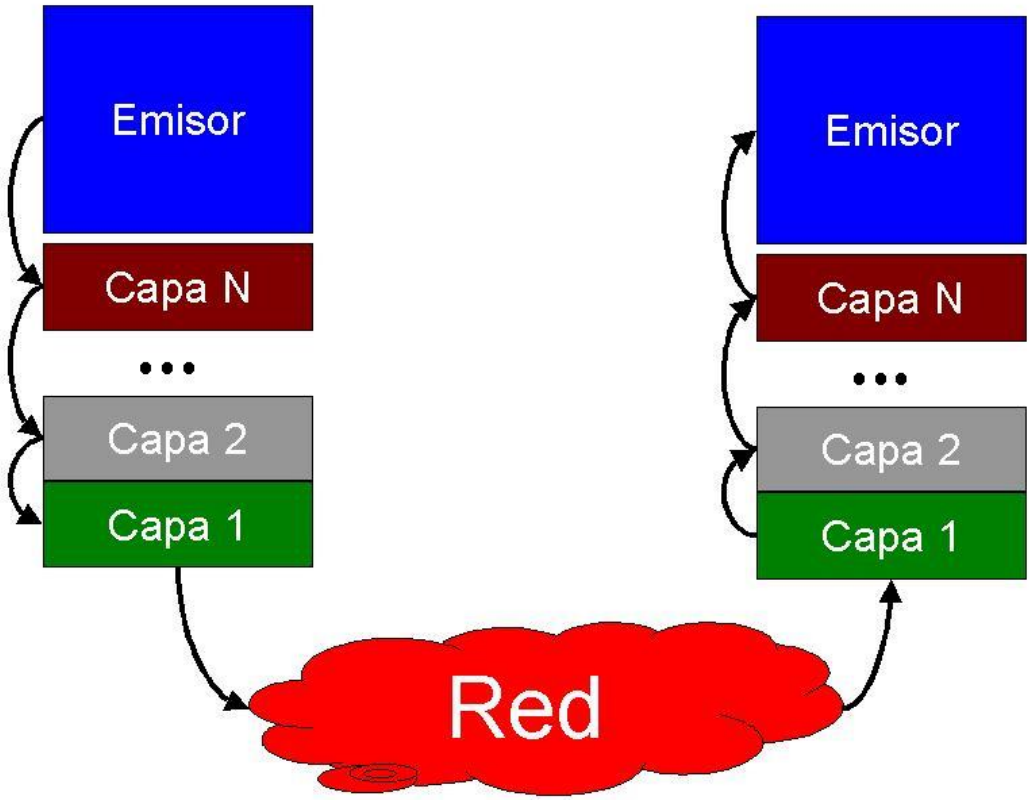
Sockets

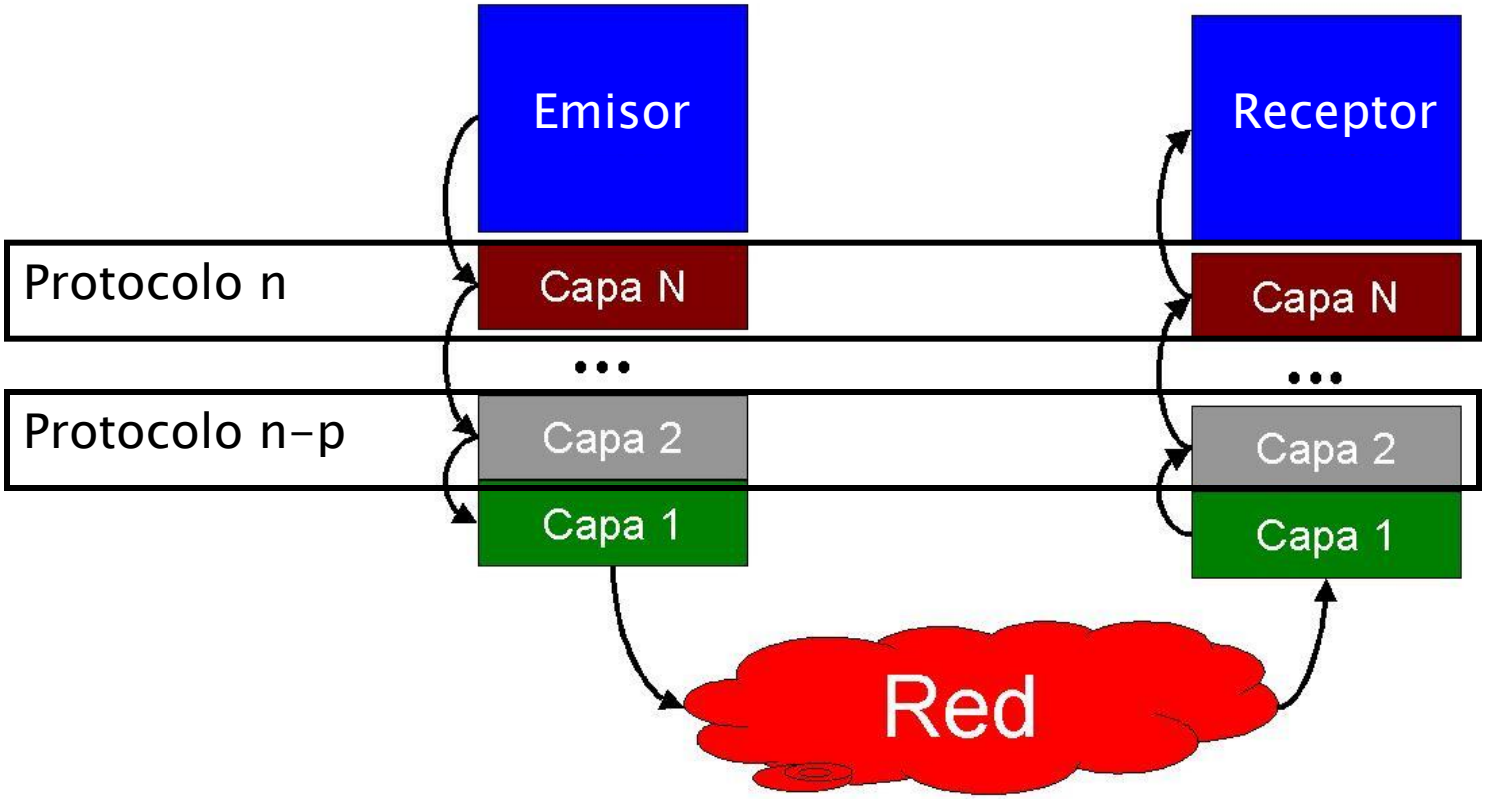
Programación en red

- El término programación en red se refiere a la escritura de programas que se ejecutan a través de múltiples dispositivos que están conectados a través de una red

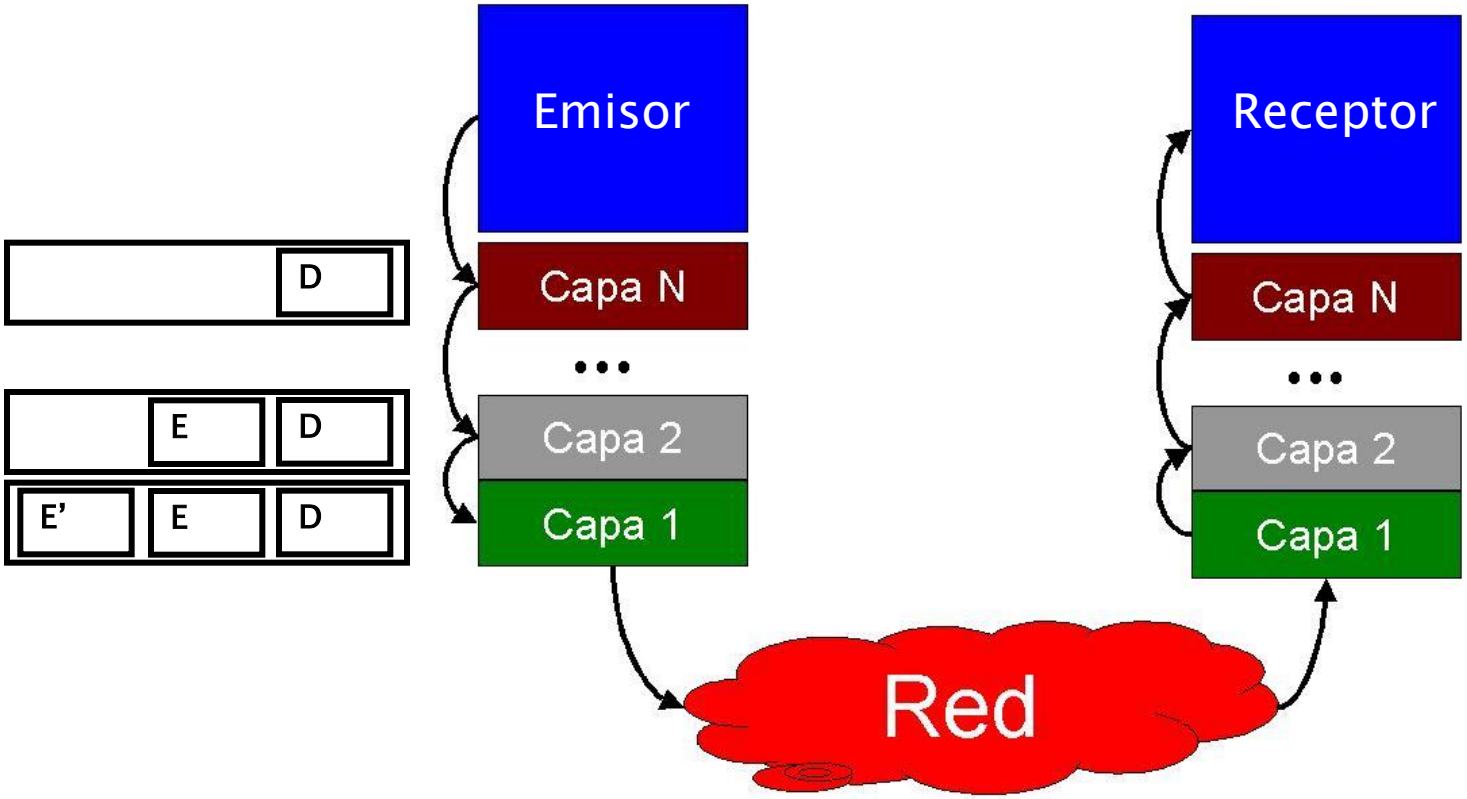
Modelo de referencia



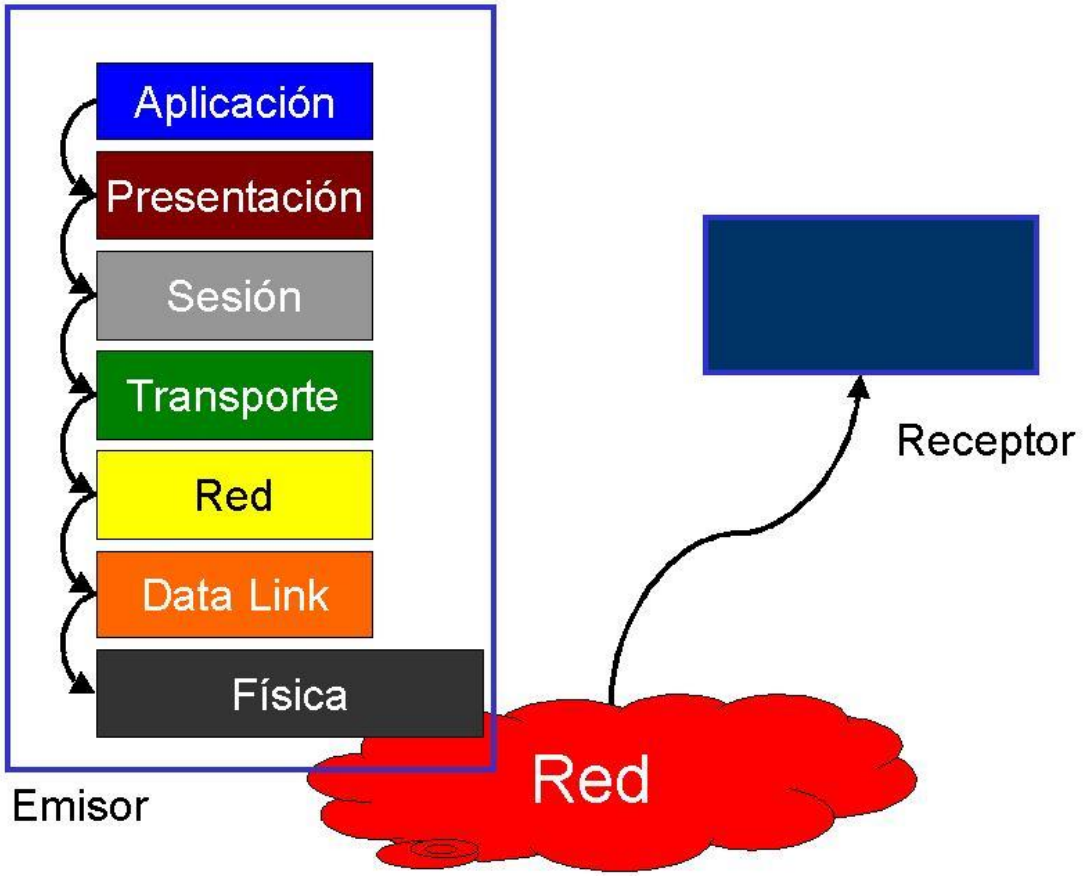
Modelo de referencia



Modelo de referencia



Modelo de referencia



java.net

- El paquete de las APIs de java contiene una colección de clase e interfaces que proveen comunicación de bajo nivel
- Este paquete provee soporte para dos protocolos comunes de red (capa de transporte):
  - TCP
  - UDP

TCP

- TCP son las siglas de ‘Transmission Control Protocol’, el cual permite comunicación confiable entre dos aplicaciones
  - Este protocolo asegura que todos los paquetes enviados llegaron a su destino en el mismo orden que se enviaron
  - El que recibió el mensaje envía un reconocimiento de recepción al que envió el mensaje para informar que el paquete llegó correctamente y puede enviar el siguiente
  - TCP se utiliza típicamente con el Protocolo de Internet, conocido comúnmente como TCP/IP

UDP

- UDP son las siglas de ‘User Datagram Protocol’, el cual permite una conexión connection-less, por medio de la cual se envian paquetes de datos a través de las aplicaciones
  - UDP no asegura que los datos enviados llegan en el mismo orden de los que se enviaron
  - El que recibe no envia ningún reconocimiento de recepción
  - Por esta razón, este protocolo es más rápido que el TCP
  - Los datos enviados son llamados datagramas

Puerto de comunicación

- Para comunicarse con otra computadora, todo los datos fluyen a través de un solo medio (capa de transmisión)
  - Por ejemplo el cable RJ45 en la tarjeta de red
  - Los datos recibidos pueden tener diferentes destinos en la computadora
  - Los puertos de comunicación son utilizados para direccionar datos a sus programas asociados (capa de aplicación)

Puerto de comunicación

- Los puertos tienen un número de 16 bits
  - Entonces los valores son de 0 a 65535
  - Los primeros 1024 bits son reservados por el sistema. Cada uno de ellos corresponde a un servicio específico, pro ejemplo:

Port 21 : FTP (File Transfer Protocol)
Port 23 : Telnet
Port 80 : HTTP (Hyper Text Transfer Protocol)
Port 119 : Usenet

Puerto de comunicación

- Un puerto de comunicación debe ser usado para una sola conexión
  - Un único servicio/aplicación se puede conectar a un puerto y un único servicio/aplicación puede escuchar en un puerto

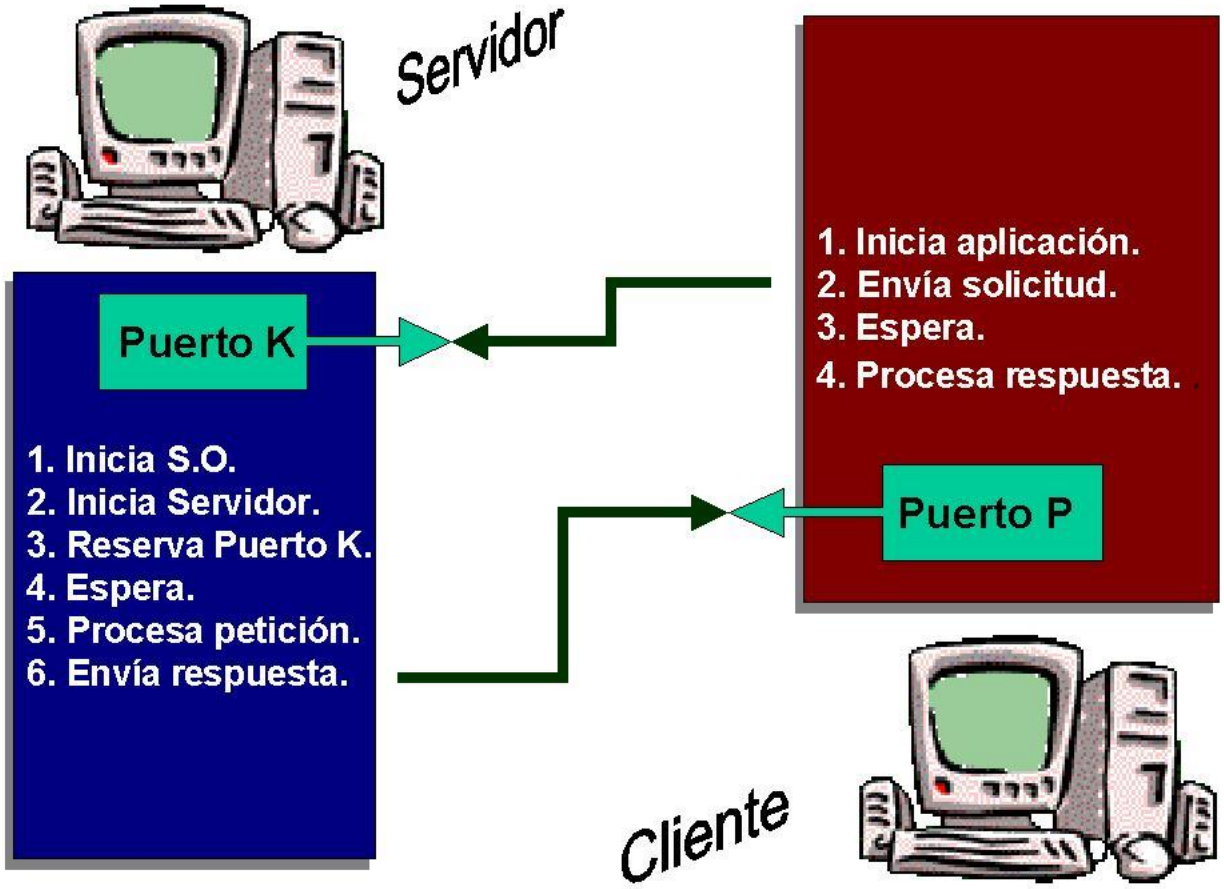
Socket

- Un socket es una interfaz de software que establece una comunicación bidireccional entre un servidor y uno o más clientes
- El socket asocia el servidor con un puerto específico en hardware, en donde esta corriendo
  - De tal forma que cualquier cliente, en cualquier lugar en la red, con un socket asociado a ese mismo puerto, pueda comunicarse con el servidor

Socket

- Un servidor provee recursos a una red de clientes
- Los clientes envían solicitudes al servidor y el servidor responde a las solicitudes

Modelo cliente-servidor



Transmisión basada en flujos, orientada a la conexión

»

Transmisión basada en flujos

- La trasmisión orientada a la conexión es como el sistema telefónico en el que se marca y recibe una conexión al teléfono de una persona con quién se desea comunicar
  - La conexión se mantiene todo el tiempo, incluso no se esté hablando

Creación de un Servidor utilizando sockets de flujo

- Paso 1
  - Crear un objeto ServerSocket

  Ejemplo:

  ServerSocket servidor = new ServerSocket(puerto, longitud cola);

  puerto: especifica el número de puerto. A menudo se le conoce como punto de negociación (handshake)

  longitud cola: especifica el número máximo de clientes que pueden conectarse al servidor

ServerSocket servidor = new ServerSocket(12345, 100);

Creación de un Servidor utilizando sockets de flujo

- Paso 2
  - El manejo de cada conexión de los clientes se realiza a través de un objeto Socket
  - El servidor escucha indefinidamente (bloquea) para esperar a que un cliente se conecte
  - Para escuchar una conexión de un cliente, el programa llama al método accept

```
Socket conexion = servidor.accept();
```

Creación de un Servidor utilizando sockets de flujo

- Paso 3
  - En este paso se obtienen los objetos OutputStream e InputStream que permiten al servidor comunicarse con el cliente
  - Utilizando objetos del tipo ObjectOutputStream y ObjectInputStream se pueden manejar objetos de varios tipos

```
salida = new ObjectOutputStream(
conexion.getOutputStream() );

entrada = new ObjectInputStream(
conexion.getInputStream() );
```

Creación de un Servidor utilizando sockets de flujo

- Paso 4
  - En el paso 4 es la fase de procesamiento, en la cual el servidor y el cliente se comunican a través de los objetos OutputStream e InputStream, pro ejemplo:

```
salida.writeObject( "SERVIDOR>>> " + mensaje );
```

Creación de un Servidor utilizando sockets de flujo

- Paso 5
  - Cuando se completa la transmisión , el servidor cierra la conexión invocando al método close en los flujos y en el objeto socket

```
try {
        salida.close();
        entrada.close();
        conexion.close();
    }
    catch( IOException ioException ) {
        ioException.printStackTrace();
    }
```

Creación de un Cliente utilizando sockets de flujo

- Paso 1
  - Se crea un objeto Socket para conectarse al servidor

  Socket cliente = new Socket(dirIPServidor, puerto);

  Cuando el sistema no puede resolver la dirección del servidor ocurre una excepción *UnknownHostException*

Creación de un Cliente utilizando sockets de flujo

- Paso 2
  - El cliente utiliza los métodos getInputStream y getOutputStream de la clase Socket para obtener las referencias a los objetos InputStream y OutputStream de Socket

Creación de un Cliente utilizando sockets de flujo

- Paso 3
  - Esta es la fase de procesamiento, en la cual el cliente y el servidor se comunican a través de los objetos InputStream y OutputStream

Creación de un Cliente utilizando sockets de flujo

- Paso 4
  - En el paso 4, el cliente cierra la conexión cuando se completa la transmisión invocando al método close en los flujos y en el objeto Socket

```
try {
        salida.close();
        entrada.close();
        cliente.close();
    }
    catch( IOException ioException ) {
        ioException.printStackTrace();
    }
}
```