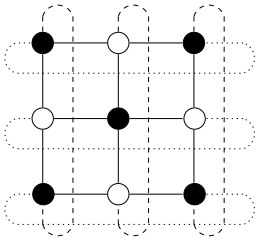

E52. Redes Informáticas

Práctica 7. El nivel de transporte en Internet: Protocolos TCP y UPD

Descripción del equipo y del software

1. Dos ordenadores de tipo PC con S.O. GNU/Linux.
2. Adaptador de red *Ethernet* con conector RJ-45 (estándares 10BaseT/100BaseT).
3. Conmutador 10BaseT.
4. Monitor de red y analizador de protocolos *Ethereal*.
5. Compilador *gcc* de C.

1. Introducción

Como vimos en la práctica anterior, el conjunto de protocolos TCP/IP consta de una serie de normas para el nivel de red y el nivel de transporte que tienen como característica común el uso de IP (*Internet Protocol*) para el intercambio de *datagramas* entre estaciones conectadas a redes de cualquier naturaleza.

En esta práctica se analizarán los protocolos de nivel de transporte de Internet, TCP (*Transmission Control Protocol*) y UDP (*User Datagram Protocol*), capturándose para este propósito tráfico de ambos tipos mediante la aplicación *Ethereal*. Es conveniente recordar aquí que la diferencia fundamental entre el modo de funcionamiento de estos protocolos es que, mientras que TCP ofrece un servicio orientado a conexión y fiable, UDP es todo lo contrario: el servicio es sin conexión y, además, no fiable. Al bloque de datos enviado por el protocolo TCP (cabecera de TCP más datos del nivel de aplicación) se le denomina *segmento*. A falta de un nombre mejor, al bloque de datos enviado por el protocolo UDP le denominaremos igual que en el nivel de red, es decir, *datagrama*.

A fin de generar el tráfico necesario para su posterior análisis, se procederá a comunicar dos procesos en máquinas diferentes mediante paso de mensajes. Éste es el mecanismo de comunicación entre procesos más utilizado hoy en día. Su ventaja más destacable es que puede ser empleado en cualquier tipo de arquitectura, sea monoprocesador, multiprocesador con memoria compartida, multicomputador o sistema distribuido.

Los procesos comunicantes se generarán a partir de sencillos programas escritos en C que utilizan la interfaz de programación de aplicaciones (API) de sockets, basada en paso de mensajes. No forma parte de los objetivos de esta práctica el estudio en detalle de esta interfaz, que se pospone a prácticas posteriores, sino que la utilización de la API de sockets se hace a fin de disponer de un mejor control sobre el envío/recepción de mensajes entre procesos.

2. Desarrollo de la práctica

Para la realización de esta práctica, dependiendo de la disponibilidad de equipos, se formarán grupos de trabajo de modo que cada grupo disponga de dos ordenadores. Para evitar la confusión con otro tráfico durante las capturas, es muy conveniente que establezcáis en *Ethereal* filtros lo más precisos posible, que incluyan los puertos TCP o UDP implicados en el intercambio de paquetes.

2.1. Comunicación entre procesos mediante paso de mensajes

El tráfico TCP se generará mediante un par de programas muy sencillos, escritos en C, que contienen el código correspondiente a un *cliente* y un *servidor*. Siendo TCP un protocolo orientado a conexión, el intercambio de información debe ir precedido y seguido, respectivamente, de las fases de establecimiento y liberación de la conexión. Esquemáticamente la evolución temporal de las acciones que llevan a cabo cliente y servidor se muestra a continuación:

Cliente	**Servidor**
Fase 1: Establecimiento de la conexión	
C1: Solicitar la conexión	S1: En espera de una conexión... S2: Recibida la solicitud de conexión
Fase 2: Intercambio de datos	
C2: Leer un número C3: Enviar el número al servidor	S2: En espera de un número... S3: Recibido el número del cliente S4: Escribir el número
Fase 3: Liberación de conexión	
C4: Liberar la conexión	S5: Liberar la conexión

El código en C del cliente se muestra a continuación:

```
#include "cliente_servidor.h"
int main()
{
  int                sock, numero;
  struct sockaddr_in servidor;
  struct hostent     *hp;

  hp = gethostbyname(nombre_dns_de_servidor);
  if (!hp) {
    fprintf(stderr, "gethostbyname: %s\n", hstrerror(h_errno));
    exit(1);
  }
  servidor.sin_family = AF_INET;
  servidor.sin_port = htons(puerto_en_servidor);
  memcpy(&servidor.sin_addr, hp->h_addr, hp->h_length);
```

```
  sock = socket(AF_INET, SOCK_STREAM, 0);
  if (sock < 0) {
    perror("socket");
    exit(1);
  }

  /* Fase 1: Establecimiento de la conexión */
  if (connect(sock, (struct sockaddr *) &servidor,
              sizeof(struct sockaddr_in)) < 0) {
    perror("connect");
    exit(1);
  }

  /* Fase 2: Intercambio de datos */
  fprintf(stdout, "Dame el número: ");
  fflush(stdout);
  fscanf(stdin, "%d", &numero);
  if (send(sock, &numero, sizeof(int), 0) < 0) {
    perror("send");
    exit(1);
  }

  /* Fase 3: Liberación de la conexión */
  if (close(sock) < 0) {
    perror("close");
    exit(1);
  }

  exit(0);
}
```

El código en C del servidor se muestra a continuación:

```
#include "cliente_servidor.h"
int main()
{
  int                sock, nuevo_sock, longitud, numero;
  struct sockaddr_in yo, cliente;

  yo.sin_family = AF_INET;
  yo.sin_port = htons(puerto_en_servidor);
  yo.sin_addr.s_addr = INADDR_ANY;

  sock = socket(AF_INET, SOCK_STREAM, 0);
  if (sock < 0) {
    perror("socket");
    exit(1);
  }

  if (bind(sock, (struct sockaddr *) &yo,
           sizeof(struct sockaddr_in)) < 0) {
    perror("bind");
    exit(1);
```

```
  }

  if (listen(sock, 5) < 0) {
    perror("listen");
    exit(1);
  }

  /* Fase 1: Establecimiento de la conexión */
  longitud = sizeof(struct sockaddr_in);
  nuevo_sock = accept(sock, (struct sockaddr *) &cliente, &longitud);
  if (nuevo_sock < 0) {
    perror("accept");
    exit(1);
  }

  /* Fase 2: Intercambio de datos */
  if (recv(nuevo_sock, &numero, sizeof(int), 0) < 0) {
    perror("recv");
    exit(1);
  }
  fprintf(stdout, "El número es: %d\n", numero);

  /* Fase 3: Liberación de la conexión */
  if (close(nuevo_sock) < 0) {
    perror("close");
    exit(1);
  }

  exit(0);
}
```

Por último, tanto el código del cliente como el código del servidor incluyen el fichero de cabecera `cliente_servidor.h` cuyo código se muestra a continuación:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#include <unistd.h>

#define nombre_dns_de_servidor "labsop01.act.uji.es"
#define puerto_en_servidor 16385
```

Es importante hacer notar aquí que, dependiendo del ordenador donde se ejecute el proceso correspondiente al servidor, puede ser necesario modificar la línea del fichero `cliente_servidor.h` donde figura el nombre DNS del servidor.

Los códigos de estas rutinas pueden descargarse desde la página web de la asignatura.

2.2. Establecimiento de la conexión en TCP

En primer lugar se analizarán los segmentos que se intercambian durante el establecimiento de una conexión sobre TCP. Para ello se deben efectuar los siguientes pasos:

1. Descargar el código del cliente sobre uno de los ordenadores disponibles, al que llamaremos `labsopC`, el código del servidor sobre el otro, al que llamaremos `labsopS`.
2. Modificar el fichero `cliente_servidor.h` para que el nombre DNS del servidor sea correcto en ambos ordenadores (como mínimo en el cliente).
3. Generar el fichero ejecutable correspondiente a cada ordenador (por ejemplo, `cc -o servidor servidor.c`).
4. Configurar el analizador de tráfico *Ethereal* para que capture únicamente el tráfico TCP entre `labsopC` y `labsopS` (en el puerto adecuado), e iniciar la captura.
5. Lanzar a ejecución el proceso servidor en `labsopS`.
6. Lanzar a ejecución el proceso cliente en `labsopC`.
7. Detener la captura antes de introducir el número solicitado por el cliente y analizar el tráfico generado entre cliente y servidor.

   A continuación se ofrecen algunas preguntas para ayudar a este análisis:

   - ¿Cuántos segmentos TCP se han intercambiado entre cliente y servidor?
   - ¿Qué información figura en los campos de puerto de origen y puerto de destino del primer segmento que circula del cliente al servidor? ¿Por qué?
   - ¿Qué información figura en los campos de número de secuencia y de reconocimiento de los segmentos intercambiados durante el establecimiento de la conexión? ¿Por qué?
   - Relacionad el intercambio de estos segmentos con el diagrama (o máquina) de estados correspondiente al establecimiento de una conexión TCP visto en clase de teoría.
   - ¿Qué otra información puedes identificar como parte de la cabecera de TCP?

2.3. Intercambio de datos en TCP

En este apartado se pretende reflexionar sobre la eficiencia del intercambio de datos sobre una conexión TCP. Para ello se repetirán los pasos de la captura anterior, pero incluyendo también y considerando únicamente los segmentos correspondientes al intercambio de datos (la fase 2).

Con el tráfico capturado durante la fase 2 a la vista, contestad a continuación a las siguientes cuestiones:

- ¿Cuántos segmentos TCP se intercambian entre cliente y servidor para enviar un único dato (el número intercambiado)?
- ¿Cuál es la eficiencia de este protocolo (ratio entre bytes de datos "útiles" y bytes de datos enviados)?
- ¿Qué información figura en los campos de número de secuencia y de reconocimiento de los segmentos intercambiados durante la transmisión de datos? ¿Por qué?

Modificad el código del servidor para que, al recibir el número, le sume 1 y lo devuelva al cliente. Modificad asimismo el código del cliente para que muestre el número devuelto en la salida estándar. Capturad el tráfico correspondiente al intercambio de datos de los nuevos programas, y contestad a las siguientes preguntas:

- ¿Cuántos segmentos TCP se intercambian ahora entre cliente y servidor durante el intercambio de datos?
- ¿Es eficiente?

2.4. Liberación de la conexión en TCP

El objetivo de este tercer apartado sobre TCP es analizar la etapa final de este tipo de conexiones, es decir, estudiar el tráfico que se genera cuando se libera la conexión. Para ello, bastará con repetir la captura de los paquetes intercambiados, pero cuidándose ahora de capturar también el tráfico correspondiente a la liberación de la conexión. Una vez disponible este tráfico en la ventana de análisis de la aplicación, es conveniente responder a las siguientes preguntas:

- ¿Cuántos segmentos de TCP se han intercambiado entre cliente y servidor para efectuar la desconexión?
- ¿Qué información figura en los campos de número de secuencia y de reconocimiento de los segmentos intercambiados durante la liberación de la conexión? ¿Por qué?
- Relacionad el intercambio de estos segmentos con el diagrama de estados correspondiente a la liberación de conexión TCP visto en clase de teoría.

2.5. Tráfico sobre UDP

El tráfico UDP se generará también mediante un par de programas en C muy sencillos a los que, debido a la naturaleza sin conexión de este protocolo, llamaremos *emisor* y *receptor*. En este caso el emisor leerá el número de su entrada estándar y lo enviará al receptor que lo visualizará en su salida estándar.

Los códigos de estas rutinas también pueden descargarse desde la página web de la asignatura. Será necesario que modifiquéis el fichero `emisor_receptor.h` para establecer el nombre real del receptor.

Para el análisis del tráfico UDP procederemos a configurar adecuadamente los filtros de la aplicación *Ethereal* para la captura de este tipo de tráfico y a ejecutar *emisor* y *receptor* en máquinas diferentes. A la vista de los resultados (*datagramas* capturados), contestad a las siguientes preguntas:

- ¿Qué información figura en los campos de puerto origen y puerto de destino del primer datagrama que circula del emisor al receptor? ¿Por qué?
- ¿Qué valor figura en el campo de longitud de la cabecera del datagrama UDP? ¿A qué corresponde este valor?
- ¿Ha sido utilizado el campo de comprobación de error?

- ¿Cuántos datagramas UDP se intercambian entre emisor y receptor para enviar un único dato?
- ¿Qué protocolo es más eficiente, TCP o UDP?