TLD.com	gmail.com, ___,MX
TLD.com	dns.upv.es	gmail.com, dns.gmail.com, NS. dns.gmail.com, IP_ dns.gmail.com, A
dns.upv.es	dns.gmail.com	gmail.com, ___,MX
dns.gmail.com	dns.upv.es	gmail.com, smtp.gmail.com, MX smtp.gmail.com, IP_smtp.gmail.com, A
dns.upv.es	smtp.upv.es	gmail.com, smtp.gmail.com, MX smtp.gmail.com, IP_smtp.gmail.com, A

4. (1,5 ptos) Escribe un programa en java que implemente un servidor TCP secuencial. El servidor debe escuchar en el puerto 8888. Al conectarse un cliente, el servidor enviará un mensaje de texto indicando "Eres el cliente nº x", donde x indica el número de clientes que se han conectado desde que el servidor se puso en ejecución. Así el primer cliente que se conecte será el nº 1, el segundo el nº 2, y así sucesivamente. A continuación, el servidor visualizará en pantalla la respuesta enviada por el cliente y cerrará la conexión.

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

public class ServidorTCP
{
    public static void main(String args[]) throws IOException  {
     int i = 0;
         ServerSocket ss = new ServerSocket(8888);
         while(true){
            Socket s = ss.accept();
            i = i+1;
            PrintWriter esc = new PrintWriter(s.getOutputStream(), true);
            Scanner lee = new PrintWriter(s.getInputStream());
            esc.println("Eres el cliente número " + i);
            System.out.println(lee.nextLine());
            s.close();
          }   // while




    }
}
```

Explica, incluyendo también la orden a ejecutar, cómo puedes probar el funcionamiento de tu servidor mediante el programa `nc` si el cliente y el servidor se ejecutan en el mismo ordenador (rdc01.disca.upv.es).

Desde un terminal hay que ejecutar la orden: **`nc localhost 8888.`**

Al conectarse, el cliente (programa nc) debe recibir la línea de texto del servidor "Eres el cliente número 1", después el usuario tecleará una línea de texto que finalice con el carácter de fin de línea. El contenido de esa línea debe aparecer en la salida estándar del servidor. Si se ejecuta la orden nc varias veces seguidas el número de cliente debe ir incrementándose de uno en uno.

5. (1,5 ptos) Tras establecer una conexión, la tabla refleja la evolución de la ventana de recepción de B en cada RTT. Suponiendo que A tiene infinitos segmentos para enviar, que en el **RTT=4** se detectan 3 ACK's duplicados y en **RTT=7** se produce un TimeOut (estos eventos se detectan a final del RTT, y por tanto afectan al siguiente RTT) completa la tabla siguiente. No existen más errores ni se emplean reconocimientos retardados. Todos los elementos se miden en segmentos.

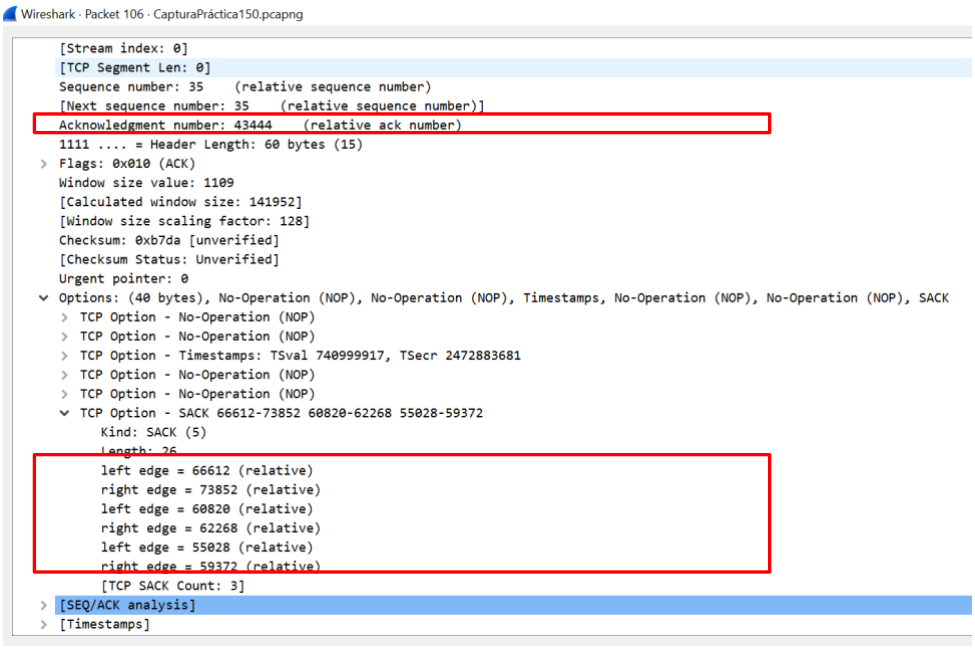
RTT	1	2	3	4	5	6	7	8	9	10
V_rec(B)	20	20	20	12	20	15	15	15	15	15
Umbral(A)	20	20	20	20	6	6	6	4	4	4
V_cong(A)	2	4	8	16	6	7	8	1	2	4
V_trans(A)	2	4	8	12	6	7	8	1	2	4

6. (1,5 ptos) La siguiente imagen muestra información sobre una captura realizada con Wireshark:

a) Indica hasta qué byte de datos se ha recibido correctamente y en secuencia. Marca en la imagen el campo(s) de la cabecera que te proporciona esta información.

Hasta el byte **43443** inclusive.

b) ¿Se ha podido perder algún segmento? En caso afirmativo, indica qué rango de bytes se han podido perder y, si el receptor ha recibido algún segmento fuera de orden, indica cuales. Recuadra en la imagen el campo(s) de la cabecera que te proporciona esta información.



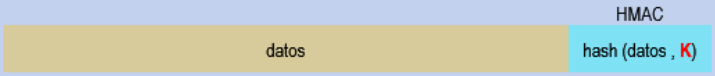
Bytes perdidos: Sí, viendo el ACK y el Left Edge del primer bloque indicado en la opción SACK, deducimos que se ha perdido algún(os) segmento(s), concretamente los indicados a continuación.

43444-55027: desde el byte esperado que indica el campo ACK hasta el byte anterior al que se indica como recibido en el primer bloque de la opción SACK (Left Edge de dicho bloque).
59372-60819: Bytes entre el primer y segundo bloque recibido, especificados en la opción SACK.
62268-66611: Bytes entre el segundo y tercer bloque recibido, especificados en la opción SACK.

Bytes recibidos fuera de orden:
55028-59371,
60820-62267,
66612-73851.

7. (1 pto) Explica brevemente en qué consiste el HMAC (Hash-based Message Authentication Code) y qué propiedades de seguridad se consiguen con él.

- Basado en clave SECRETA
- HMAC = Hash (datos, K)



K es la clave **SECRETA** (simétrica) no confundir con K+ o K-

- Si al calcular el HASH de un bloque añadimos "virtualmente" la clave secreta (conocida por TX y RX):
  - El pirata ya no puede alterar los datos y recalcular el HMAC
  - No solo verificamos la INTEGRIDAD de los datos, también la AUTENTICIDAD del transmisor. (El que ha transmitido es alguien de "nuestra red" porque conoce la K.)
  -