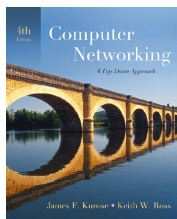


Introducción a las Redes de Computadoras

Capitulo 2 Capa de Aplicación



Nota acerca de las transparencias del curso:
Estas transparencias están basadas en el sitio web que
acompaña el libro, y
han sido modificadas por los docentes del curso.
All material copyright 1996-2007
J.F Kurose and K.W. Ross, All Rights Reserved

Capa de aplicación

- 2.1 Principio de aplicaciones de red
- 2.2 Web y HTTP
- 2.3 Programación con sockets
 - TCP
 - UDP
- 2.4 Aplicaciones P2P
- 2.5 FTP
- 2.6 Correo electrónico SMTP, POP3, IMAP
- 2.7 DNS

Programación con Sockets

Objetivo: aprender a construir una aplicación cliente/servidor que se comunican a través de sockets

Socket API

- Presentada en BSD4.1 UNIX, 1981
- Explícitamente creados, utilizados y liberados por las aplicaciones
- Paradigma cliente/servidor
- Dos tipos de transporte vía sockets:
 - Datagramas (no confiable)
 - Flujo o stream de bytes (confiable)

socket

interfaz *local al host*,
creada por la aplicación
y *controlada por el S.O.*
(una "puerta") a través de
la cual los procesos
pueden *enviar y recibir*
mensajes desde y hacia
otros procesos

Programación con Sockets

- El socket es la interfaz que la aplicación tiene con las capas inferiores
- Encapsula todo el manejo de la comunicación entre un punto y otro y da la sensación a la aplicación de estar dialogando directamente con la aplicación del otro lado
- Los sockets están asociados a una **dirección** y un **puerto** en el host y conocen la dirección y puerto donde está el socket destino

Programación con Sockets

Las primitivas presentadas para la API de sockets son

SOCKET	Crea un nuevo punto de comunicación
BIND	Engancha el socket con una dirección y puerto locales
LISTEN	Anuncia que se aceptarán conexiones, e indica el largo de la cola
ACCEPT	Bloquea al llamador hasta que llega un intento de conexión
CONNECT	Activamente intenta establecer una conexión
SEND	Envía datos sobre la conexión
RECEIVE	Recibe datos de la conexión
CLOSE	Cierra la conexión

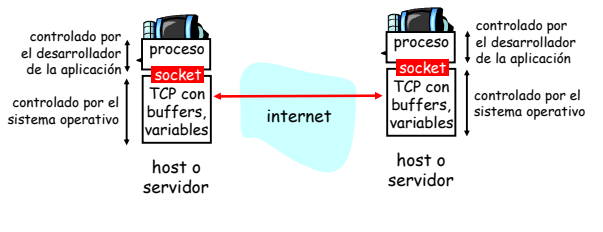
Capa de aplicación

- 2.1 Principio de aplicaciones de red
- 2.2 Web y HTTP
- 2.3 Programación con sockets
 - TCP
 - UDP
- 2.4 Aplicaciones P2P
- 2.5 FTP
- 2.6 Correo electrónico SMTP, POP3, IMAP
- 2.7 DNS

Programación con Sockets TCP

Socket: puerta entre los procesos de aplicación y la capa de transporte (UDP or TCP)

Servicio TCP: transferencia confiable de bytes de un proceso a otro



Programación con Sockets TCP

Lo que hace el servidor

- el proceso servidor debe estar ejecutándose
- el servidor debe haber creado un socket que recibe el contacto del cliente

Lo que hace el cliente

- crea un socket TCP local al cliente
- especifica la dirección IP y el puerto del servidor
- cuando el cliente crea el socket, establece la conexión TCP con el servidor

- Cuando es contactado por el cliente, el servidor TCP crea un nuevo socket a través del cual el proceso servidor se comunicará con el cliente
 - permite que un servidor dialogue con muchos clientes

Programación con Sockets TCP

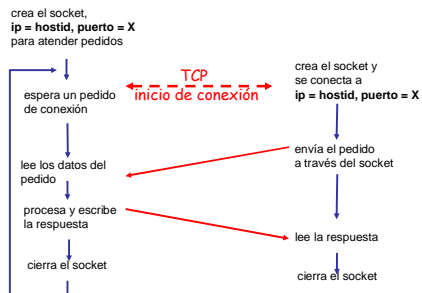
Desde el punto de vista de la aplicación...

TCP provee una transferencia confiable y en orden de bytes entre el cliente y el servidor

Interacción cliente/servidor: TCP

Servidor (se ejecuta en `hostid`)

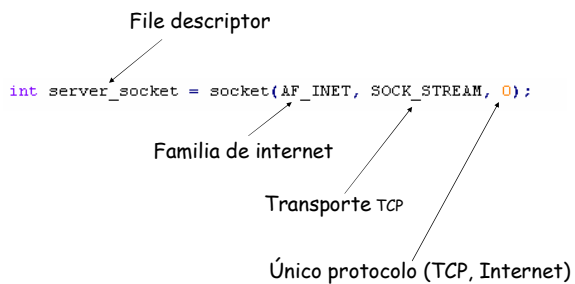
Cliente



Sockets TCP en C

- Los sockets son file descriptors
- Altamente genérico → Gran complejidad de implementación
- Primitivas iguales a las propuestas por Berkeley
- Socket "conectado" y socket "desconectado" se representan de la misma manera

Sockets TCP en C (Servidor)



Sockets TCP en C (Servidor)

Dirección tipo Internet

```
struct sockaddr_in server_addr;
socklen_t server_addr_size = sizeof server_addr;
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(PORT); // Ojo con el endiannes
server_addr.sin_addr.s_addr = inet_addr(MY_IP);
bind(
    server_socket,
    (struct sockaddr*)&server_addr, server_addr_size
);

listen(server_socket, MAX_QUEUE); // Conexiones pendientes a la vez
```

Sockets TCP en C (Servidor)

Se bloquea esperando que alguien se conecte

```
struct sockaddr_in client_addr;
socklen_t client_addr_size = sizeof client_addr;
int socket_to_client = accept(
    server_socket,
    (struct sockaddr *)&client_addr, &client_addr_size
);
```

Devuelve "copia" del socket

Esta será la dirección del socket que está del otro lado

Sockets TCP en C (Servidor)

Ojo! ¿Y si no alcanza el buffer?

```
char* data = malloc(MAX_MSG_SIZE);
int data_size = MAX_MSG_SIZE;
int received_data_size = recv(socket_to_client, data, data_size, 0);
...procesar...
int sent_data_size = send(socket_to_client, data, received_data_size, 0);
```

Ojo! ¿Y si no manda todo?

```
close(socket_to_client); // Cerremos la conexión Pero el socket original no se cerró!
```

Sockets TCP en C (Cliente)

```
int client_socket = socket(AF_INET, SOCK_STREAM, 0);
```

Igual que el servidor

```
struct addrinfo hints, *res;
memset(&hints, 0, sizeof hints);
hints.ai_family = AF_INET;
hints.ai_socktype = SOCK_STREAM;
getaddrinfo(HOST, PORT, &hints, &res);
```

¿Cuál es la dirección IP/puerto que buscamos?

Entre otras cosas Consulta al DNS

Sockets TCP en C (Cliente)

```
connect(client_socket, res->ai_addr, res->ai_addrlen);
```

Desbloqueará al servidor

Los datos que utiliza son precisamente los que le devolvió getaddrinfo

send(), recv() y close() son análogos al servidor

Sockets TCP en Java

En Java se manejan primitivas de sockets ligeramente diferentes a las vistas anteriormente

- Una clase para socket "desconectado": **ServerSocket**
 - está "a la espera" de que un cliente venga a conectarse
- Una clase para socket "conectado": **Socket**
 - conexión ya establecida, se pueden enviar y recibir bytes
 - para eso provee dos un *Stream* de salida y uno de entrada

Sockets TCP en Java

En Java se manejan primitivas de sockets ligeramente diferentes a las vistas anteriormente

SOCKET	Socket, ServerSocket
BIND	ServerSocket.bind()
LISTEN	No hay una primitiva correspondiente
ACCEPT	ServerSocket.accept()
CONNECT	new Socket(address, port) o Socket.bind()
SEND	Socket.getOutputStream().write()
RECEIVE	Socket.getInputStream().read()
CLOSE	Socket.close()

Sockets TCP en Java

- **stream** es una secuencia de caracteres que fluye hacia o desde un proceso
- **stream de entrada** se engancha a alguna fuente de entrada
 - teclado
 - socket
- **stream de salida** se engancha a una fuente de salida
 - monitor
 - socket
- un socket provee dos streams: uno de salida y uno de entrada
- se utilizan para **enviar** y **recibir** datos respectivamente
- también tenemos un stream para tomar datos de la entrada estándar y otro para imprimir datos en pantalla

Sockets TCP en Java (Cliente)

```

public class ClienteTCP {
    public static void main(String[] args) throws Exception {
        String host = "localhost";
        int puerto = 6789;
        if (args.length == 2) {
            host = args[0];
            puerto = new Integer(args[1]);
        } else if (args.length == 1) {
            puerto = new Integer(args[0]);
        }
        BufferedReader entradaDelUsuario =
            new BufferedReader(
                new InputStreamReader(System.in));
        Socket socketCliente = new Socket(host, puerto);
        DataOutputStream salidaAlServidor =
            new DataOutputStream(
                socketCliente.getOutputStream());
    }
}
    
```

configurar host y puerto del servidor →

crear stream de entrada →

crear socket cliente y conectarse al servidor →

crear stream de salida enganchado al socket →

Sokcets TCP en Java (Cliente)

```
crear stream de entrada enganchado al socket } BufferedReader entradaDelServidor =  
new BufferedReader(  
new InputStreamReader(  
socketCliente.getInputStream());  
)  
  
enviar datos al servidor } String frase = entradaDelUsuario.readLine();  
salidaAlServidor.writeBytes(frase + "\n");  
  
leer datos desde el servidor } String fraseModificada =  
entradaDelServidor.readLine();  
System.out.println(fraseModificada);  
  
socketCliente.close();  
)  
)
```

Sokcets TCP en Java (Servidor)

```
public class ServidorTCP {  
  
configurar puerto } public static void main(String[] args) throws Exception {  
int puerto = 6789;  
if (args.length > 0) puerto = new Integer(args[0]);  
  
crear socket de bienvenida } ServerSocket socketServidor = new ServerSocket(puerto);  
  
esperar alguna conexión de parte del cliente } while (true) {  
Socket socketConexion = socketServidor.accept();  
  
crear stream de entrada, enganchado al socket } BufferedReader entradaDelCliente =  
new BufferedReader(  
new InputStreamReader(  
socketConexion.getInputStream());  
)  
}
```

Sokcets TCP en Java (Servidor)

```
crear stream de salida, enganchado al socket } DataOutputStream salidaAlCliente =  
new DataOutputStream(  
socketConexion.getOutputStream());  
  
leer datos enviados por el cliente } String linea = entradaDelCliente.readLine();  
String mayusculas = linea.toUpperCase();  
  
escribir la respuesta en el socket } salidaAlCliente.writeBytes(mayusculas + "\n");  
socketConexion.close();  
  
termina el loop de procesamiento, así que volverá a esperar la siguiente conexión  
)  
)
```

Capa de aplicación

- | | |
|--------------------------------------|---|
| 2.1 Principio de aplicaciones de red | 2.4 Aplicaciones P2P |
| 2.2 Web y HTTP | 2.5 FTP |
| 2.3 Programación con sockets | 2.6 Correo electrónico SMTP, POP3, IMAP |
| • TCP | 2.7 DNS |
| • UDP | |

Programación con Sockets UDP

UDP: no hay conexión entre el cliente y el servidor

- no hay *handshaking*
- el emisor indicará la IP y el puerto en cada paquete que envíe
- El receptor sabrá a qué IP y puerto contestar pues lo obtendrá del paquete

UDP: los datos pueden llegar a destino en desorden, o no llegar

Programación con Sockets UDP

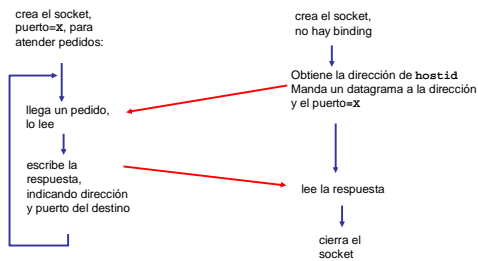
Desde el punto de vista de la aplicación...

UDP provee una transferencia no confiable de grupos de bytes (datagramas) entre el cliente y el servidor

Interacción cliente/servidor: UDP

Servidor (ejecutándose en `hostid`)

Cliente



Sockets UDP en C

- API prácticamente igual a TCP
 - Crearlo con `SOCK_DGRAM` en vez de `SOCK_STREAM`
- No hay `ACCEPT` ni `CONNECT`
- Al enviar datos se debe indicar siempre la `sockaddr` del destino
- Al recibir datos siempre nos indicará la `sockaddr` que los envió

Sockets UDP en C

```
struct sockaddr_in remote_addr;
socklen_t remote_addr_size = sizeof remote_addr;
char* data = malloc(MAX_MSG_SIZE);
int data_size = MAX_MSG_SIZE;
int received_data_size = recvfrom(
    server_socket,
    data, data_size, 0,
    (struct sockaddr*)&remote_addr, &remote_addr_size
);
```

Al recibir datos, nos traemos la dirección del socket remoto donde se enviaron

Sockets UDP en C

```
int sent_data_size = sendto(
    server_socket,
    data, received_data_size, 0,
    (struct sockaddr*)& remote_addr, remote_addr_size
);
```

A esa misma dirección es
que enviamos luego la
respuesta

Sockets UDP en Java

- No es igual a TCP, se usa una clase específica denominada `DatagramSocket`
- Cliente y servidor utilizan la misma clase. No existe un `serverDatagramSocket` pues no existe la noción de "bloquearse esperando la conexión"
- No hay streams de entrada y salida, porque la comunicación se basa en paquetes (datagramas)

Sockets UDP en Java

- Los paquetes que viajan se representan mediante la clase `DatagramPacket`
- A cada paquete que se envíe se le debe indicar la dirección y puerto a donde está destinado
- Cuando el servidor recibe uno de estos paquetes, la dirección y puerto del cliente estarán en el paquete, por lo que sabrá a dónde responder

Sockets UDP en Java (Cliente)

```
public class ClienteUDP {  
    public static void main(String[] args) throws Exception {  
        String host = "localhost";  
        int puerto = 9876;  
        if (args.length == 2) {  
            host = args[0];  
            puerto = new Integer(args[1]);  
        } else if (args.length == 1) {  
            puerto = new Integer(args[0]);  
        }  
        BufferedReader entradaDelUsuario =  
            new BufferedReader(  
                new InputStreamReader(System.in));  
        DatagramSocket socketCliente = new DatagramSocket();  
        InetAddress direccionIP = InetAddress.getByHost(host);  
        String frase = entradaDelUsuario.readLine();  
    }  
}
```

configurar host y puerto

crear stream de entrada del teclado

crear socket del cliente

traducir el nombre de host a su IP (puede usar el DNS)

Sockets UDP en Java (Cliente)

```
        DatagramPacket paqueteEnvio =  
            new DatagramPacket(  
                frase.getBytes(), frase.length(),  
                direccionIP, puerto);  
        socketCliente.send(paqueteEnvio);  
        DatagramPacket paqueteRecepcion =  
            new DatagramPacket(new byte[1024], 1024);  
        socketCliente.receive(paqueteRecepcion);  
        int largo = paqueteRecepcion.getLength();  
        String fraseModificada =  
            new String(paqueteRecepcion.getData(), 0, largo);  
        System.out.println(fraseModificada);  
        socketCliente.close();  
    }  
}
```

crear datagrama con datos, largo, IP y puerto

mandar datagrama al servidor

leer datagrama enviado por el servidor

armar frase a desplegar

Sockets UDP en Java (Servidor)

```
public class ServidorUDP {  
    public static void main(String[] args) throws Exception {  
        int puerto = 9876;  
        if (args.length > 0) {  
            puerto = new Integer(args[0]);  
        }  
        DatagramSocket socketServidor =  
            new DatagramSocket(puerto);  
        while (true) {  
            DatagramPacket paqueteRecepcion =  
                new DatagramPacket(new byte[1024], 1024);  
            socketServidor.receive(paqueteRecepcion);  
        }  
    }  
}
```

configurar el puerto

crear socket del servidor

crear espacio para el datagrama a recibir

recibir datagrama

Sockets UDP en Java (Servidor)

```
String frase =  
    new String(paqueteRecepcion.getData(), 0,  
              paqueteRecepcion.getLength());  
  
InetAddress direccionDestino = paqueteRecepcion.getAddress();  
int puertoDestino = paqueteRecepcion.getPort();  
  
byte[] data = frase.toUpperCase().getBytes();  
  
DatagramPacket paqueteEnvio =  
    new DatagramPacket(  
        data, frase.length(),  
        direccionDestino, puertoDestino);  
  
socketServidor.send(paqueteEnvio);  
    }  
}
```

obtener la IP y puerto del emisor

crear datagrama con la respuesta

escribir datagrama en el socket

termina el loop de procesamiento, así que volverá a esperar la llegada de un datagrama nuevo

Programación con sockets: referencias

Sockets en C

- Beej's Guide To Network Programming, <http://www.beej.us/guide/bgnet/>
- "Unix Network Programming" (J. Kurose), <http://manic.cs.umass.edu/~amldemo/courseware/intro>
- Tanenbaum - Computer Networks Fourth Edition

Sockets en Java

- "All About Sockets" (Sun tutorial), <http://www.javaworld.com/javaworld/jw-12-1996/jw-12-sockets.html>
- "Socket Programming in Java: a tutorial," <http://www.javaworld.com/javaworld/jw-12-1996/jw-12-sockets.html>
- API de Sockets en Java 1.4.2 <http://java.sun.com/j2se/1.4.2/docs/api/index.html>
