CAPÍTULO 3: PROGRAMACIÓN DE COMUNICACIONES EN RED

Programación de Servicios y Procesos

ÍNDICE

- Conceptos básicos: Comunicación entre aplicaciones
- Protocolos de comunicaciones: IP, TCP, UDP
- Sockets
- Modelos de Comunicaciones

Conceptos básicos

- Muchos sistemas computacionales de la actualidad siguen el modelo de **computación distribuida**.
- Aplicaciones a través de Internet, móviles, etc.
- La mayoría de superordenadores modernos son sistemas distribuidos.

Sistema distribuido

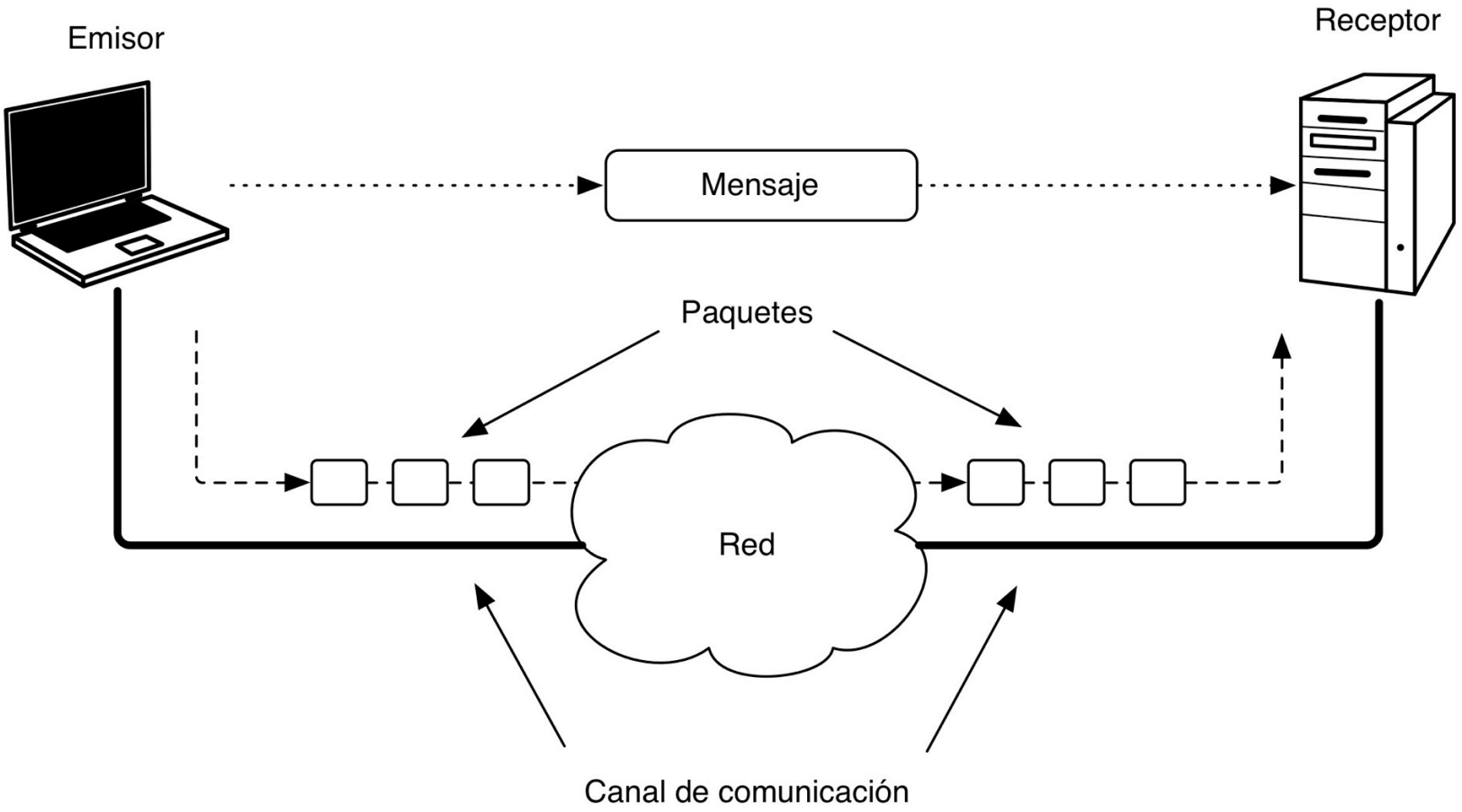
- Está formado por **más de un elemento computacional** distinto e independiente (un procesador dentro de una máquina, un ordenador dentro de una red, etc), que no comparte memoria con el resto.
- Los elementos que forman el sistema distribuido **no están sincronizados**: No hay reloj común.
- Los elementos que forman el sistema están **conectados a una red de comunicaciones**.

Alberto Sánchez Campos
Jesús Montes Sánchez

Comunicación entre aplicaciones

- Es la base del funcionamiento de todo sistema distribuido.
- En el proceso de comunicación se distingue:
  - Mensaje
  - Emisor
  - Receptor
  - Paquete
  - Canal de comunicación
  - Protocolo de comunicaciones

Comunicación entre aplicaciones



Alberto Sánchez Campos
Jesús Montes Sánchez

Ra-Ma®

Comunicación entre aplicaciones

- Mensaje:
  - Es la información que se intercambia entre las aplicaciones que se comunican.
- Emisor:
  - Es la aplicación que envía el mensaje.
- Receptor:
  - Es la aplicación que recibe el mensaje.
- Paquete:
  - Es la unidad básica de información que intercambian dos dispositivos de comunicación.

Alberto Sánchez Campos
Jesús Montes Sánchez

Comunicación entre aplicaciones

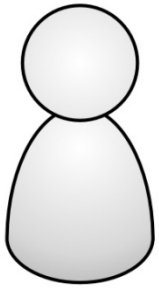
- Canal de comunicación:
  - Es el medio por el que se transmiten los paquetes, que conecta el emisor con el receptor.
- Protocolo de comunicaciones:
  - Es el conjunto de reglas que fijan cómo se deben intercambiar paquetes entre los diferentes elementos que se comunican entre sí.

Alberto Sánchez Campos
Jesús Montes Sánchez

Protocolos de comunicaciones

- Para que las diferentes aplicaciones que forman un sistema distribuido puedan comunicarse, debe existir una serie de mecanismos que hagan posible esa comunicación:
  - Elementos hardware
  - Elemento software
- Todos estos componentes se organizan en lo que se denomina una **jerarquía** o **pila de protocolos**.

Pila de protocolos IP



Usuario

Nivel de aplicación

Nivel de transporte

Nivel de Internet

Nivel de red

Alberto Sánchez Campos
Jesús Montes Sánchez

Pila de protocolos IP

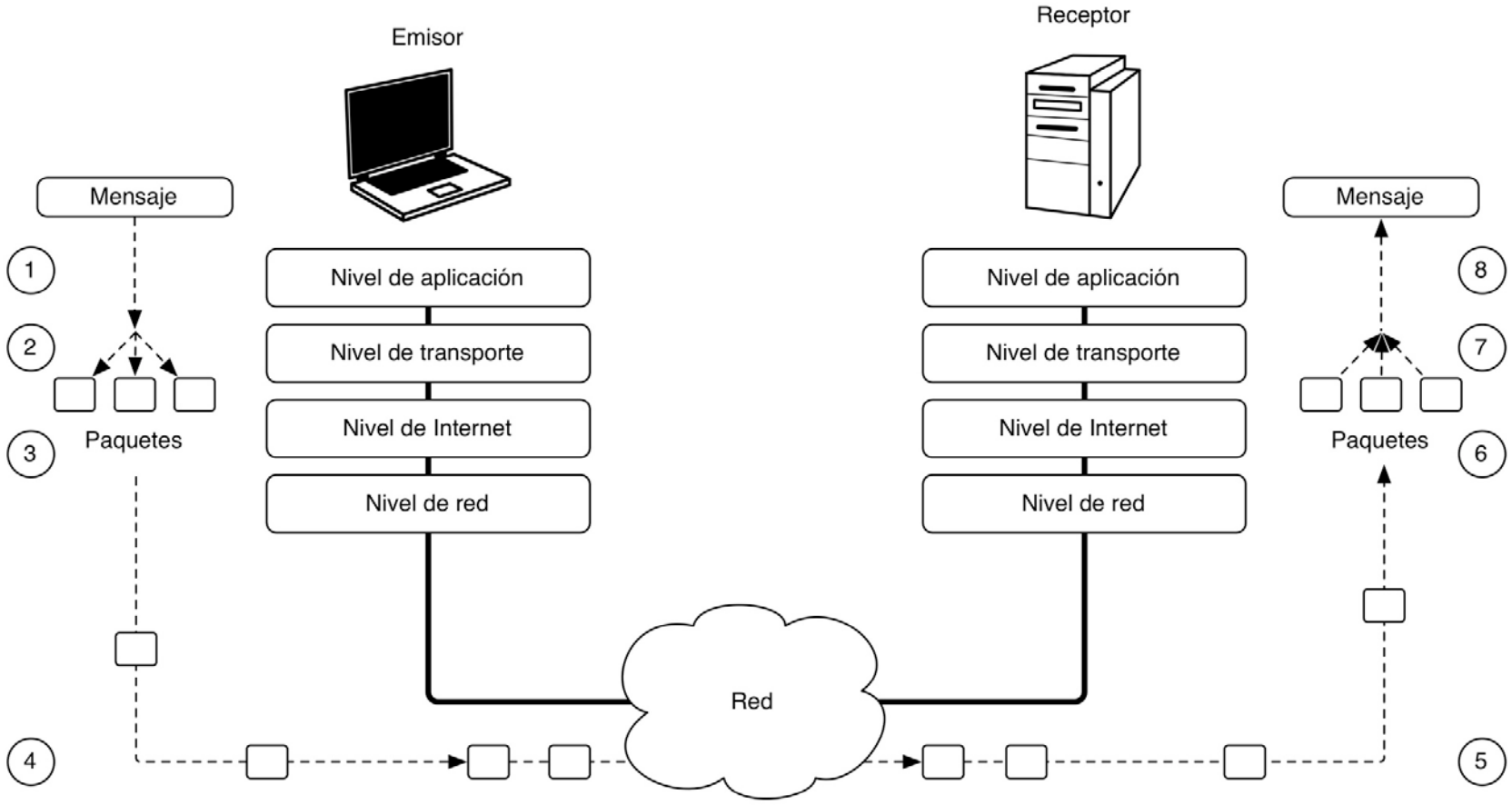
- Nivel de red:
  - Lo componen los elementos hardware de comunicaciones y sus controladores básicos.
  - Se encarga de trasmitir los paquetes de información.
- Nivel de Internet:
  - Lo componen los elementos software que se encargan de dirigir los paquetes por la red, asegurándose de que lleguen a su destino.
  - También llamado **nivel IP**.

Alberto Sánchez Campos
Jesús Montes Sánchez

Pila de protocolos IP

- Nivel de transporte:
  - Lo componen los elementos software cuya función es crear el canal de comunicación, descomponer el mensaje en paquetes y gestionar su transmisión entre el emisor y el receptor.
  - Los dos protocolos de transporte fundamentales: TCP y UDP.
- Nivel de aplicación:
  - Lo componen las aplicaciones que forman el sistema distribuido, que hacen uso de los niveles inferiores para poder transferir mensajes entre ellas

Funcionamiento de la pila de protocolos



Alberto Sánchez Campos
Jesús Montes Sánchez

Ra-Ma®

Protocolo de trasporte TCP

- Garantiza que los datos no se pierden.
- Garantiza que los mensajes llegarán en orden.
- Se trata de un **protocolo orientado a conexión**.

Protocolo orientado a conexión

- Es aquel en que el canal de comunicaciones entre dos aplicaciones permanece abierto durante un cierto tiempo, permitiendo enviar múltiples mensajes de manera fiable por el mismo.
- Opera en tres fases:
  1. Establecimiento de la conexión.
  2. Envío de mensajes.
  3. Cierre de la conexión.
- El ejemplo mas habitual es el protocolo TCP.

Alberto Sánchez Campos
Jesús Montes Sánchez

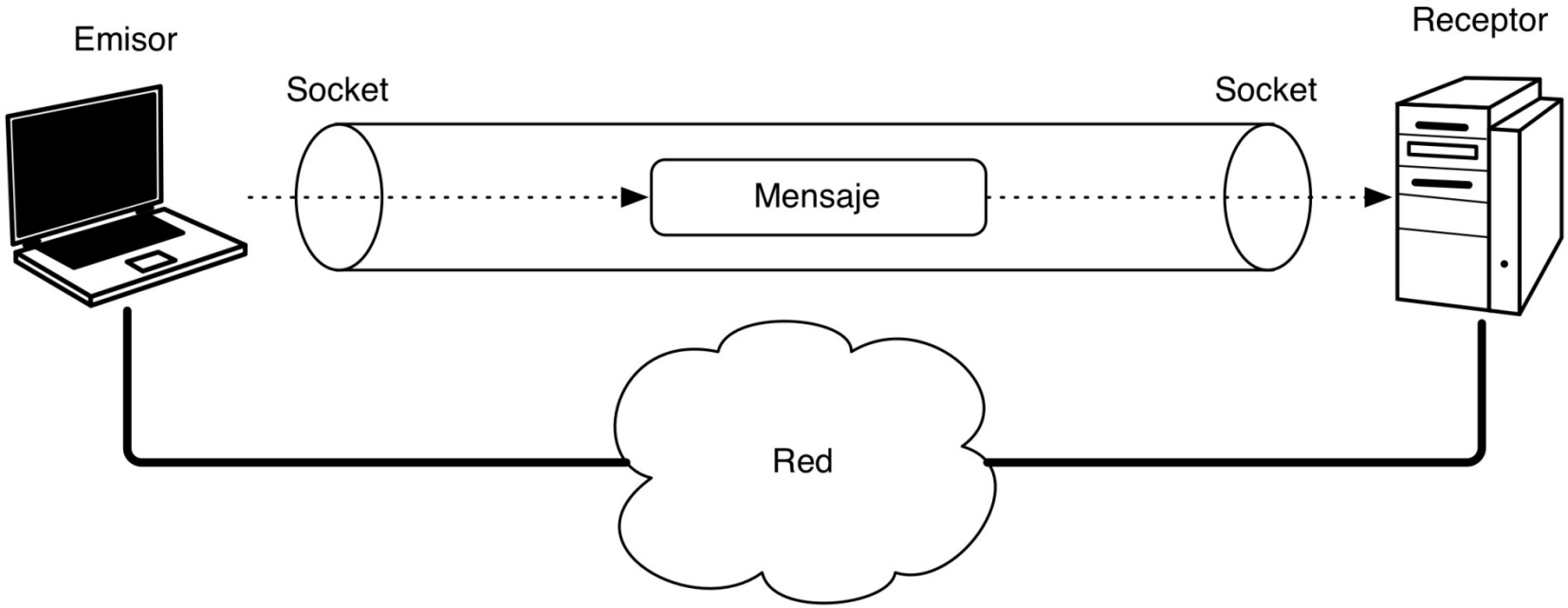
Protocolo de transporte UDP

- **Protocolo NO orientado a conexión**. Esto lo hace más rápido que TCP, ya que no es necesario establecer conexiones, etc.
- No garantiza que los mensajes lleguen siempre.
- No garantiza que los mensajes lleguen en el mismo orden que fueron enviados.
- Permite enviar mensajes de 64 KB **como máximo**.
- En UDP, los mensajes se denominan “datagramas” (*datagrams* en ingles).

Sockets

- Los *sockets* son el mecanismo de comunicación básico fundamental que se usa para realizar transferencias de información entre aplicaciones.
- Proporcionan una abstracción de la pila de protocolos.
- Un *socket* (en inglés, literalmente, un “enchufe”) representa el extremo de un canal de comunicación establecido entre un emisor y un receptor.

Sockets



Alberto Sánchez Campos
Jesús Montes Sánchez

Direcciones IP y puertos

- Una dirección IP es un número que identifica de forma única a cada máquina de la red, y que sirve para comunicarse con ella.
- Un puerto es un número que identifica a un *socket* dentro de una máquina.

Alberto Sánchez Campos
Jesús Montes Sánchez

Sockets stream

- Son orientados a conexión.
- Cuando operan sobre IP, emplean TCP.
- Un *socket stream* se utiliza para comunicarse siempre con el mismo receptor, manteniendo el canal de comunicación abierto entre ambas partes hasta que se termina la conexión.
- Una parte ejerce la función de **proceso cliente** y otra de **proceso servidor**.

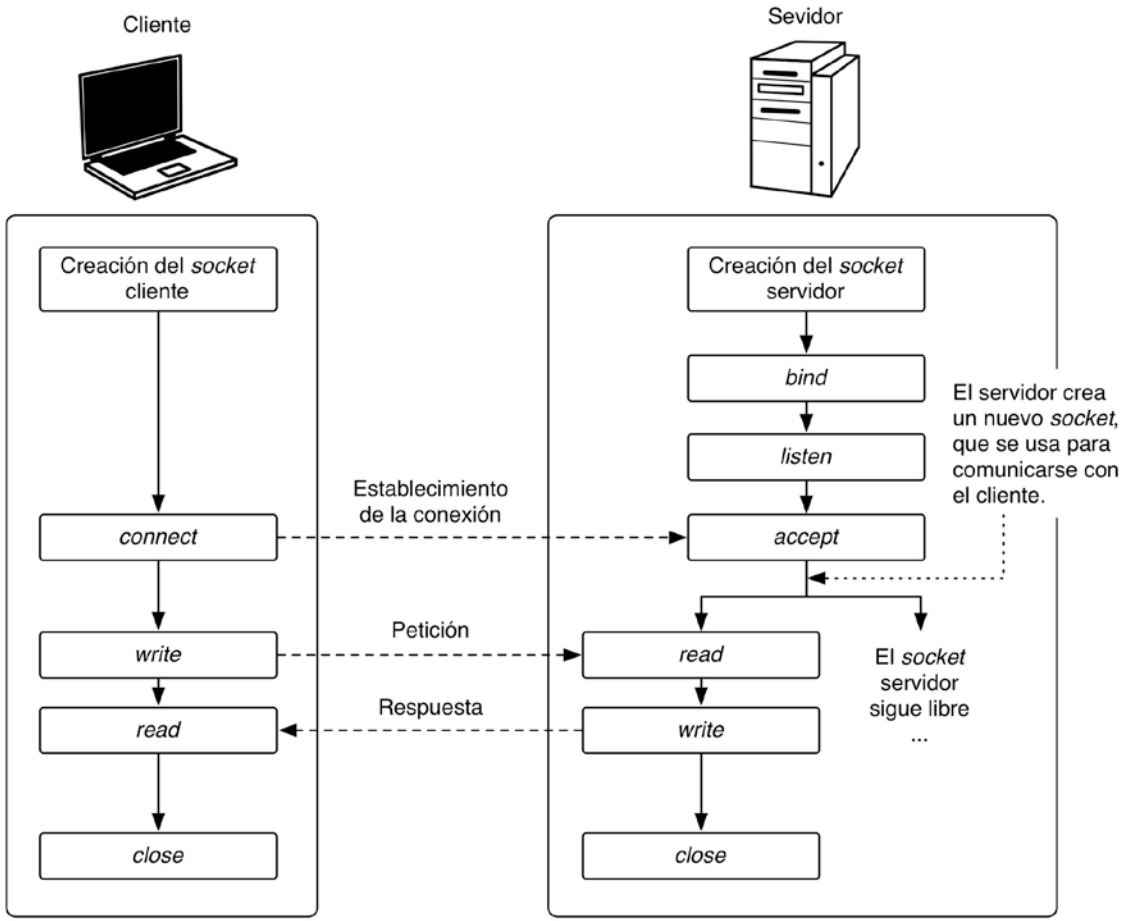
Sockets stream

- Proceso cliente:
  1. Creación del socket.
  2. Conexión del socket (*connect*).
  3. Envío y recepción de mensajes.
  4. Cierre de la conexión (*close*).

Sockets stream

- Proceso servidor:
  1. Creación del socket.
  2. Asignación de dirección y puerto (*bind*).
  3. Escucha (*listen*).
  4. Aceptación de conexiones (*accept*). Esta operación implica la creación de un nuevo socket, que se usa para comunicarse con el cliente que se ha conectado.
  5. Envío y recepción de mensajes.
  6. Cierre de la conexión (*close*).

Sockets stream



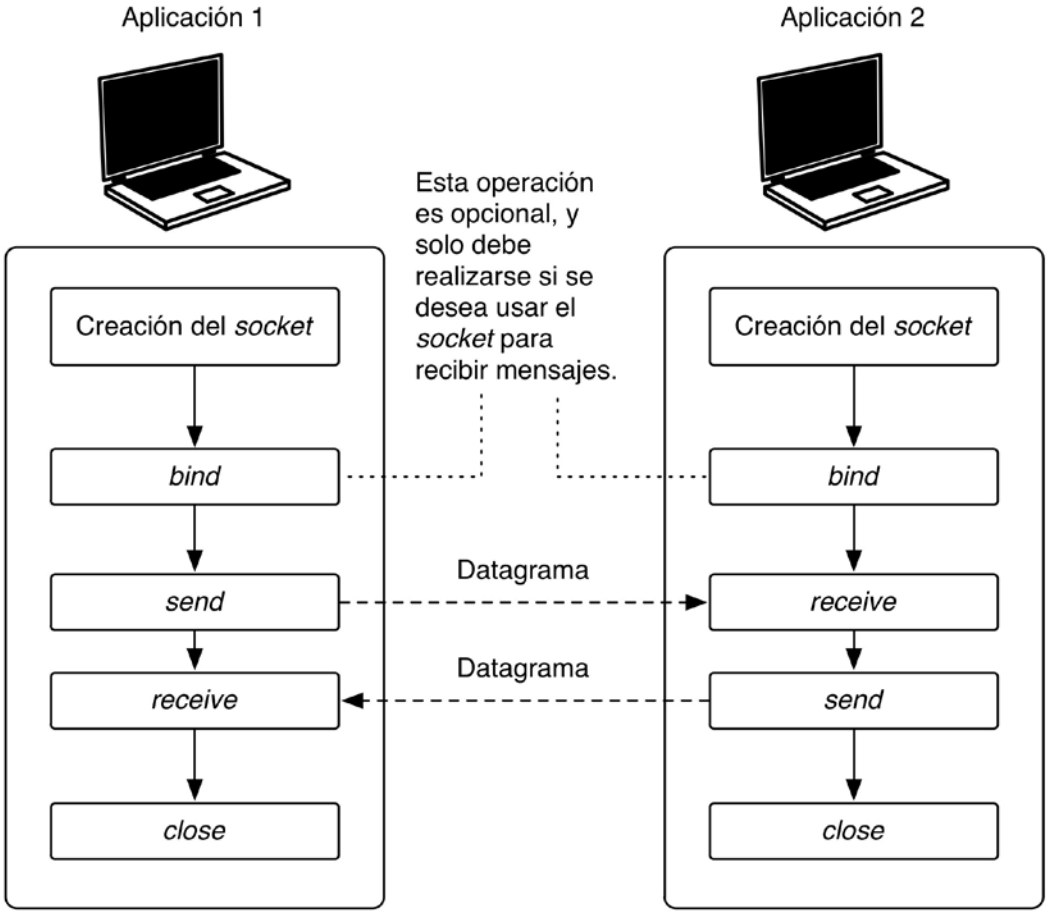
Alberto Sánchez Campos
Jesús Montes Sánchez

Ra-Ma®

Sockets datagram

- Son no orientados a conexión.
- Cuando operan sobre IP, emplean UDP.
- Cuando se usan *sockets datagram* no existe diferencia entre proceso servidor y proceso cliente.
- Pasos para enviar mensajes:
  - Creación del socket.
  - Asignación de dirección y puerto (*bind*). Solo necesaria para poder recibir mensajes.
  - Envío y/o recepción de mensajes.
  - Cierre del socket.

Sockets datagram



Alberto Sánchez Campos
Jesús Montes Sánchez

Ra-Ma®

Programación con sockets

- *java.net.Socket*, para la creación de *sockets stream* cliente.
- *java.net.ServerSocket*, para la creación de *sockets stream* servidor.
- *java.net.DatagramSocket*, para la creación de *sockets datagram*.

Ejemplo con sockets stream (1)

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

public class ClienteSocketStream {

   public static void main(String[] args) {
      try {
         Socket clientSocket = new Socket();
         InetSocketAddress addr = new InetSocketAddress("localhost", 5555);
         clientSocket.connect(addr);
         InputStream is = clientSocket.getInputStream();
         OutputStream os = clientSocket.getOutputStream();
         String mensaje = "mensaje desde el cliente";
         os.write(mensaje.getBytes());
         clientSocket.close();

      } catch (IOException e) {
         e.printStackTrace();
      }
   }
}
```

Alberto Sánchez Campos
Jesús Montes Sánchez

Ejemplo con sockets stream (2)

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

public class ServeridorSocketStream {

   public static void main(String[] args) {
      try {

         ServerSocket serverSocket = new ServerSocket();
         InetSocketAddress addr = new InetSocketAddress("localhost", 5555);
         Socket newSocket = serverSocket.accept();
         InputStream is = newSocket.getInputStream();
         OutputStream os = newSocket.getOutputStream();
         byte[] mensaje = new byte[25];
         is.read(mensaje);
         newSocket.close();
         serverSocket.close();
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
}
```

Alberto Sánchez Campos
Jesús Montes Sánchez

Ejemplo con sockets datagram

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

public class EmisorDatagram {

   public static void main(String[] args){
      try {
         DatagramSocket datagramSocket = new DatagramSocket();
         String mensaje = "mensaje desde el emisor";
         InetAddress addr = InetAddress.getByName("localhost");
         DatagramPacket datagrama = new DatagramPacket(mensaje.getBytes(),
                                                       mensaje.getBytes().length, addr, 5555);
         datagramSocket.send(datagrama);
         datagramSocket.close();
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
}
```

Alberto Sánchez Campos
Jesús Montes Sánchez

Modelos de comunicaciones

- Los *sockets* son una herramienta básica para enviar y recibir mensajes.
- A la hora de desarrollar aplicaciones distribuidas debemos tener en cuenta aspectos de más alto nivel.
- Dependiendo de cuál sea el propósito de nuestra aplicación, y cómo vaya a funcionar internamente, deberemos escoger un modelo de comunicaciones distinto.

Modelos de comunicaciones

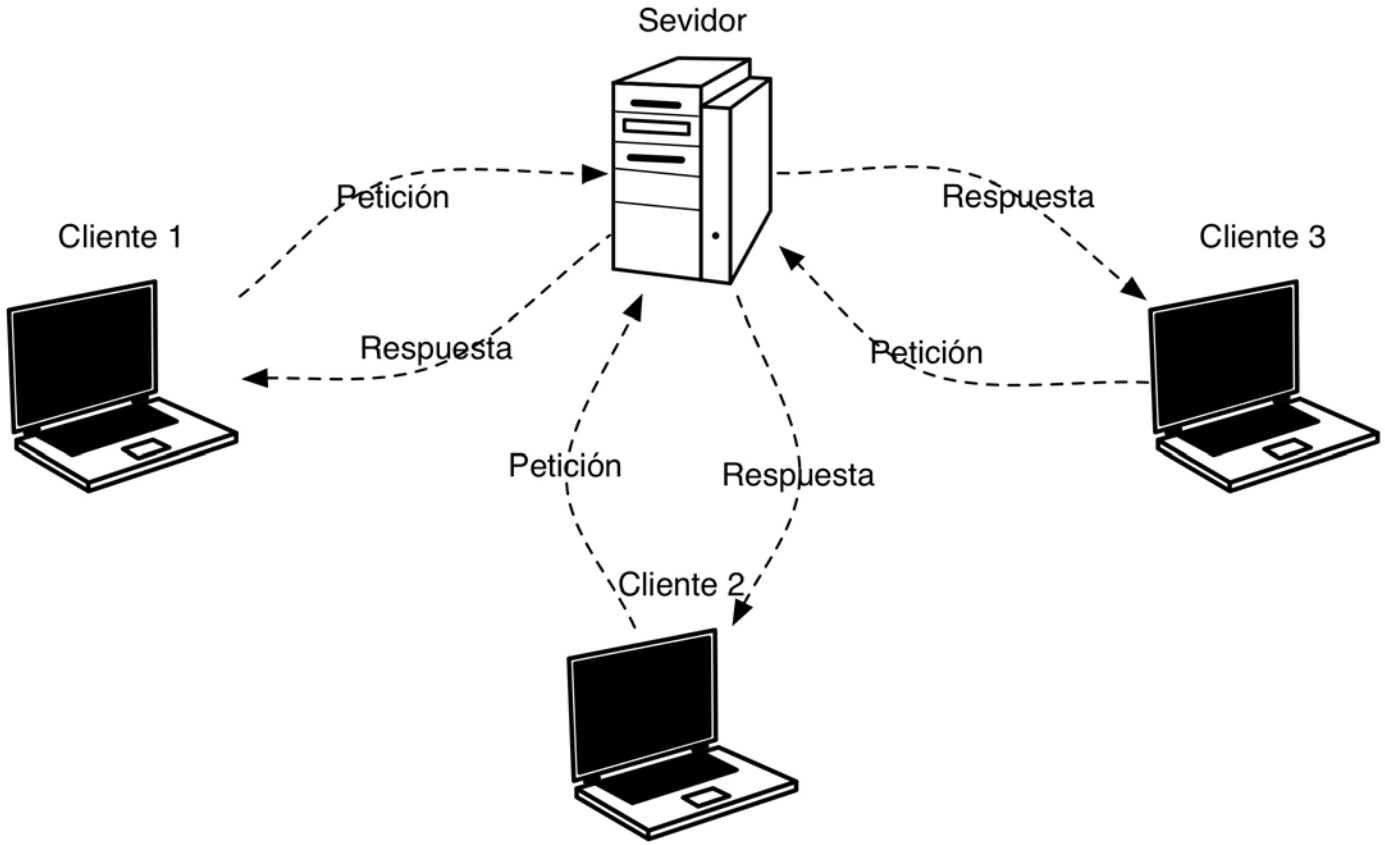
- Un **modelo de comunicaciones** es una arquitectura general que especifica cómo se comunican entre sí los diferentes elementos de una aplicación distribuida.
- Un modelo de comunicaciones normalmente define aspectos como cuántos elementos tiene el sistema, qué función realiza cada uno, etc.
- Los modelos más usados en la actualidad son:
  - *Cliente/servidor*.
  - *Comunicación en grupo.*

Modelo cliente/servidor

- El más sencillo de los comúnmente usados en la actualidad.
- En este modelo, un proceso central, llamado *servidor*, ofrece una serie de servicios a uno o más procesos *cliente*.
- El proceso *servidor* debe estar alojado en una máquina fácilmente accesible en la red, y conocida por los *clientes*.
- Cuando un *cliente* requiere sus servicios, se conecta con el *servidor*, iniciando el proceso de comunicación.

Alberto Sánchez Campos
Jesús Montes Sánchez

Modelo cliente/servidor



Alberto Sánchez Campos
Jesús Montes Sánchez

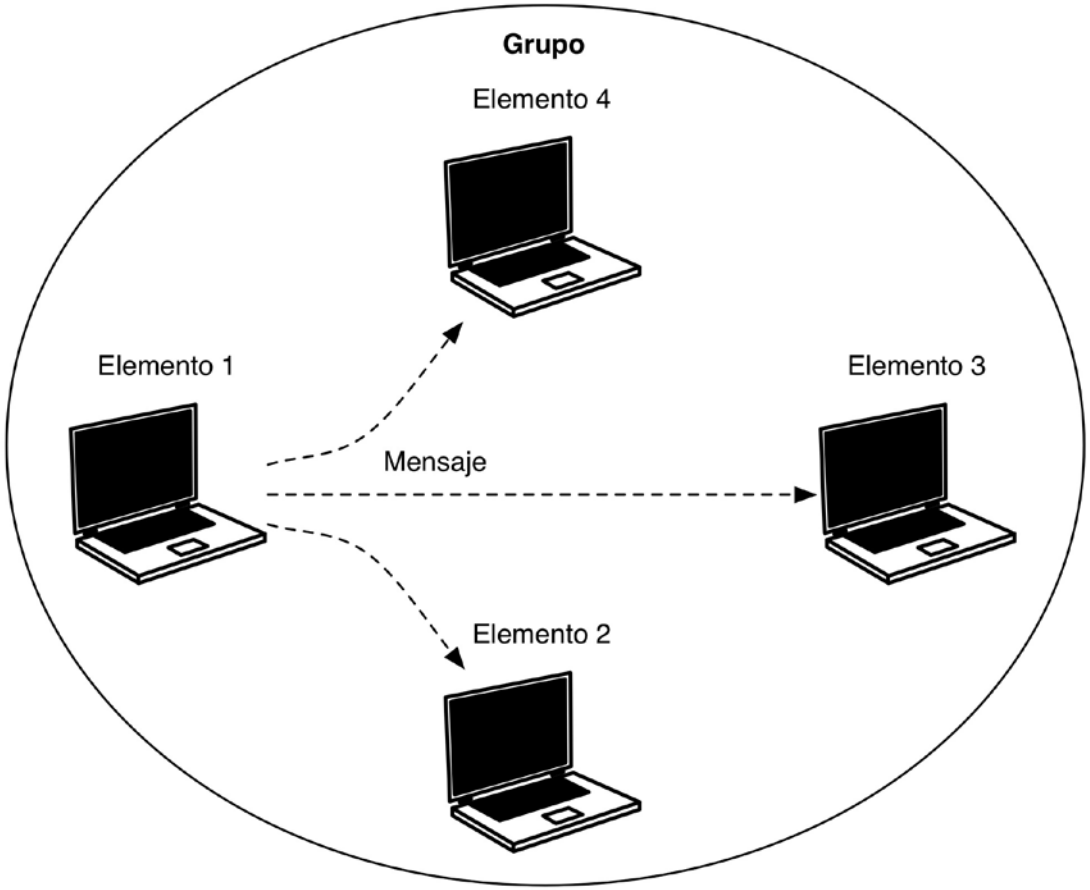
Ra-Ma®

Modelo de comunicación en grupo

- Es la alternativa más común al modelo cliente/servidor.
- En este modelo no existen roles diferenciados.
- En la comunicación en grupo existe un conjunto de dos o más elementos (procesos, aplicaciones, etc.) que cooperan en un trabajo común.
- A este conjunto se le llama *grupo*, y los elementos que lo forman se consideran todos iguales, sin roles ni jerarquías definidas.
- Los mensajes se transmiten mediante *radiado*.

Alberto Sánchez Campos
Jesús Montes Sánchez

Modelo de comunicación en grupo



Alberto Sánchez Campos
Jesús Montes Sánchez

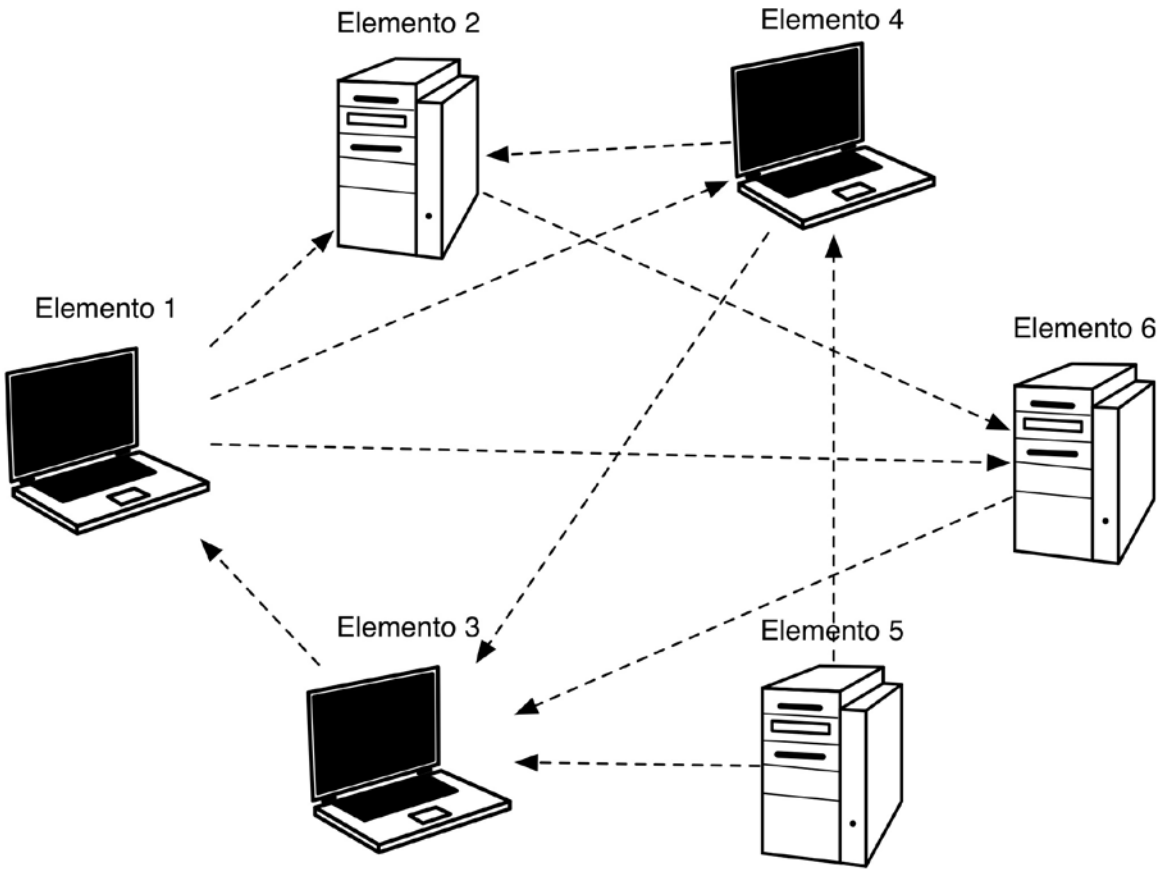
Modelos híbiridos

- Las aplicaciones distribuidas más avanzadas suelen tener requisitos de comunicaciones muy complejos, que requieren de modelos de comunicaciones sofisticados.
- En muchos casos, los modelos de comunicaciones reales implementados en estas aplicaciones mezclan conceptos del modelo cliente/servidor y la comunicación en grupo, dando lugar a enfoques híbridos, como las redes *peer-to-peer* (P2P).

Redes *peer-to-peer*

- Una red P2P está formada por un grupo de elementos distribuidos que colaboran con un objetivo común.
- Cualquier elemento puede desempeñar los roles de *servidor* o *cliente*, como si de un modelo cliente/servidor se tratase.
- Las redes P2P puedan ofrecer servicios de forma similar al modelo cliente/servidor.
- Cualquier aplicación puede conectarse a la red como un *cliente*, localizar un *servidor* y enviarle una petición.
- Si permanece en la red P2P, con el tiempo ese mismo *cliente* puede hacer a su vez de *servidor* para otros elementos de la red.

Redes *peer-to-peer*



Alberto Sánchez Campos
Jesús Montes Sánchez

Ra-Ma®