03. Programación de comunicaciones en red

Programación de Servicios y procesos - 2º DAM

Luis del Moral Martínez

versión 20.10

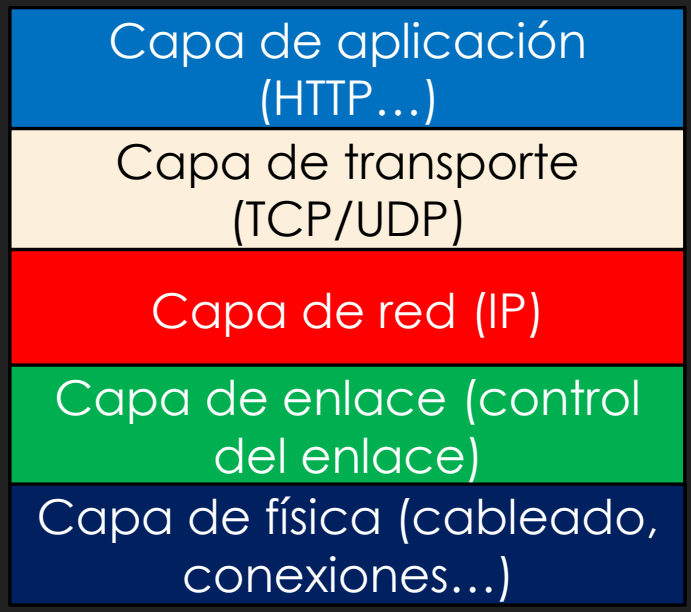
Contenidos del tema

1. **Programación de comunicaciones en red**
   - 1.1 Clases para comunicaciones en red
   - 1.2 Concepto de Socket
   - 1.3 Tipos de Sockets
   - 1.4 Socket orientado a la conexión TCP
   - 1.5 Socket no orientado a la conexión UDP
   - 1.6 Multicast (envío a múltiples destinos)
   - 1.7 Envío de objetos a través de Sockets
   - 1.8 Conexión de múltiples clientes

1.1 Clases para comunicaciones en red

Familia de protocolos TCP/IP

- La familia de **protocolos TCP/IP** permite la comunicación entre dos ordenadores
- Tiene **cinco capas** o niveles de abstracción:
  - **Aplicación**: aplicaciones para los usuarios
  - **Transporte**: suministra servicio de conexión extremo a extremo
  - **Red**: selecciona la mejor ruta para enviar paquetes por la red
  - **Enlace**: interfaz con la red local. Recibe datagramas de red
  - **Física**: cableado e interconexiones



Capas TCP/IP

1.1 Clases para comunicaciones en red

Protocolos TCP y UDP

- Los equipos conectados a Internet se comunican entre sí usando los protocolos **TCP** y **UDP**:
  - **Protocolo TCP**
    - Basado en la conexión
    - Garantiza que los datos llegarán en el mismo orden que fueron enviados
  - **Protocolo UDP**
    - No está basado en la conexión
    - Envía paquetes de datos independientes (datagramas)
    - El orden no es importante y no se garantiza la recepción de la información

1.1 Clases para comunicaciones en red

Los puertos

- Los protocolos **TCP** y **UDP** usan puertos para enviar y recibir datos
- Un ordenador tiene una conexión física con una red
- Los datos llegan a través de esa conexión, pero pueden dirigirse a diferentes aplicaciones
- De esta forma, una comunicación **TCP** vincula un **Socket** a un número de puerto concreto
- Los primeros 1024 puertos están reservados: 80 (http), 443(https)…

1.1 Clases para comunicaciones en red

El paquete java.net

- El paquete **java.net** contiene clases e interfaces para implementar aplicaciones de red:
  - **Clase URL**: representa un puntero **URL** (*Uniform Resource Locator*) hacia ***Internet***
  - **Clase URLConnection**: maneja operaciones más complejas sobre URLs
  - **Clases ServerSocket y Socket**: dan soporte a sockets TCP (cliente-servidor)
  - **Clases DatagramSocket, MulticasSocket y DatagramPacket**: dan soporte a la comunicación con UDP
  - **Clase InetAddress**: representa a las direcciones de Internet
- Ahora analizaremos con detalle y usando ejemplos cada una de las clases

1.1 Clases para comunicaciones en red

El paquete java.net

- El paquete **java.net** contiene clases e interfaces para implementar aplicaciones de red:
  - **Clase URL**: representa un puntero **URL** (*Uniform Resource Locator*) hacia ***Internet***
  - **Clase URLConnection**: maneja operaciones más complejas sobre URLs
  - **Clases ServerSocket y Socket**: dan soporte a sockets TCP (cliente-servidor)
  - **Clases DatagramSocket, MulticasSocket y DatagramPacket**: dan soporte a la comunicación con UDP
  - **Clase InetAddress**: representa a las direcciones de Internet
- Ahora analizaremos con detalle y usando ejemplos cada una de las clases

1.1 Clases para comunicaciones en red

Clase InetAddress

- Esta clase representa una dirección IP (Internet Protocol)
- Tiene dos subclases: **Inet4Address** (direcciones IPv4) e **Inet6Address** (direcciones IPv6)
- Si el host no se resuelve se lanza la excepción **UnknownHostException**
- **Métodos de la clase InetAddress**: enlace

1.1 Clases para comunicaciones en red

Clase URL

- Esta clase representa un puntero a un recurso de la Web (fichero, directorio, consulta a BBDD…)
- En general, la URL que localiza recursos usando los protocolos HTTP o HTTPS tiene varias partes:
- **http://host[:puerto][/directoriodelservidor][?argumentos]**
  - **host**: nombre de la máquina en la que reside el recurso
  - **[:puerto]**: número de puerto en el que el servidor escucha las peticiones (puerto 80 para **HTTP**)
  - **[/directorioservidor]**: ruta donde se encuentra el recurso en el sistema de archivos del servidor
  - **[?argumentos]**: parámetros adicionales para el recurso (**POST** o **GET** en **PHP**, por ejemplo)

1.1 Clases para comunicaciones en red

Clase URL

- Si la URL no está bien construida se lanza la excepción **MalformedURLException**
- **Métodos de la clase URL**: enlace

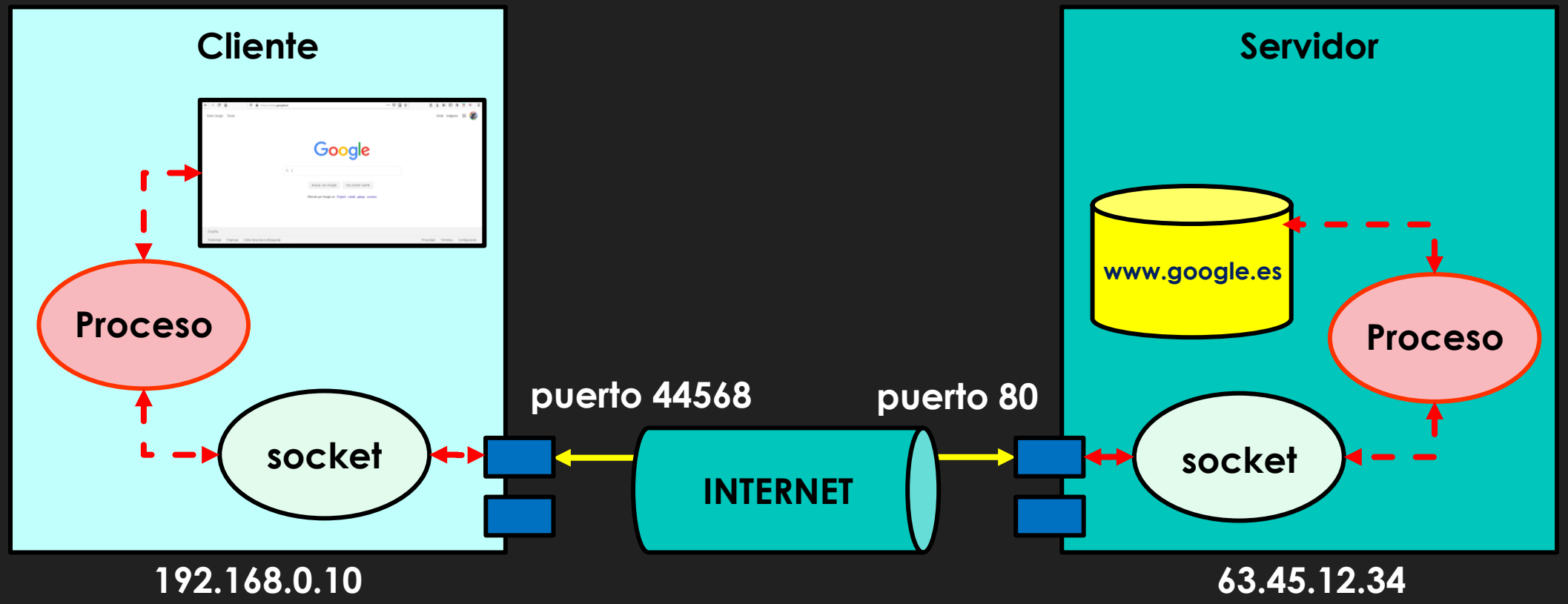
1.2 Concepto de Socket

¿Qué es un Socket?

- Los protocolos TCP y UDP emplean los Sockets como **puntos de comunicación** entre procesos
- Cada **Socket** tiene dos elementos:
  - Dirección IP del host en la que se ejecuta la aplicación
  - Puerto local vinculado a la aplicación
- Los mensajes que se envíen a esa dirección y ese puerto llegarán al proceso receptor
- Generalmente, los servidores tienen puertos acordados (80, 443…)
- Los puertos de los clientes pueden ser arbitrarios

1.2 Concepto de Socket

¿Qué es un Socket?



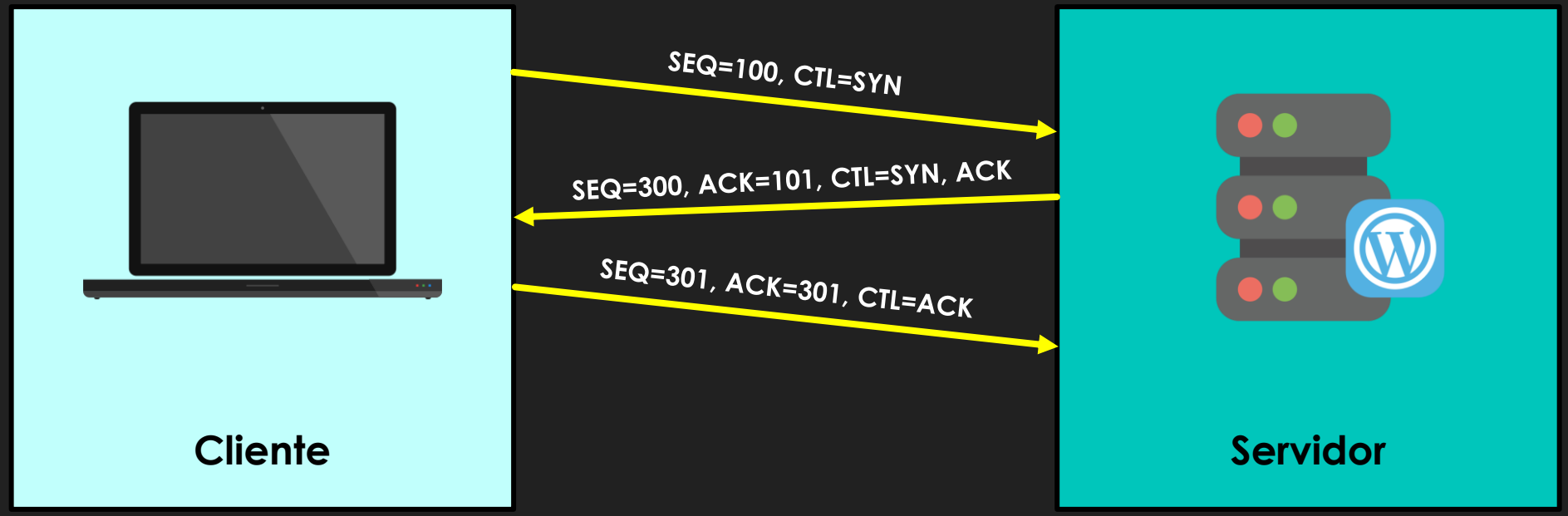
1.2 Concepto de Socket

¿Cómo funciona un Socket?

- El puerto de destino identifica hacia qué aplicación o proceso se dirigen los datos
- El programa servidor se ejecuta en una máquina y puertos conocidos
- El programa cliente conoce el nombre de la máquina (DNS) y el puerto de destino
- El cliente realiza una solicitud de conexión a través del puerto para acceder al servicio
- Como el cliente se identifica ante el servidor, tiene que usar un puerto local de su máquina
- El puerto local del cliente puede ser cualquiera
- Los primeros 1024 puertos están reservados. **Más información**: enlace

1.2 Concepto de Socket

Estableciendo una conexión (TCP/IP handshake)



1.4 Socket orientado a la conexión TCP

Socket orientado a la conexión (TCP)

- Utiliza el protocolo TCP para establecer la comunicación entre las aplicaciones
- La conexión es confiable y se garantiza la entrega de los paquetes y el orden de envío
- TCP utiliza acuse de recibo (ACK) de los mensajes, de forma que se reenvían los fallidos
- Los procesos que se comunican establecen una conexión mediante un **stream**
- Un stream es una secuencia ordenada de bytes que fluye en ambas direcciones
- Cuando se ha establecido la conexión, los dos procesos usan el stream para comunicarse
- **Clases en Java**: **Socket** (para el cliente) y **ServerSocket** (para el servidor): enlace, enlace

1.4 Socket orientado a la conexión TCP

Gestión de Sockets TCP

1. El programa **servidor** crea un socket con **ServerSocket**(puerto) y espera con **accept**()
2. El **cliente** solicita la conexión
3. El **cliente** establece la conexión con **Socket**(host, puerto)
4. El cliente y el servidor se comunicación con **InputStream** y **OutputStream**
5. Los Sockets se cierran (¡El orden de cierre es importante!)
   - Primero se cierran los **Streams**
   - Después se cierra el propio **Socket**

1.5 Socket no orientado a la conexión UDP

Socket orientado a la conexión (UDP)

- Utiliza el protocolo UDP para establecer la comunicación entre las aplicaciones
- La conexión no es confiable y no se garantiza la entrega de los paquetes ni el orden de envío
- Se envían datagramas sin establecer ninguna conexión
- Los procesos que quieren enviar o recibir se vinculan a una IP y un puerto mediante un conector
- UDP se emplea cuando se requiere una entrega rápida
- Se suele emplear en aplicaciones de transmisión multimedia (telefonía, videoconferencias…)
- **Clases en Java**: **DatagramSocket** y **DatagramPacket**: enlace, enlace

1.5 Socket no orientado a la conexión UDP

Gestión de Sockets UDP

1. El **servidor** crea un socket asociado a un puerto local para escuchar peticiones de clientes
2. El **cliente** crea un socket para comunicarse con el servidor
   - Para enviar datagramas necesita conocer la IP y el puerto por el que escucha el servidor
   - Para enviar la petición en forma de datagrama usará el método **send**()
3. El servidor recibe las peticiones con el método **receive**()
   - En el datagrama se incluye, además del mensaje, el IP y puerto origen
4. El cliente recibe la respuesta del servidor mediante el método **receive**()
5. El servidor permanece a la espera de más peticiones

1.6 Multicast (envío a múltiples destinos)

Enviando información a múltiples destinos

- La clase **MulticastSocket** permite enviar paquetes a múltiples destinos a la vez
- Para recibir estos mensajes se establece primero un grupo Multicast
- Estas direcciones IP que forman el grupo comparten el mismo puerto
- Se utilizan direcciones de **clase D** (224.0.0.0 – 239.255.255.255)
- **Más información sobre la clase MulticasSocket**: enlace

1.7 Envío de objetos a través de Sockets

Enviando objetos en red a través de TCP

- Los Streams también permiten enviar objetos a través de la red
- Las clases **ObjectInputStream** y **ObjectOutputStream** permiten enviar objetos en los flujos TCP
- Para leer y escribir objetos se emplean los métodos **readObject**() y **writeObject**()
- **Clases en Java**: **ObjectInputStream** y **ObjectOutputStream**: enlace, enlace

1.7 Envío de objetos a través de Sockets

Enviando objetos en red a través de UDP

- En Sockets UDP se pueden enviar objetos con **ByteArrayOutputStream** y **ByteArrayInputStream**
- El objeto debe ser convertido a un array de bytes
- Después el array de bytes se envía con un datagrama
- **Clases en Java**: **ByteArrayOutputStream** y **ByteArrayInputStream** : enlace, enlace

1.7 Envío de objetos a través de Sockets

Enviando objetos en red a través de UDP

```
// Paso 1 - Convertir objeto a byte
ByteArrayOutputStream bs = new ByteArrayOutputStream();
ObjectOutputStream out = new ObjectOutputStream (bs);
out.writeObject(persona); // Se escribe el objeto en el stream
out.close();
byte[] bytes = bs.toByteArray();
```

1.7 Envío de objetos a través de Sockets

Enviando objetos en red a través de UDP

```
// Paso 2 - Recibir datagrama
byte[] recibidos = new byte [1024];
DatagramPacket paqRecibido = new DatagramPacket(recibidos, recibidos.length);
socket.receive(paqRecibido); // Se recibe el datagrama
```

1.7 Envío de objetos a través de Sockets

Enviando objetos en red a través de UDP

```
// Paso 3 – Convertir los Bytes al objeto
ByteArrayInputStream bs = new ByteArrayInputStream(recibidos);
ObjectInputStream in = new ObjectInputStream(bs);
Persona persona = (Persona) in.readObject(); // Se obtiene el objeto
in.close();
```

1.8 Conexión de múltiples clientes

Conexión de múltiples clientes utilizando hilos

- Lo normal es que una aplicación servidor pueda atender a múltiples clientes a la vez
- Para resolver este problema se utilizará la solución multihilo
- Cada hilo servidor se encargará de gestionar las peticiones del cliente
- El servidor principal espera nuevos clientes para lanzar los hilos que los gestionen

Créditos de las imágenes y figuras

Cliparts e iconos

- **Obtenidos mediante la herramienta web IconFinder** (según sus disposiciones):
  - Diapositivas 1, 14
  - Según la plataforma IconFinder, dicho material puede usarse libremente (free comercial use)
  - A fecha de edición de este material, todos los cliparts son free for comercial use (sin restricciones)

Diagramas, gráficas e imágenes

- Se han desarrollado en PowerPoint y se han incrustado en esta presentación
- Todos estos materiales se han desarrollado por el autor
- Para el resto de recursos se han especificado sus fabricantes, propietarios o enlaces
- Si no se especifica copyright con la imagen, entonces es de desarrollo propio o CC0