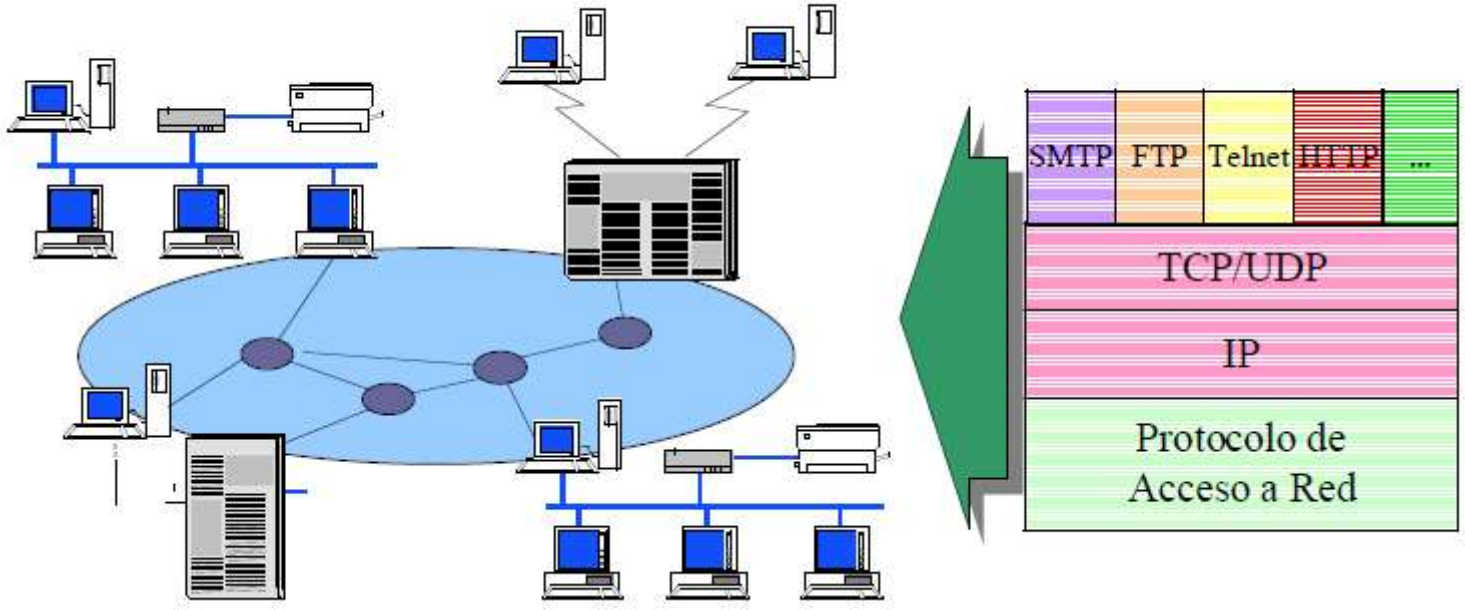
UD4: Programación de Comunicaciones en red

PSP

Ana Alonso

Curso 2024-2025

Internet – Protocolos de Internet



Internet - Internet Protocol (IP)

- Parte superior del nivel de red
- Protocolo datagrama, no orientado a conexión y no fiable
  - No hay recuperación de errores
  - Hay comprobación de errores, y los paquetes IP erróneos se tiran, sin notificar al emisor
  - Soporta fragmentación de datos en paquetes IP (<1400 bytes)
- Direcciones IP
  - Cada máquina tiene una dirección única
  - 4 clases de direcciones IP: A, B, C (network ID+host ID) y D (grupos multicast)
  - Los IP gateways encaminan los paquetes IP según su network ID.

Internet

- Problemas con IP:

- Los paquetes IP no se autentifican: El emisor puede mentir sobre su identidad, y puede enviar paquetes a cualquier máquina o puerto en la red.

  Un sistema puede mentir sobre su dirección de máquina.

- Los mensajes IP no son fiables: Pueden ser retirados en cualquier lugar.

Internet – Protocolos nivel de transporte

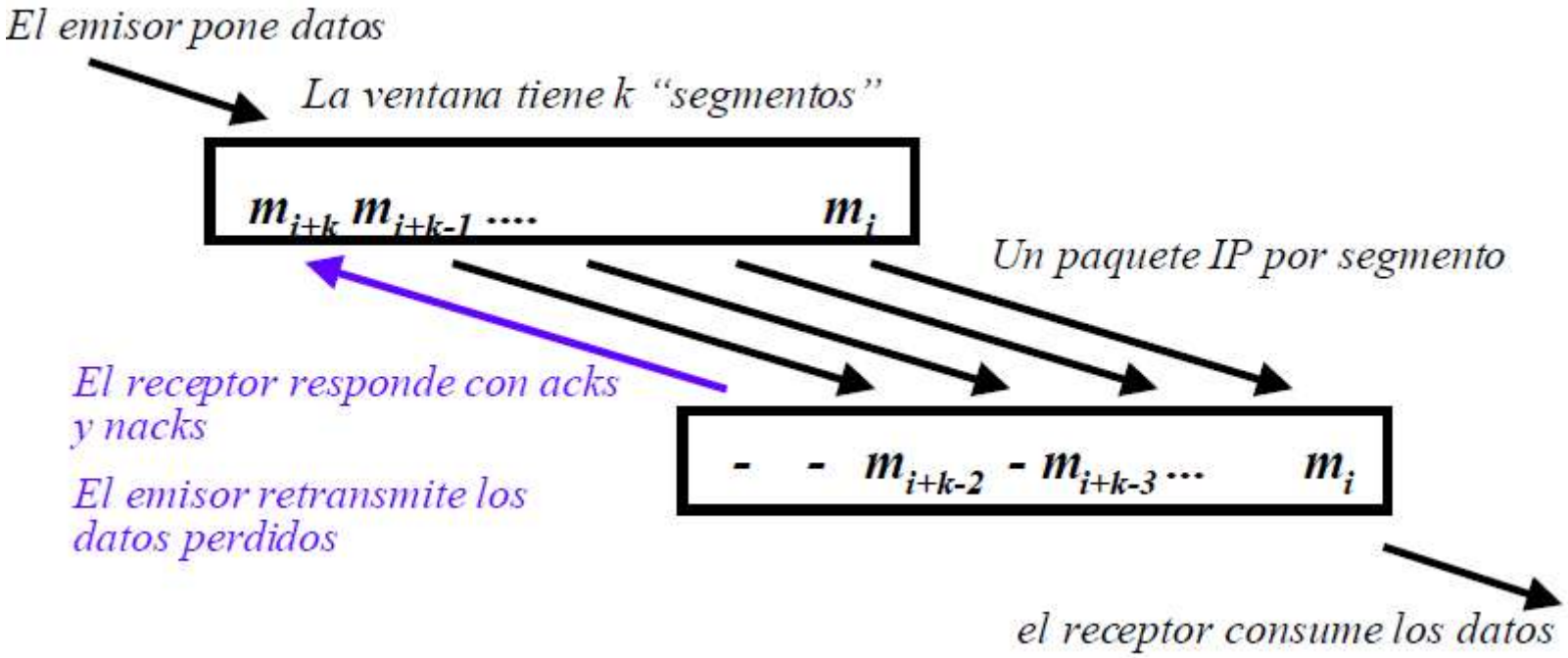
- TCP (Transmission Control Protocol)
  - Protocolo orientado a conexión, fiable (recuperación de errores), y con control de flujo
  - Establece un camino de bytes (byte stream)
- UDP (User Datagram Protocol)
  - Protocolo no orientado a conexión y no fiable
  - Si se recibe un paquete sin errores se pasa al proceso de usuario destinatario, si no, se descarta silenciosamente
  - Límite de tamaño de datagrama: 64 KB
  - UDP Multicast.

t

- En una red local, UDP es más eficiente y normalmente no hay errores.
- A través de Internet es más seguro utilizar TCP

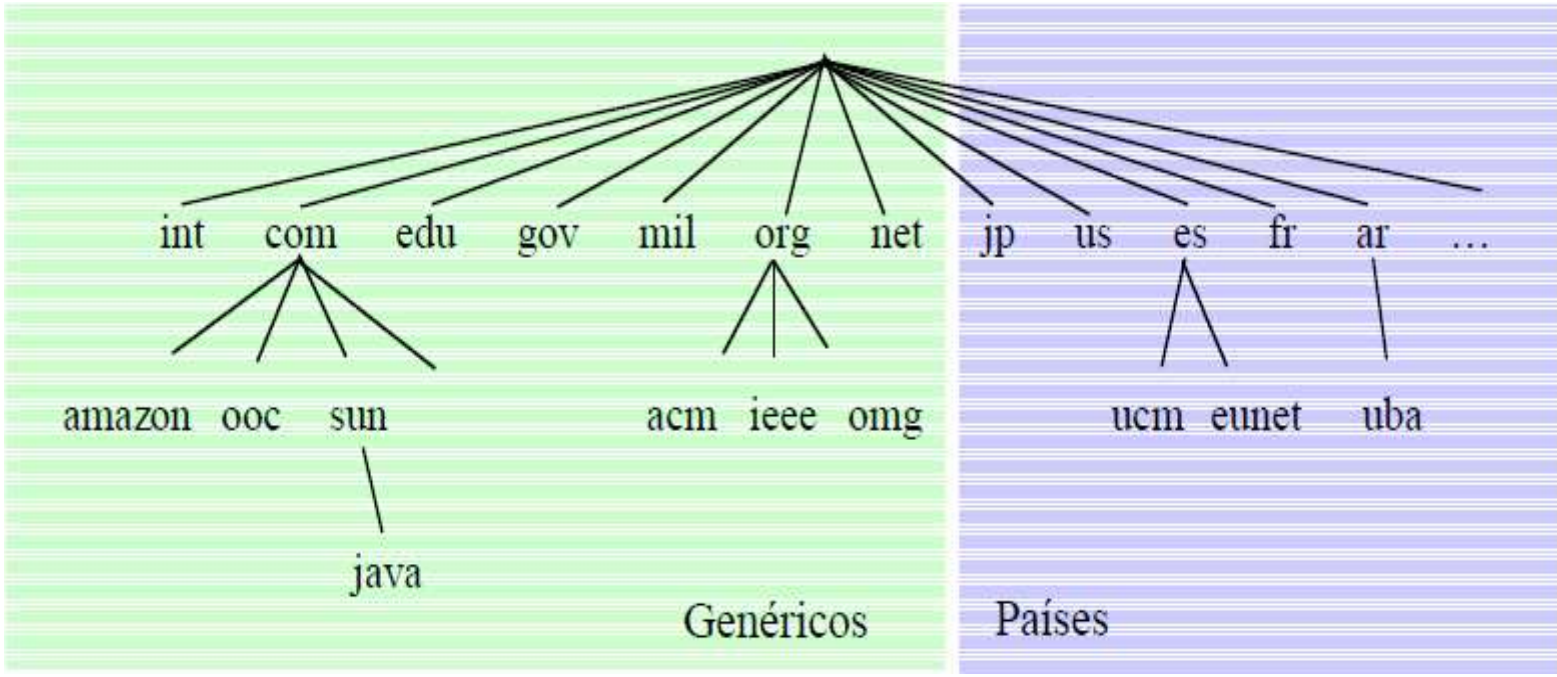
Protocolo de Ventana deslizante en TCP

- El tamaño de la ventana se reconfigura dinámicamente



DNS (Domain Name System)

Correspondencia entre máquinas y direcciones IP.

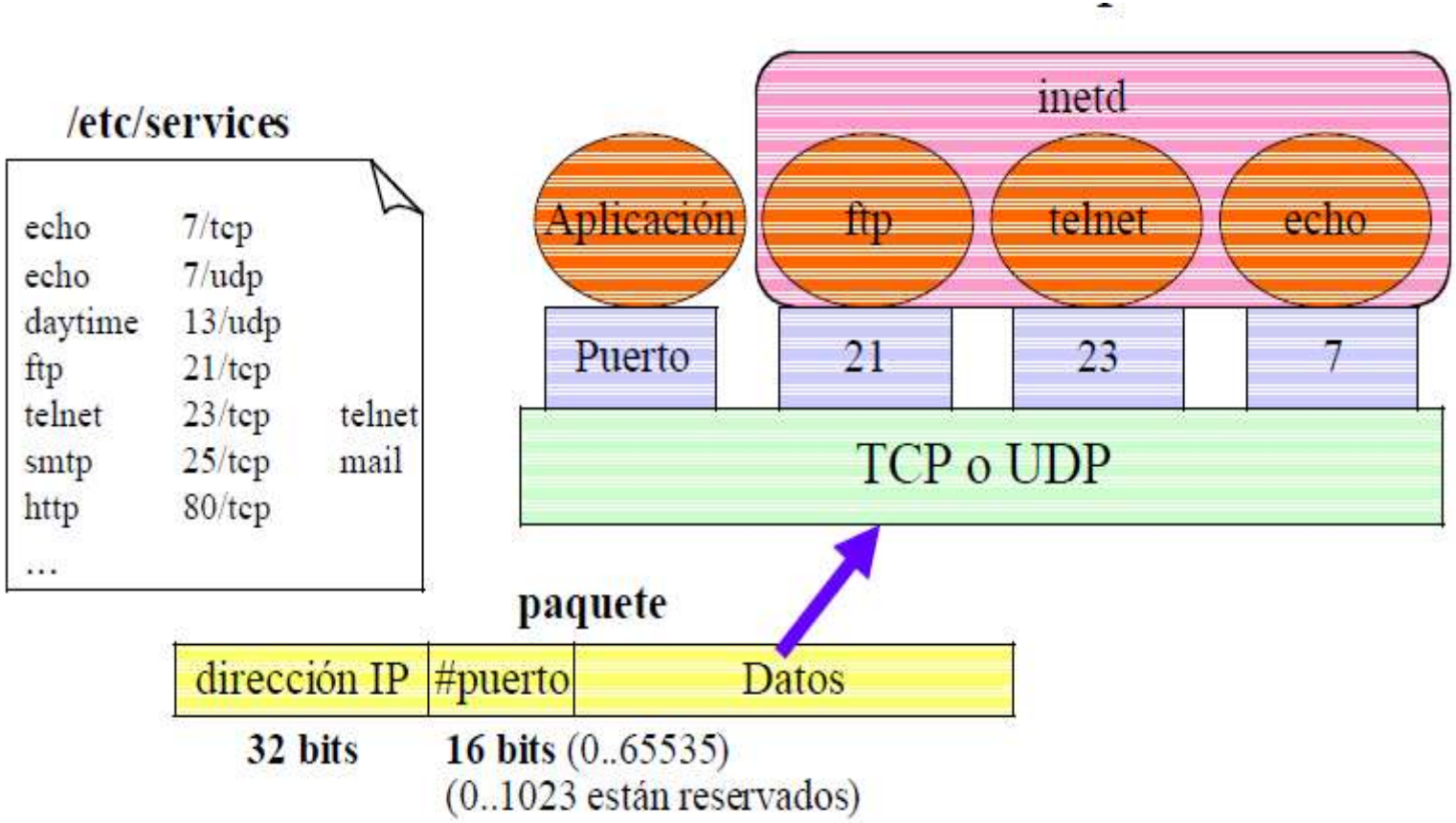


Protocolos de nivel de aplicación

- FTP: File Transport Protocol
- Telnet: Terminal Remoto
- Email: Simple Mail Transfer Protocol (SMTP)
- News: Network News Transfer Protocol (NNTP)
- DNS: Domain name service protocol
- NIS: Network information service (“Yellow Pages”)
- NFS: Network file system protocol
- X11: X-server display protocol
- Web: HyperText Transfer Protocol (HTTP)

Puertos

Cada servicio está asociado a un Puerto.



Puertos

- Los protocolos TCP y UDP usan puertos para asignar datos entrantes a un proceso en particular que se ejecuta en un ordenador.
- Un ordenador tiene una única conexión a la red y el ordenador lo envía a la aplicación adecuada mediante el uso de puertos.
- Los datos transmitidos a través de Internet van acompañados de información de direccionamiento que identifica la máquina y el puerto para el que está destinada.
- En una comunicación basada en TCP, una aplicación de servidor vincula un socket a un número de puerto específico.
- Algunos puertos están reservados para servicios específicos. Los número de **puerto inferiores a 1024** se identifican con **servicios históricos** y son puertos conocidos. Y los número de Puerto entre 49152 y 65535 están disponibles para uso general y se denominan **puertos efímeros**.

Clases java para comunicaciones en red

Java.net

- Clase **URL**: representa un puntero a un recurso en la web.
- Clase **URLConnection** : admite operaciones más complejas en las URLs.

- URL (Universal Resource Locator)

  http://maquina/directorio/fichero.html

  - nombre de recurso (p.ej., un fichero) Identificador de protocolo
- Clase URL
  - URL javasoft = new URL("http://www.javasoft.com/");
  - URL javadownload = new URL(javasoft, "download.html");
- Métodos para manipular el URL: getProtocol, getHost, getPort, getFile, getRef, openStream

Clases java para comunicaciones en red

Para dar soporte a los sockets TCP:

- Clase **ServerSocket** : utilizada por el programa servidor para crear un socket en el puerto en el que se escucha las peticiones de conexión de los clientes. Métodos:
  - **accept()** : recibe las peticiones de establecimiento de conexión, proporcionando un socket.
  - **close ( )** : Cierra el socket.
- Clase **Socket** : utilizada tanto por el cliente como por el programa servidor para comunicarse entre sí leyendo y escribiendo datos usando streams. Métodos:
  - **close ( )** : Cierra el socket.
  - **connect ():** Conecta este socket con el servidor.
  - **getInputStream ():** Proporciona un stream de lectura.
  - **getOutPutStream ():** Proporciona un stream de escritura.

Clases java para comnicaciones en red

- Clases **DatagramSocket , MulticastSocket y DatagramPacket**: para dar soporte a la comunicación vía datagramas UDP.

- Clase **InetAddress** : que representa las direcciones de Internet. Métodos:

  - **getLocalHost():** devuelve un objeto InetAddress que representa la dirección IP de la máquina donde se está ejecutando el programa.
  - **getByName (String host):** devuelve un objeto InetAddress que representa la dirección IP de la máquina que se especifica como parámetro (host). Este parámetro puede ser el nombre de la máquina, nombre de un dominio o una dirección IP.
  - **getAllByName (String host):** devuelve un array de objetos InetAddress. Este método es útil para averigurar todas las direcciones IP que tenga asignada una máquina.
  - **getHostAddress ():** devuelve la dir. IP de un objeto InetAddress en forma de cadena.
  - **getHostName ():** devuelve el nombre del host de un objeto InetAddress.
  - **getCanonicalHostName ():** dirección real del host de un objeto InetAddress.

Socket

- Abstracción programable de canal de comunicación

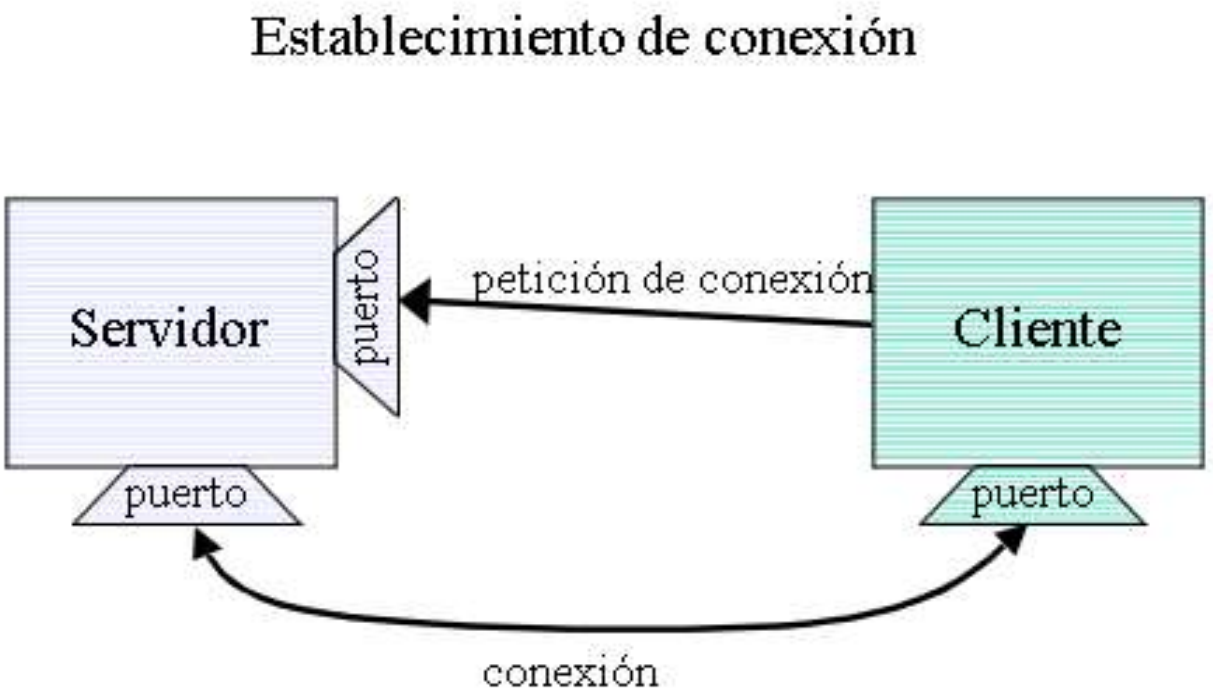
  dirección de socket = dirección IP + número de puerto.

- Dos procesos se pueden intercambiar información usando un par de sockets:
  - Los mensajes van entre un socket de un proceso y otro socket en otro proceso.
  - Cuando los mensajes son enviados, se encolan en el socket hasta que el protocolo de red los haya transmitido.
  - Cuando llegan, los mensajes son encolados en el socket de recepción hasta que el proceso receptor los recoja.
- Ciclo de vida igual al sistema de E/S Unix: creación, lectura y escritura y destrucción.

Tipos de Sockets

- **Socket Stream (TCP)** , servicio de transporte orientado a conexión:
  - En el servidor un socket atiende peticiones de conexión.
  - En el cliente un socket solicita una conexión.
  - Una vez conectados, se pueden usar para transmitir datos en ambas direcciones.
- **Socket Datagrama (UDP)** , servicio de transporte no orientado a conexión:
  - En cada datagrama es necesario enviar el descriptor del socket local y la dirección del socket que debe recibir el datagrama.

Socket streams



Programación de Sockets – lado Cliente

- Crea un socket:
  - Socket ladoCliente;
  - ladoCliente = new Socket ("maquina", numeroPuerto);
- Le asocia un flujo (stream) de datos para entrada (recepción) y otro para salida (emisión) :
  - DataInputStream entrada; PrintStream salida;
  - entrada = new DataInputStream
    (ladoCliente.getInputStream());
  - salida = new PrintStream
    (ladoCliente.getOutputStream());
- Lee y escribe de los flujos asociados
- Para finalizar, cierra los flujos y el socket :
  - salida.close( ); entrada.close( ); ladoCliente.close( );

Programación de Sockets – lado Servidor

- Crea un socket en el servidor:
  - ServerSocket servicio;
  - servicio = new ServerSocket (numeroPuerto);
- Espera la recepción de peticiones de conexión :
  - Socket socketServicio;
  - socketServicio = servicio.accept();
- Acepta la nueva conexión y crea flujos de entrada y salida de datos que envía al nuevo socket :
  - Lo normal es crear un hilo asociado para dar servicio a la nueva conexión.
- Lee y escribe de los flujos asociados.
- Para finalizar cierra los flujos y los sockets.

Programación de Sockets – UDP

- Tienen características distintas a los sockets UDP (clases, métodos) aunque el proceso de comunicación es similar.

- Diferencia fundamental:
  - Los paquetes enviados mediante UDP son independientes entre sí, por lo que las estructuras de datos que los soportan son arrays de bytes. Que deben ser dimensionados correctamente para evitar pérdida de información o desaprovechamiento del BW del canal.

Programación de Sockets UDP– Servidor

- **Proceso para envío y recepción de datagramas**:
  - Creación de un socket UDP mediante mediante la clase **DatagramSocket** asociado a un puerto.
  - Creación del array de bytes que actuará de buffer donde almacenar el mensaje recibido.
  - Creación del datagrama mediante la clase **DatagramPacket** utilizando el buffer creado anteriormente.
  - Recepción del datagrama mediante el método **receive()** del socket. En este punto el servidor se queda a la espera de envíos que vengan del cliente.
  - La generación y envío de la respuesta se realizará a partir de la información contenida en el datagrama recibido (host y puerto), y del uso del método **send()** del socket.
  - Envío del datagrama con el método **send()** del socket.

Programación de Sockets UDP– Cliente

- Obtención de la dirección del servidor mediante el uso del método **getByName()** de la clase **InetAddress**.
- Creación del socket UDP mediante la clase **DatagramSocket.**
- Generación del datagrama mediante la clase **DatagramPacket** utilizando el array de bytes con el contenido que se desea enviar.
- Envío del datagrama con el método **send()** del socket.
- Para la recepción de la respuesta se debe crear un array de bytes de tamaño suficiente para para almacenar la respuesta, y utilizar el método **receive()** del socket.
- Una vez finalizada la comunicación se cierra el socket con **close().**

Conexión con múltiples clientes

- Lo más normal es que la clase servidor pueda atender a muchos clientes simultáneamente.
- La solución para poder atender a múltiples clientes es es el **multihilo**, cada cliente será atendido por un hilo.
- Esto lo podemos implementar con sockets TCP o UDP multihilo.

Socket TCP multihilo

- El esquema básico sería el siguiente:

  - Construir un único servidor con la clase **ServerSocke**t e invocar el método **accept()** para esperar las peticiones de conexión de los clientes.
  - Cuando un cliente se conecta, el método **accept()** devuelve un objeto Socket que se usará para crear un hilo cuya misión es atender a ese cliente.
  - Después se vuelve a invocar a **accept()** para esperar a un nuevo cliente. La espera de conexiones se hace dentro de un bucle infinito.