Instituto Politécnico Nacional
Escuela Superior de Cómputo
Ingeniería en Sistemas Computacionales

Aplicaciones para Comunicaciones en Red

UNIDAD I: SOCKETS DE FLUJO

Septiembre 2015

M. en C. Sandra Ivette Bautista Rosales

Introducción

¿Qué es…?

Capa (nivel)

Protocolo

Iguales (*peers*)

Interfaz

Introducción

- Capa: Contiene los protocolos o procedimientos
  - El número de capas, su nombre, contenido y función difieren de red en red.
  - Ofrece servicios a las capas superiores, a las cuales no se les muestran los detalles reales de implementación de los servicios ofrecidos (abstracción, encapsulamiento, etc.)
  - La mayoría de las redes está organizada como una pila de capas, cada una construida a partir de la que está debajo de ella, con el propósito de reducir la complejidad de su diseño.
  - El modelo de capas describe el funcionamiento de los protocolos que se producen en cada capa y a su vez describe la interacción entre las diferentes capas.
    - Proporciona un lenguaje común para la comunicación en las redes informáticas.
    - Evita que los continuos cambios tecnológicos afecten a los protocolos y a las distintas capas.

- Protocolo: Acuerdo entre las partes en comunicación sobre cómo se debe de llevar a cabo la misma.
  - La capa *n* de una máquina mantiene una conversación con la capa *n* de otra máquina. Las reglas y convenciones utilizadas en esta conversación se conocen de manera colectiva como protocolo de capa *n*.
  - Violar el protocolo hará más difícil la comunicación, si no es que imposible.

Introducción

- Iguales: Las entidades que abarcan las capas correspondientes en diferentes máquinas.
  - Podrían ser procesos, dispositivos de hardware o incluso seres humanos, es decir, los iguales son los que se comunican a través del protocolo.

- Interfaz: Define qué operaciones y servicios primitivos pone la capa más baja a disposición de la capa superior inmediata.
  - Es una conexión entre dos máquinas o entidades de cualquier tipo, a las cuales les brinda un soporte para la comunicación a diferentes estratos.

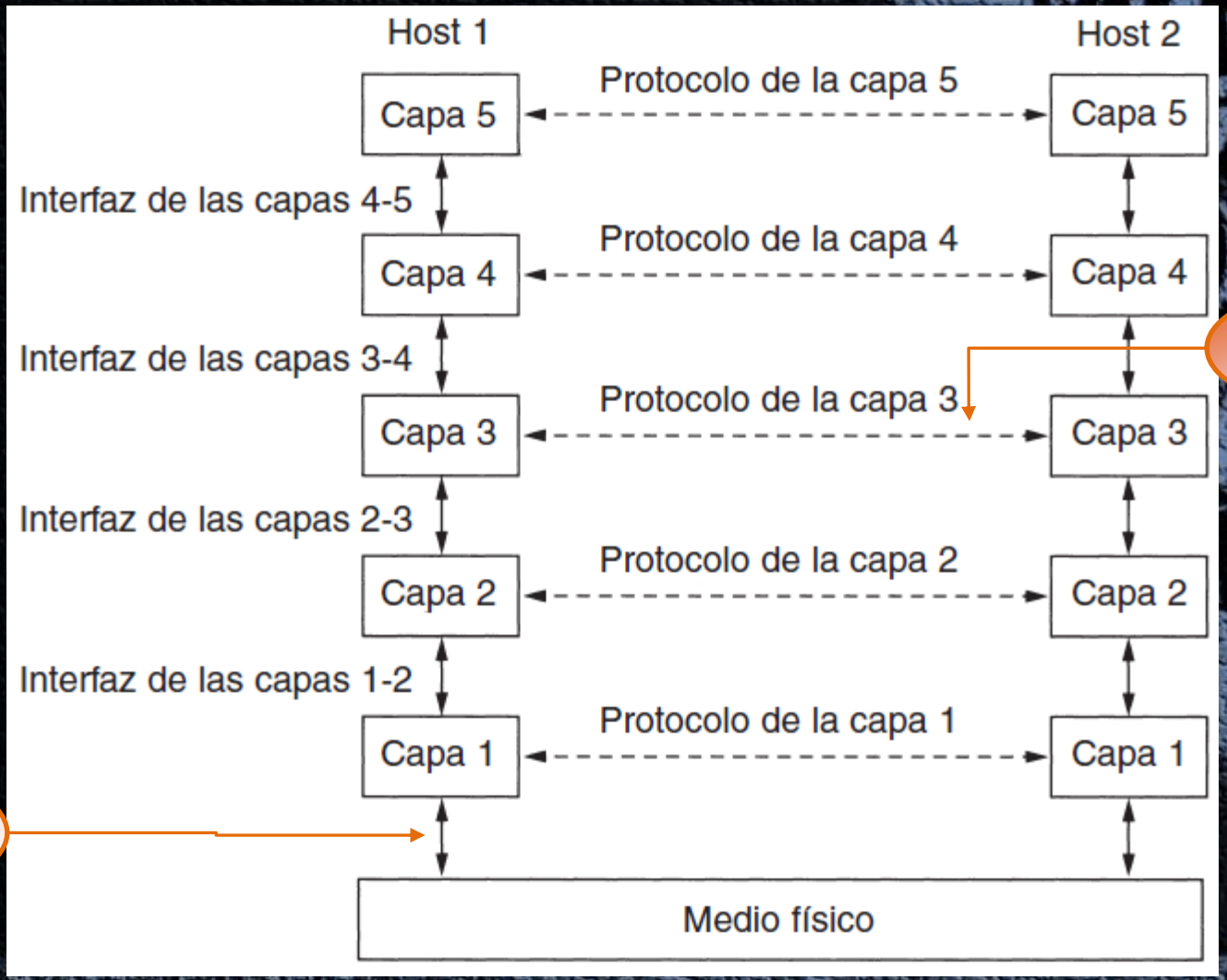


Figura 1. Capas, protocolos e interfaces [1]

Introducción

- Arquitectura de red: Conjunto de capas y protocolos
  - Su especificación debe contener información suficiente para permitir que un implementador escriba el programa o construya el hardware para cada capa de modo que se cumpla correctamente con el protocolo apropiado.
  - Ni los detalles de implementación, ni las especificaciones de las interfaces son parte de la arquitectura porque están ocultas en las máquinas.
  - Las interfaces no necesitan ser las mismas, siempre y cuando utilice correctamente los protocolos.
- Pila de protocolos: Lista de protocolos utilizados por un sistema (un protocolo por capa)

Introducción

- Identificar emisores y receptores

Introducción

- Identificar emisores y receptores
- Método para que un proceso en una máquina especifique con cuál de las demás quiere hablar.

Introducción

- Identificar emisores y receptores
- Método para que un proceso en una máquina especifique con cuál de las demás quiere hablar.
- Múltiples destinos ⇨ Destino específico

Introducción

- Identificar emisores y receptores
- Método para que un proceso en una máquina especifique con cuál de las demás quiere hablar.
- Múltiples destinos ⇨ Destino específico

Direccionamiento

Introducción

- Reglas de transferencia de datos
  - Dependiendo del sistema viajan los datos
  - Protocolo determina a cuántos canales lógicos corresponde la conexión y cuáles son sus prioridades.
  - Muchas redes proporcional al menos dos canales lógicos por conexión: uno para datos normales y otro para urgentes.

Introducción

- Los circuitos de comunicación física no son perfectos.

Introducción

- Los circuitos de comunicación física no son perfectos.

Detección de errores

Introducción

- Los circuitos de comunicación física no son perfectos.

Detección de errores

- Códigos de detección y corrección de errores
  - Ambos extremos de la conexión deben estar de acuerdo en cuál utilizar
  - Receptor: decirle al emisor cuáles mensajes se han recibido correctamente y cuáles no

Introducción

- No todos los canales de comunicación conservan el orden de envío de mensajes.
- Pérdida de secuencia
  - El protocolo debe incluir mecanismo que permita al receptor volver a unir los pedazos adecuadamente.
    - Numerar las piezas
    - ¿Qué se debe hacer con las piezas que llegan sin orden?

Introducción

- ¿Cómo evitar que un emisor rápido sature de datos a un receptor más lento?

Introducción

- ¿Cómo evitar que un emisor rápido sature de datos a un receptor más lento?

Control de flujo

Introducción

- ¿Cómo evitar que un emisor rápido sature de datos a un receptor más lento?

Control de flujo

- Limitar al emisor a una velocidad de transmisión acordada

Introducción

- Incapacidad de todos los procesos de aceptar de manera arbitraria mensajes largos.
- Mecanismos para:
  - Desensamblar
  - Transmitir y
  - Reensamblar mensajes

Introducción

- ¿Qué hacer cuando los procesos insisten en transmitir datos en unidades tan pequeñas que enviarlas por separado es ineficaz?

Introducción

- ¿Qué hacer cuando los procesos insisten en transmitir datos en unidades tan pequeñas que enviarlas por separado es ineficaz?
  - Reunir en un solo mensaje grande varios mensajes pequeños que vayan dirigidos a un destino común y desmembrar dicho mensaje una vez llegado a su destino.

Introducción

- Cuando es inconveniente o costoso establecer una conexión separada para cada par de procesos de comunicación, la capa subyacente podría decidir utilizar la misma conexión para múltiples conversaciones sin relación entre sí.

Introducción

- Cuando es inconveniente o costoso establecer una conexión separada para cada par de procesos de comunicación, la capa subyacente podría decidir utilizar la misma conexión para múltiples conversaciones sin relación entre sí.

multiplexión y desmultiplexión

Introducción

- Cuando es inconveniente o costoso establecer una conexión separada para cada par de procesos de comunicación, la capa subyacente podría decidir utilizar la misma conexión para múltiples conversaciones sin relación entre sí.

multiplexión y desmultiplexión

- Siempre y cuando esta multiplexión y desmultiplexión se realice de manera transparente , cualquier capa la podrá utilizar.

Introducción

- La multiplexión se necesita en la capa física, ej. Donde múltiples conversaciones comparten un número limitado de circuitos físicos.

Introducción

- La multiplexión se necesita en la capa física, ej. Donde múltiples conversaciones comparten un número limitado de circuitos físicos.
- Cuando hay múltiples rutas entre el origen y el destino, se debe elegir la mejor o las mejores entre todas ellas.

Introducción

- A veces esta decisión se debe dividir en dos o más capas
  - Alto nivel: ej. Leyes de privacidad
  - Bajo nivel: ej. Carga de tráfico

Introducción

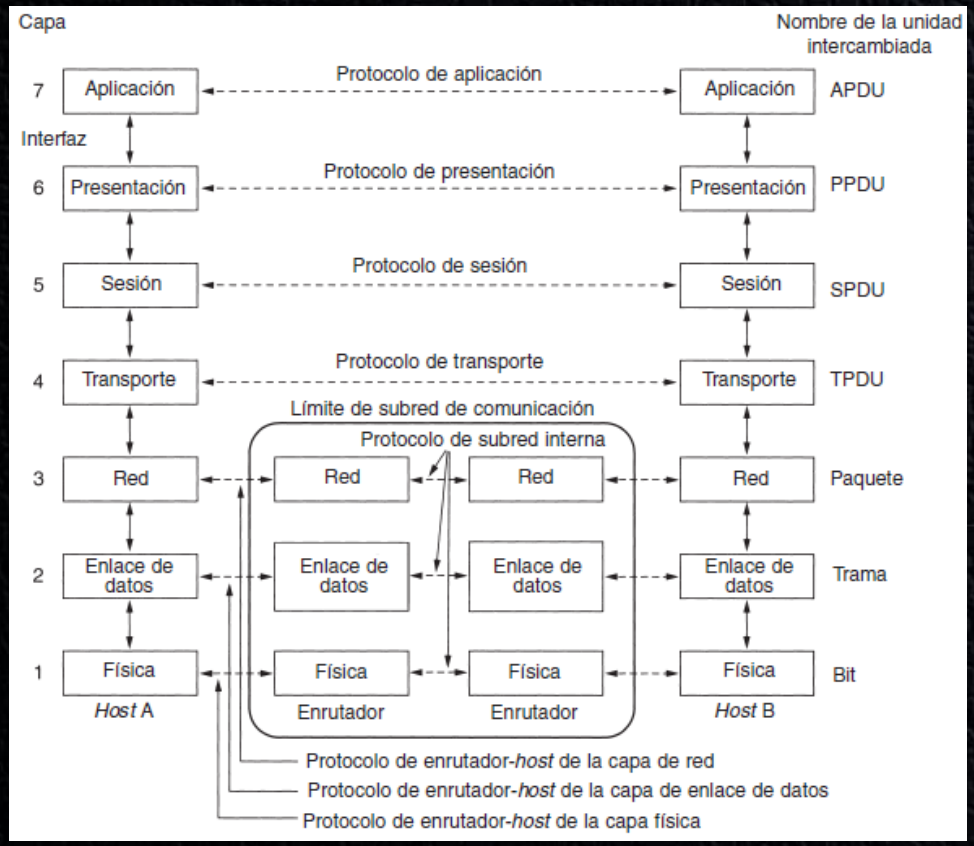
- A veces esta decisión se debe dividir en dos o más capas
  - Alto nivel: ej. Leyes de privacidad
  - Bajo nivel: ej. Carga de tráfico

Enrutamiento

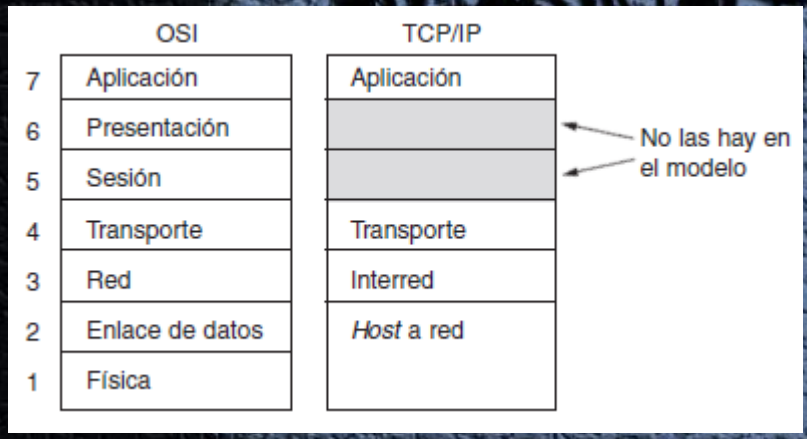
Introducción

- Servicio vs Protocolo [1]

1.1 Servicios definidos en la Capa de Transporte



Modelo de referencia OSI [1]



Modelo de referencia TCP/IP [1]

1.1 Servicios definidos en la Capa de Transporte

- El deber primario de la capa de Transporte es proporcionar la comunicación de un programa de aplicación a otro, que es llamada de extremo a extremo (end-to-end).
- Puede regular el flujo de la información y proporcionar fiabilidad en el transporte, garantizando que los datos llegan sin error y en secuencia.

Servicios orientados a la conexión y no orientados a la conexión

Servicio orientado a la conexión

- Para usar un servicio de red orientado a la conexión, el usuario:
  - Establece una conexión
  - La utiliza
  - La abandona

Servicios orientados a la conexión

- Conexión: el emisor empuja objetos (bits) en un extremo y el receptor los toma en el otro extremo.
  - Generalmente se conserva el orden de los bits
- Al establecer la conexión, el emisor, el receptor y la subred utilizan una negociación sobre los parámetros que se van a utilizar, por ejemplo:
  - Tamaño máximo del mensaje
  - Calidad del servicio solicitado
- Uno hace la propuesta y el otro la acepta, rechaza o hace una contrapropuesta.

Servicios No orientados a la conexión

- Cada mensaje (carta) lleva la dirección destino y cada una se enruta a través del sistema, independientemente de las demás.

Servicios No orientados a la conexión

- Generalmente, cuando se envían dos mensajes al mismo destino el primero que se envíe será el primero en llegar.
  - Es posible que el primero se dilate y llegue primero el segundo mensaje.

Servicios orientados a la conexión y no orientados a la conexión

- Cada servicio se puede clasificar por la calidad del servicio
  - Servicio confiable: el receptor confirma la recepción de cada mensaje para que el emisor esté seguro de que llegó
    - Introduce sobre cargas y retardos, que con frecuencia son valiosos pero a veces son indeseables.

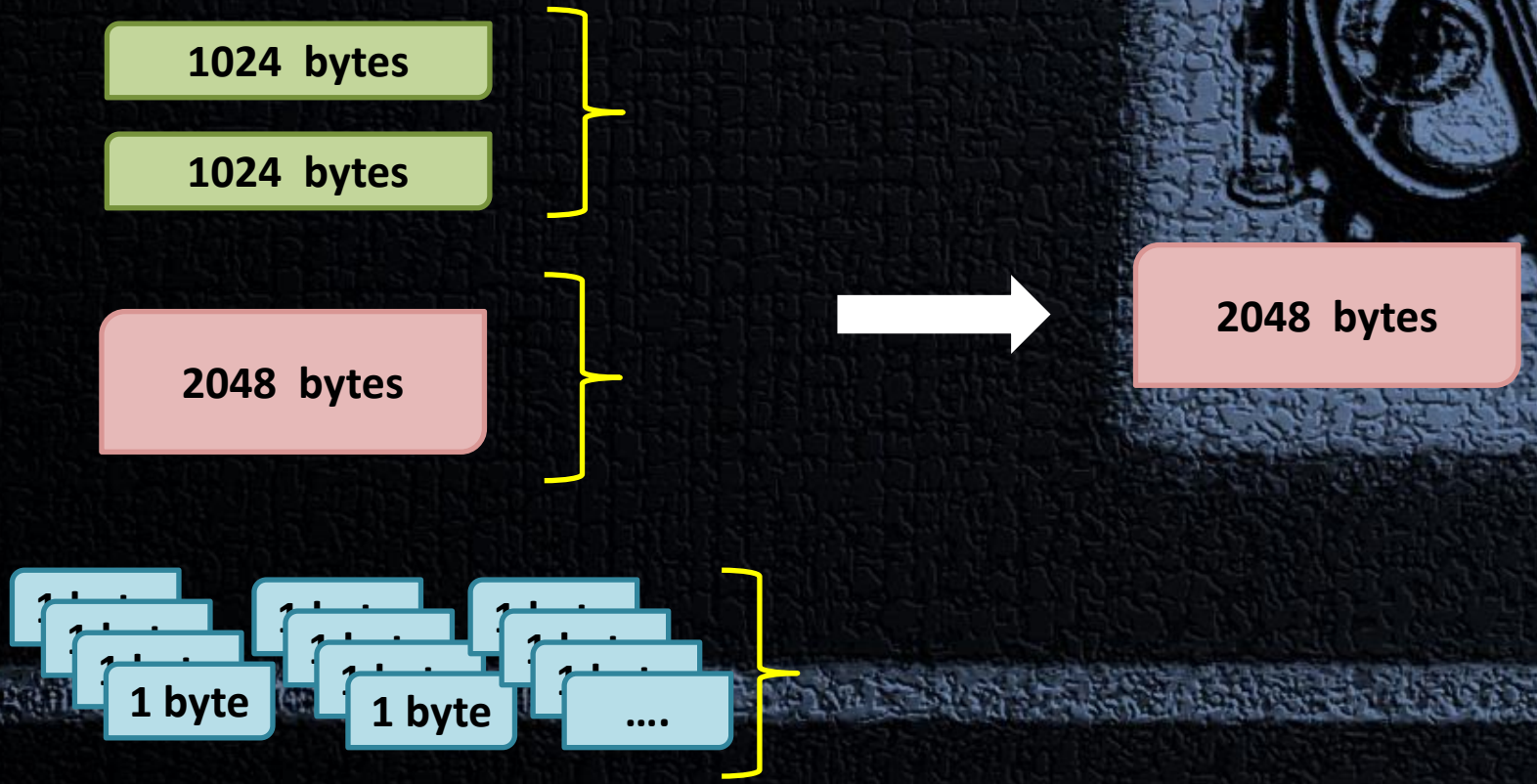
Ejemplo: Servicio orientado a la conexión

Servicios orientados a la conexión

- Un servicio orientado a la conexión confiable tiene dos variantes menores :
  1. Secuencias de mensaje: se conservan los límites del mensaje

Servicios orientados a la conexión

2. Flujo de bytes: la conexión es simplemente un flujo de bytes sin límites en el mensaje.



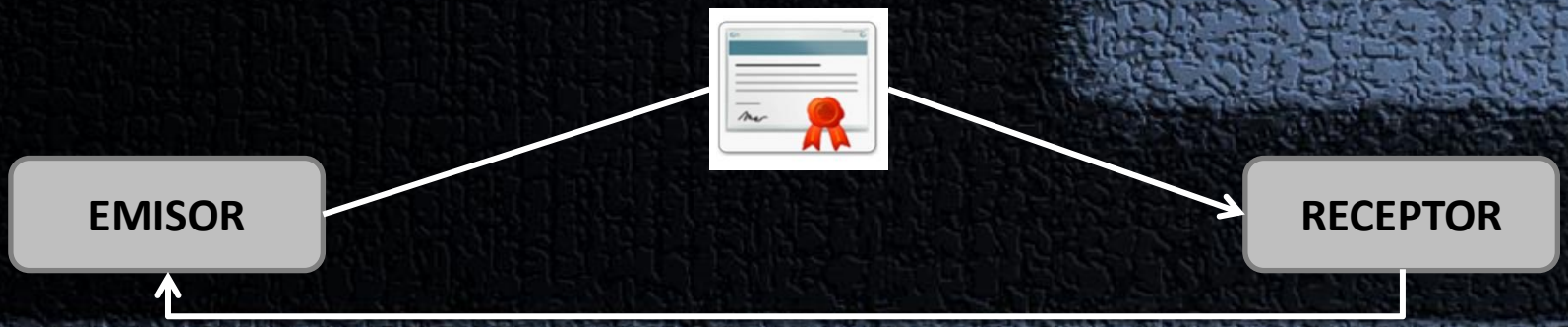
Servicios No orientados a la conexión

- No todas las aplicaciones requieren conexiones.
- Ejemplo:

- Al servicio no orientado a la conexión no confiable (sin confirmación de recepción) se le conoce como **servicio de datagramas.**

Servicios No orientados a la conexión

- En otras situaciones se desea la conveniencia de no tener que establecer una conexión para enviar un mensaje corto, pero la fiabilidad es esencial
  - **Servicio de datagramas confirmados**



Servicios No orientados a la conexión

- Servicio de solicitud-respuesta:
  - El emisor transmite un solo datagrama que contiene una solicitud, a continuación el servidor envía la respuesta

Servicios No orientados a la conexión

	Servicio	Ejemplo
Orientado a la conexión	Flujo confiable de mensajes	Secuencia de páginas
	Flujo confiable de bytes	Inicio de sesión remoto
	Conexión no confiable	Voz digitalizada
No orientado a la conexión	Datagrama no confiable	Correo electrónico basura
	Datagrama confirmado	Correo certificado
	Solicitud-respuesta	Consulta de base de datos

Seis diferentes tipos de servicio [1]

1.2 Modelo Cliente-Servidor

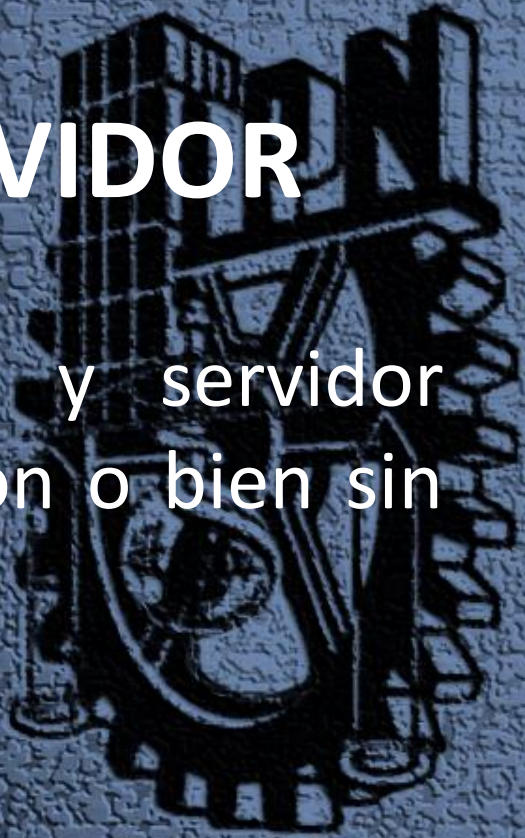
- Es el modelo de ejecución que siguen todas las aplicaciones de red.

- Un SERVIDOR es un proceso que se está ejecutando en un nodo de la red
  - Su función es gestionar el acceso a un determinado recurso.

- Un CLIENTE es un proceso que se ejecuta en el mismo o diferente nodo y realiza peticiones al servidor.
  - Las peticiones están originadas por la necesidad de acceder al recurso que gestiona el servidor.

MODELO CLIENTE/SERVIDOR

- La comunicación entre cliente y servidor puede ser orientada a la conexión o bien sin conexión.



SERVIDOR

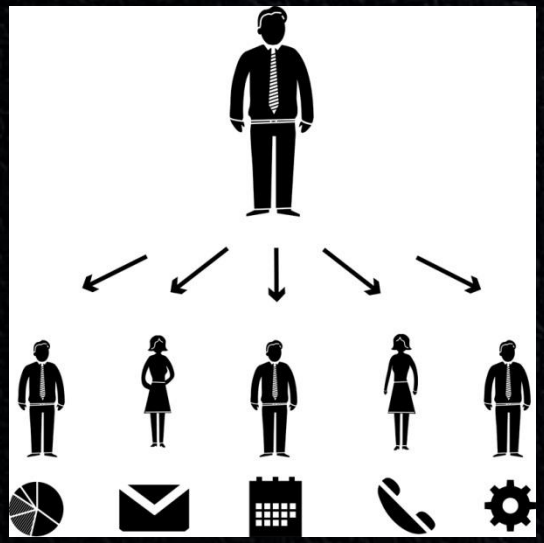
- Está continuamente esperando peticiones de servicio.
- Cuando se produce una petición, el servidor despierta y atiende al cliente.
- Cuando el servicio concluye, el servidor vuelve al estado de espera.

SERVIDOR

- De acuerdo con la forma de prestar servicio , se pueden considerar 2 tipos de servidores:

- SERVIDORES INTERACTIVOS: El servidor no solo recoge la petición de servicio, sino que él mismo se encarga de atenderla.
  - Inconveniente: Si el servidor es lento en atender a los clientes y hay una demanda de servicio muy elevada, se van a originar tiempos de espera muy grandes.

SERVIDOR



- SERVIDORES CONCURRENTES:
  - El servidor recoge cada un de las peticiones de servicio y crea otros procesos para que se encarguen de atenderlas.
  - Este tipo de servidores solo es aplicable en sistemas multiproceso.
  - Ventaja: El servidor puede recoger peticiones a muy alta velocidad, porque esta descargado de la tarea de atención al cliente.
  - En aplicaciones donde los tiempos de servicio son variables, es recomendable implementar este tipo de servidores.

SERVIDOR

- Su papel es pasivo en el establecimiento de la comunicación.
  - Para esto dispone de un socket de escucha, enlazado al puerto TCP correspondiente al servicio, sobre el que espera las peticiones de conexión.
  - Cuando llega al sistema una petición de este tipo, se despierta el proceso servidor y se crea un **nuevo socket**, que se llama *socket de servicio*, el cual se conecta al cliente.

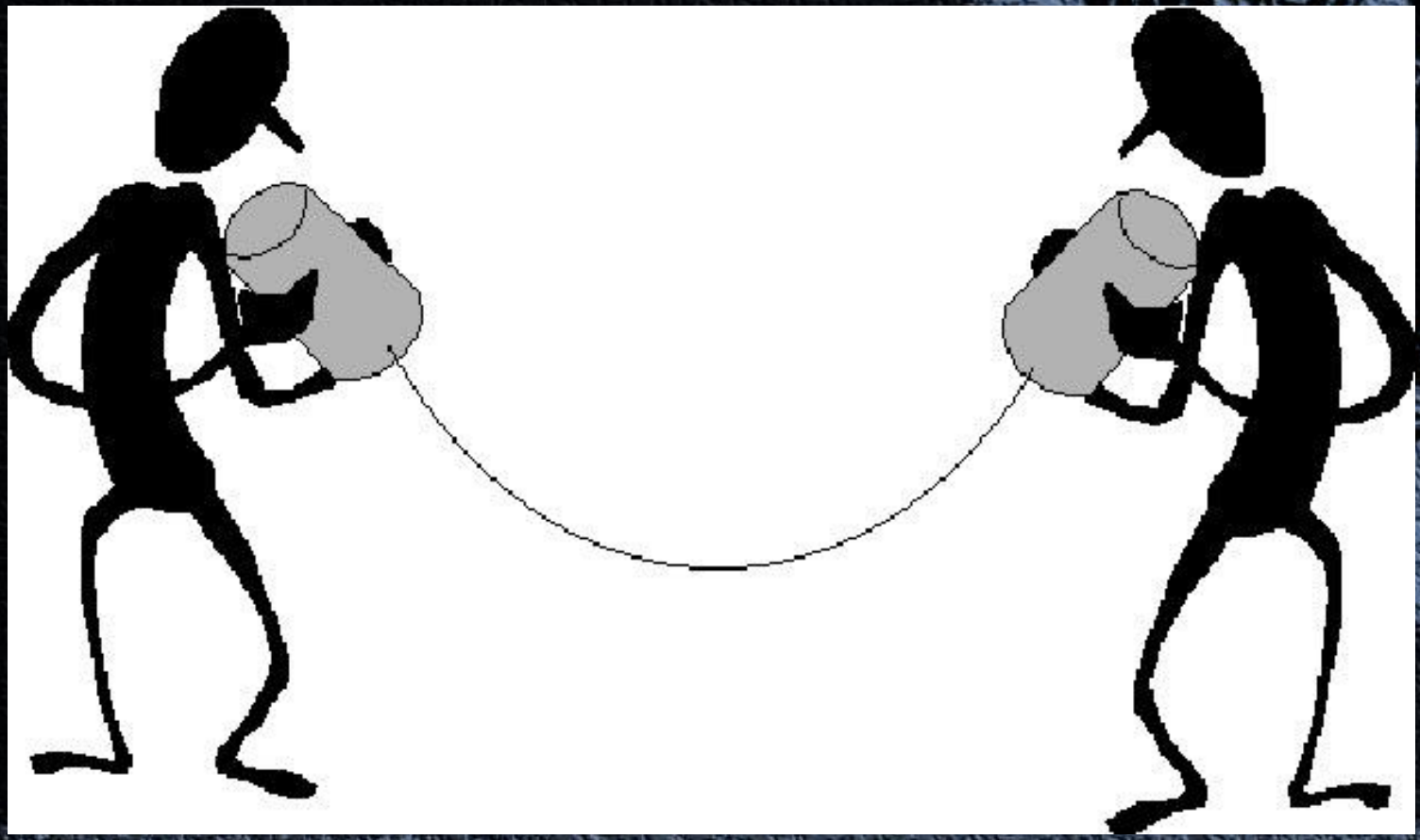
SERVIDOR

- Entonces el servidor podrá:
  - Delegar el trabajo necesario para la realización del servicio a un nuevo proceso (creado por fork), que utilizará entonces la conexión.
  - Volverá al socket de escucha.

CLIENTE

- Es la entidad activa en el establecimiento de una conexión, puesto que es el que toma la iniciativa de la demanda de conexión a un servidor.
  - Esta demanda se realiza por medio de la primitiva **connect()** solicitando el establecimiento de una conexión que será conocida por los dos extremos.
  - Además el cliente está informado del éxito o el fracaso del establecimiento de la conexión.

Comunicación entre procesos



Comunicación entre procesos

- Sistemas en red requieren dar soporte a la comunicación entre procesos
  - Permite que 2 procesos colaboren en una tarea
- Comunicación entre procesos en una misma máquina
  - Pipes, memoria compartida, señales, semáforos, etc.
- Comunicación entre procesos en máquinas distintas

Comunicación entre procesos

- Intercambio de mensajes (emisor-receptor(es))
  - Uno a uno (unicast)
  - Uno a muchos (multicast)
- Esquema típico: mecanismo petición-respuesta
  - Distintos niveles de abstracción
  - Ejemplos: interfaz sockets, mecanismos RPC (llamada a procedimiento remoto)

Sincronización en mecanismos de paso de mensajes

¿Qué es…?

Primitiva de servicio:

Un servicio está definido por un conjunto de operaciones más sencillas llamadas PRIMITIVAS.

En general, las primitivas se utilizan para realizar alguna acción o para informar de un suceso ocurrido en una entidad par.

Sincronización en mecanismos de paso de mensajes

- Conceptualmente todo mecanismo de paso de mensajes contará con las siguientes primitivas básicas:
  - Enviar: proceso emisor transmite datos a un proceso receptor.
  - Recibir: proceso receptor acepta los datos de un emisor.
  - Iniciar conexión: (opcional, en sistemas orientados a conexión) un proceso indica que desea iniciar una conexión con otro
    - Proceso activo, típica mente un cliente

Sincronización en mecanismos de paso de mensajes

- Esperar conexión: (opcional, en sistemas orientados a conexión) un proceso indica que está dispuesto a recibir conexiones
  - Proceso pasivo, típicamente un servidor
- Aceptar conexión: (opcional, en sistemas orientados a conexión) un proceso acepta la comunicación con otro
  - Proceso pasivo, típicamente un servidor

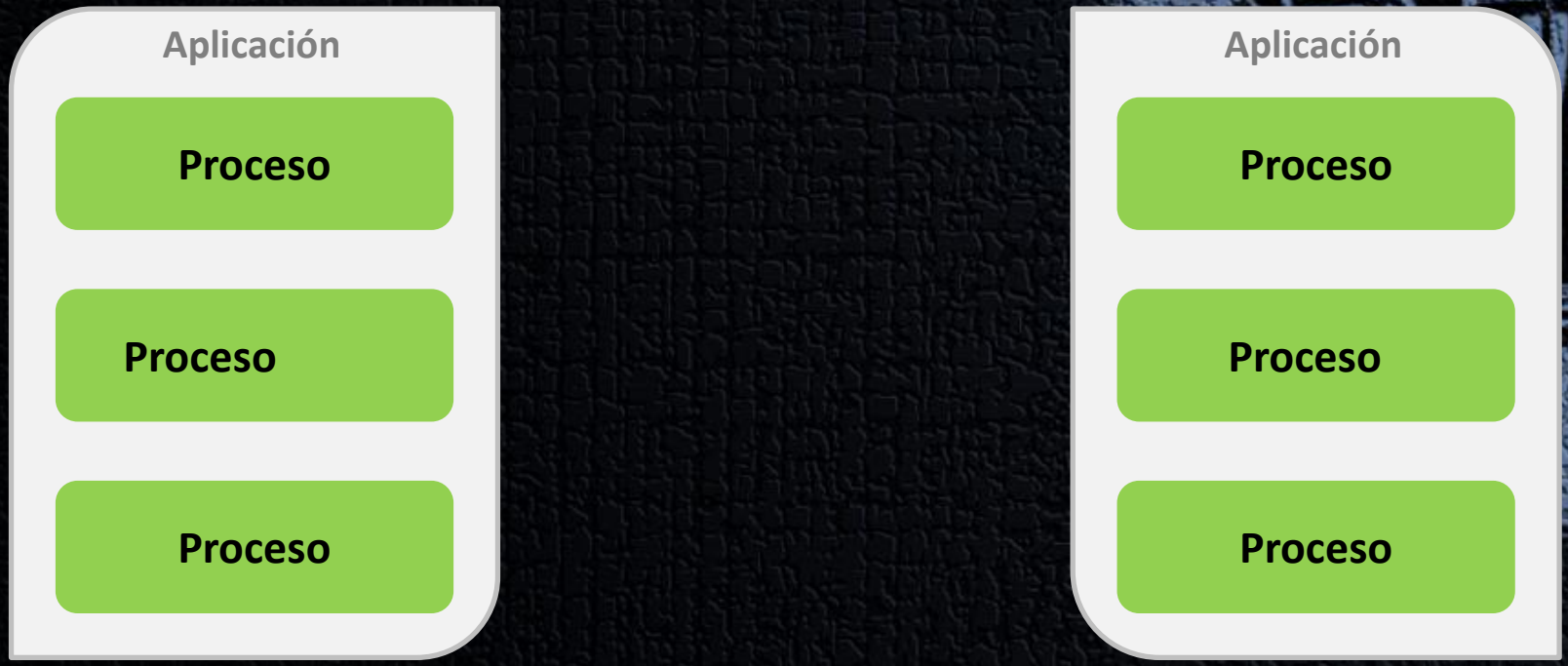
Sincronización

- Para asegurar el establecimiento de la conexión:

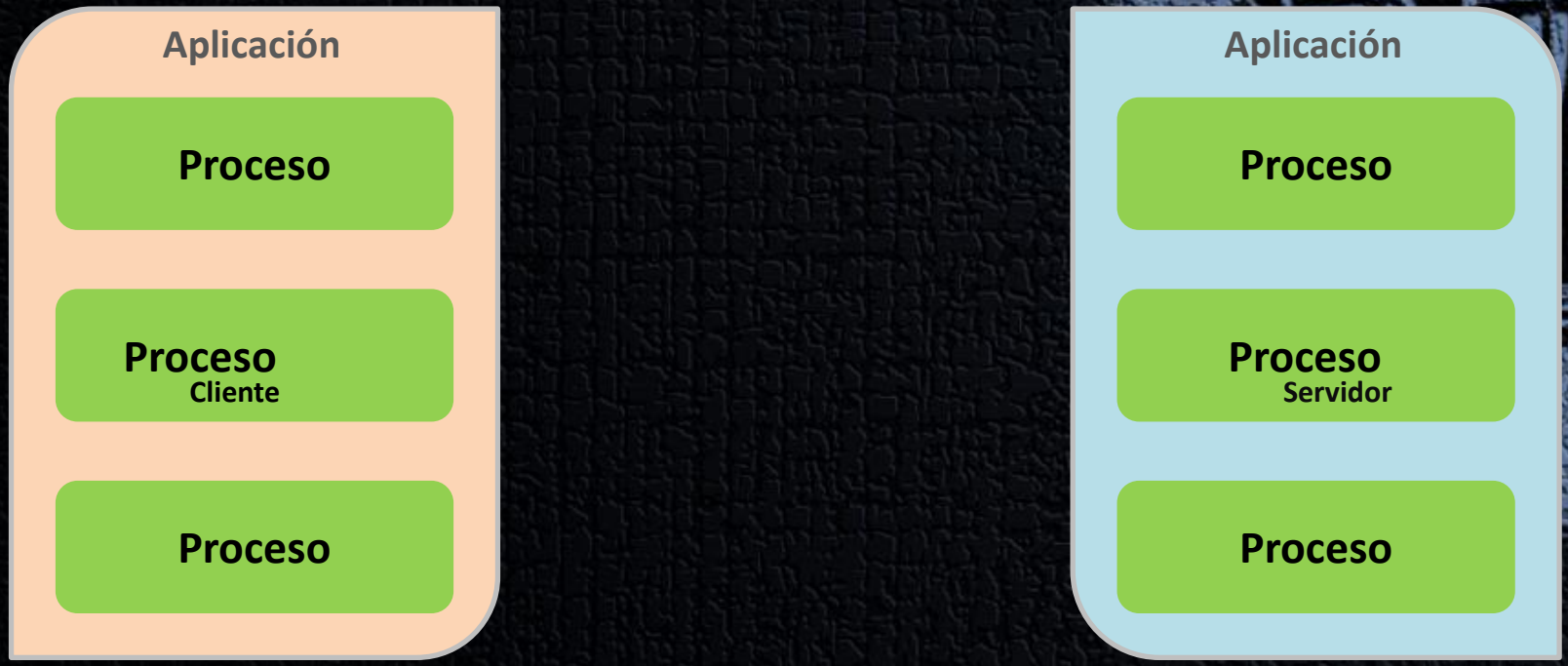
- Para asegurar la transferencia de un mensaje:

- Esquemas de petición+respuesta (2 mensajes):

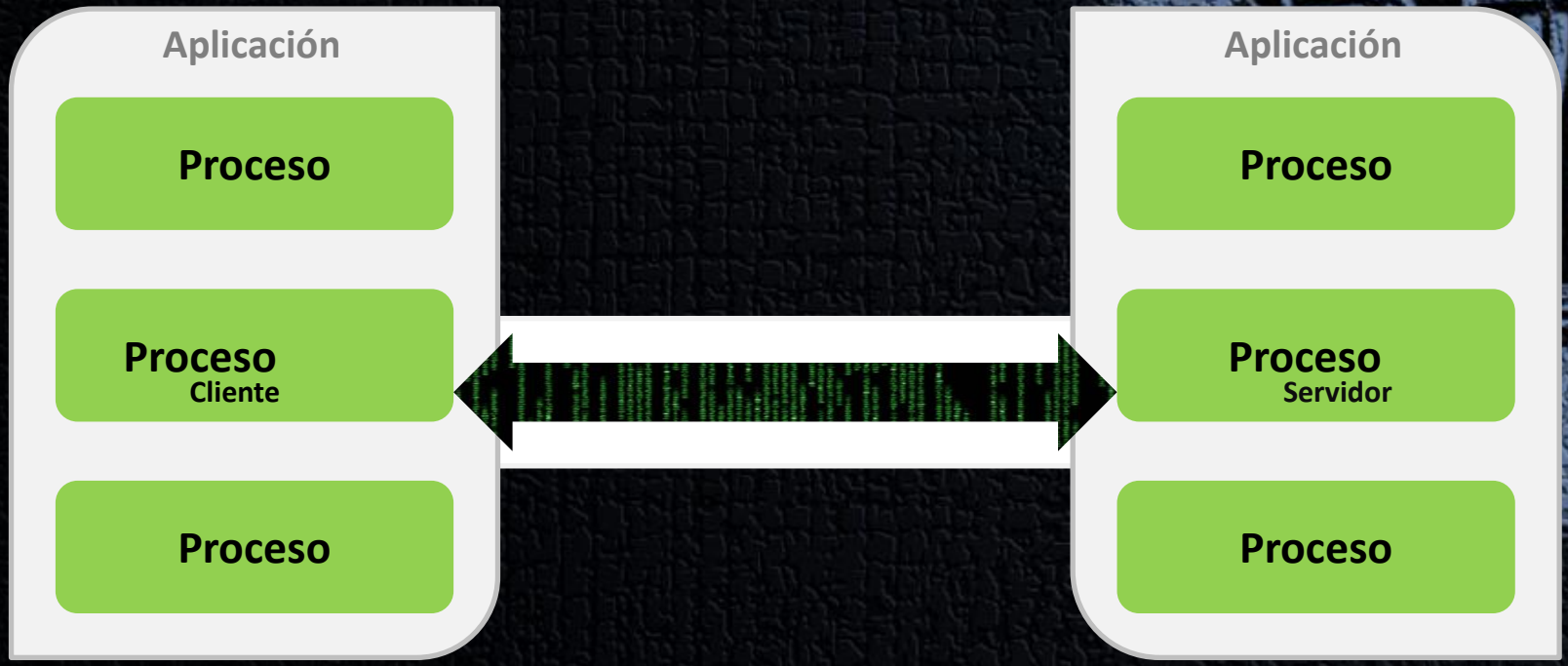
¿Qué es un socket?



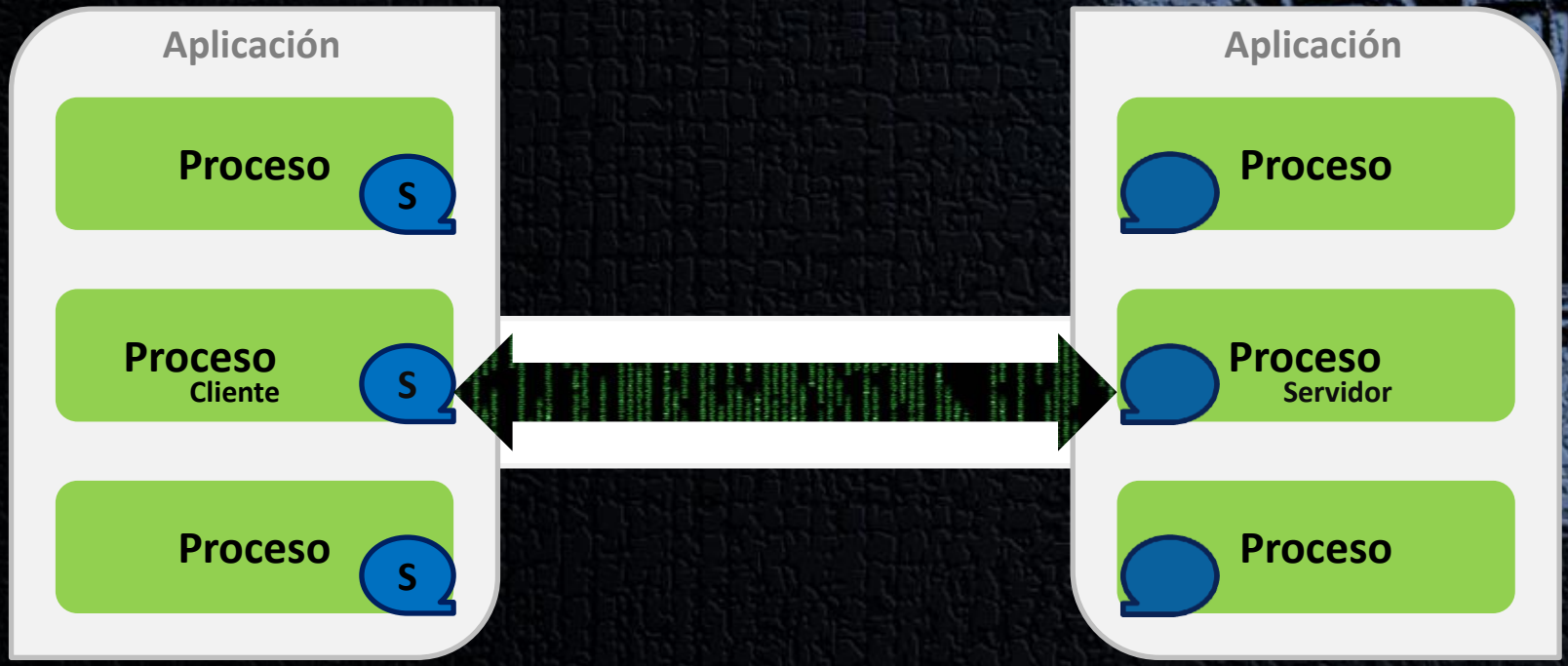
¿Qué es un socket?



¿Qué es un socket?



¿Qué es un socket?



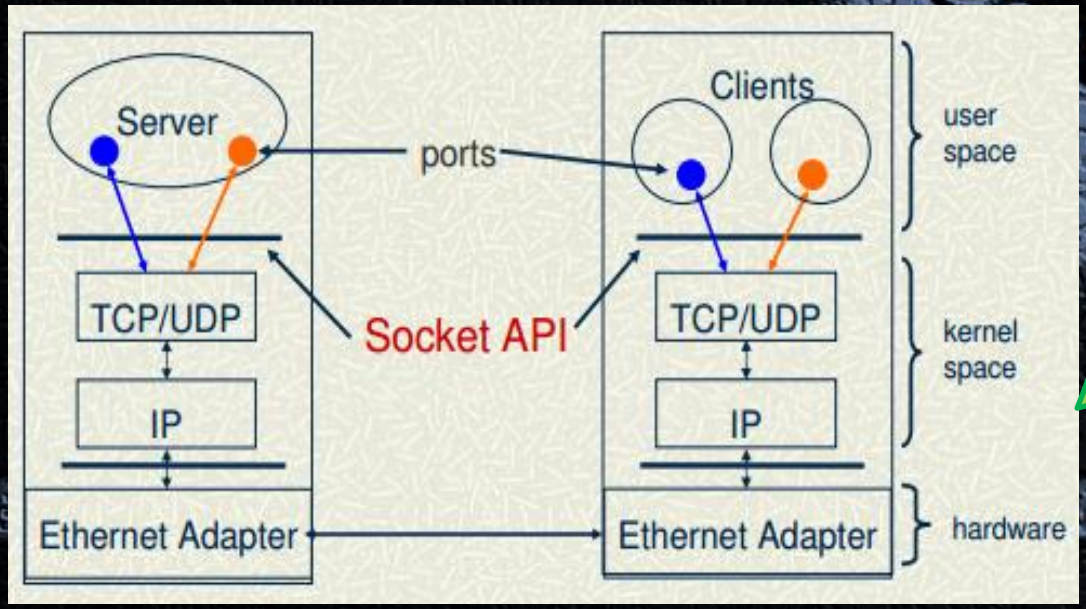
¿Qué es un socket?

- Aparecieron a principios de los 80's con el sistema UNIX de Berkeley
  - Con el fin de proporcionar un medio de comunicación a los procesos.
- Es un punto de comunicación por el cual un proceso puede emitir o recibir información
- En el interior de un proceso, un socket se identifica por un **descriptor** de la misma naturaleza que los que identifican a los archivos
  - SD: Es un número entero asociado a un fichero (archivo) abierto (conexión de red, una terminal, etc.)

Interfaz de Sockets

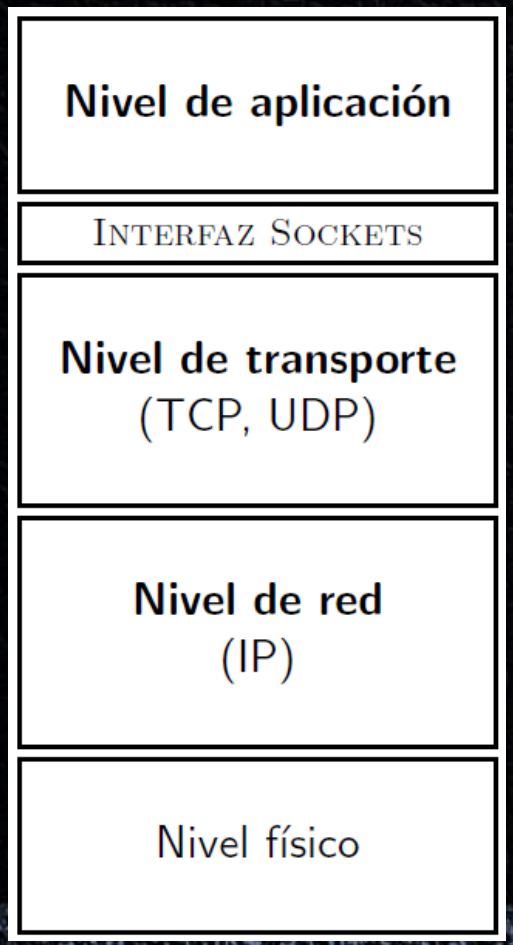
- Socket: Interfaz de programación (API) sobre el nivel de transporte
  - Abstracción que facilita al programador el acceso a los servicios y recursos del nivel de transporte
  - Ofrece un servicio punto a punto entre emisor y receptor

Los procesos de las aplicaciones residen en el espacio de usuario



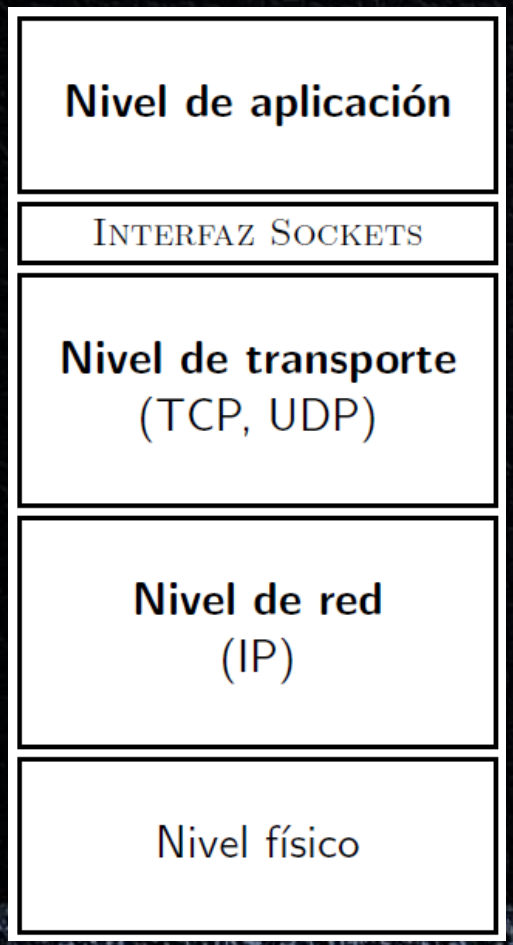
Los procesos de los protocolos de transporte forman parte del S.O.

Pila de protocolos TCP/IP



- Nivel de transporte:
  - Servicio de envío de datos extremo a extremo
  - Hace uso de servicios del nivel de red (IP)
- Protocolo TCP:
  - Servicio orientado a conexión, fiable, ordenado, con control de flujo y errores
    - Requiere establecimiento previo de una conexión entre ambos extremos
    - Ofrece flujo permanente entre los extremos en ambas direcciones
    - Controla la recepción en orden, completa y sin errores, gestionando el reenvío de paquetes perdidos

Pila de protocolos TCP/IP



- Protocolo UDP:
  - servicio no orientado a conexión, no fiable y sin control de flujo y errores
  - Cada mensaje UDP (datagrama) es independiente y se trata de forma aislada
  - No se garantiza la entrega de datagramas enviados ni que estos lleguen en orden

Primitivas para la utilización de sockets

- Secuencia de primitivas para la utilización de sockets que el cliente y el servidor tienen que usar para ambos tipos de servicio:
  - Orientado a conexión
  - No orientado a conexión

Dominio de un socket

- Representa una familia de protocolos
- Una familia o dominio de la conexión, agrupa todos aquellos sockets que comparten características comunes y con los cuáles se puede establecer una comunicación.
- Un socket está asociado a un dominio desde su creación.
- Existen diferentes dominios de comunicación, los formatos reconocidos actualmente son:

Dominio de un socket

- AF_UNIX, AF_LOCAL
- AF_INET
- AF_INET6
- AF_IPX
- AF_NETLINK
- AF_X25
- AF_AX25
- AF_ATMPVC
- AF_APPLETALK
- AF_PACKET
- AF_ALG

Los servicios de sockets son independientes del dominio

Tipos de sockets

- Define las propiedades de las comunicaciones en las que se ve envuelto un socket, esto es, el tipo de comunicación que se puede dar entre cliente y servidor.
- Estas pueden ser:
  - Fiabilidad de la transmisión: Ningún dato transmitido se pierde
  - Conservación del orden de los datos. Los datos llegan en el orden en que han sido emitidos.
  - No duplicación de datos. Solo llega a destino un ejemplar de cada dato emitido

Tipos de sockets

- El «modo conectado» en la comunicación
- Envío de mensajes urgentes

- SOCK_STREAM
- SOCK_DGRAM
- SOCK_RAW
- SOCK_SEQPACKET
- SOCK_RDM
- SOCK_PACKET

Creación de un socket

- La función socket crea un nuevo socket:
  - int socket(int dom, int tipo, int proto)

Interfaz de Sockets

- **Puertos:** identificadores usados para asociar los datos entrantes a un proceso concreto de la máquina.
  - Usados tanto en TCP como en UDP
  - Números de 16 bits
  - 0-1023: reservados por convenio (“puertos bien conocidos”)
    - Asignados a los servidores de servicios básicos

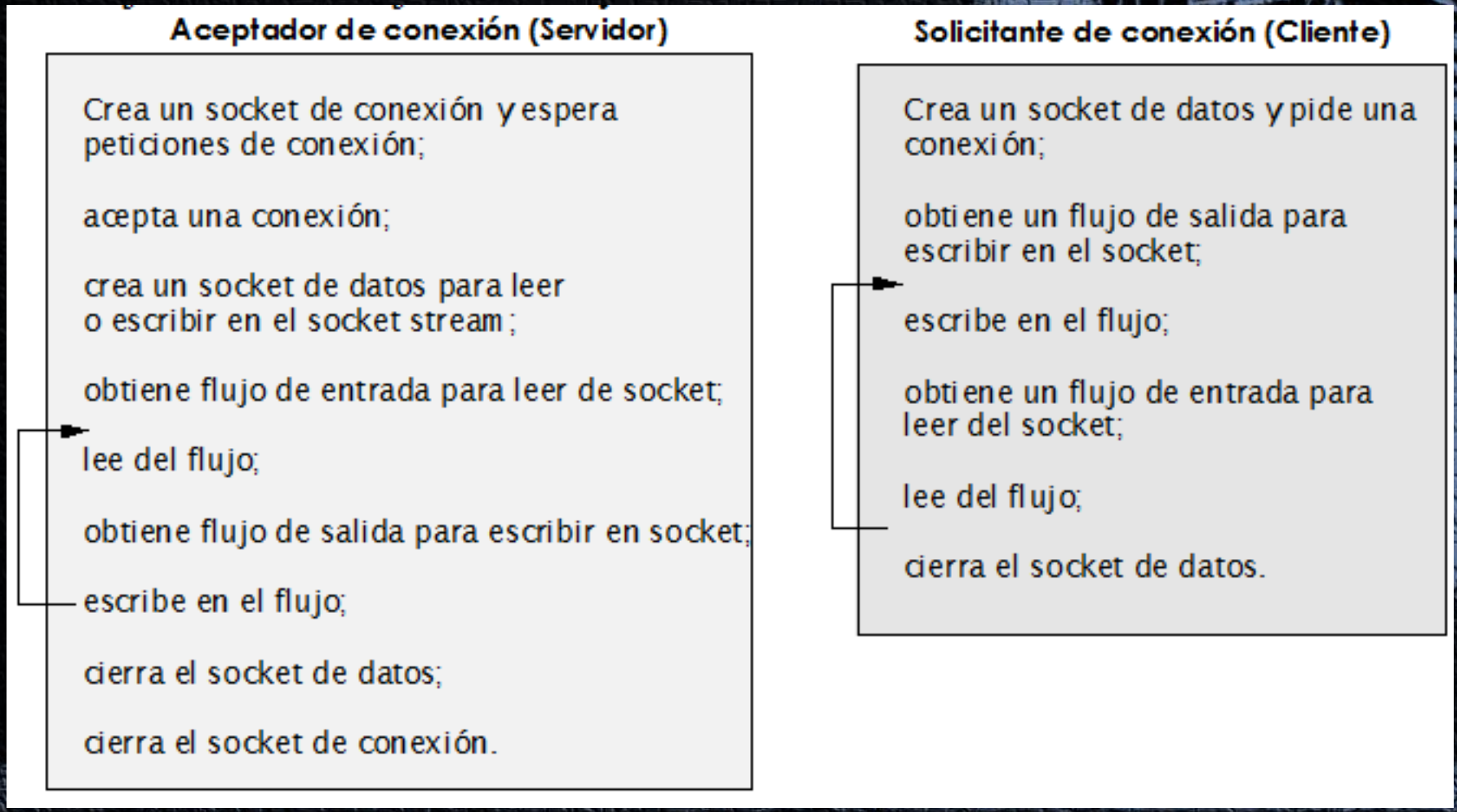
Servicio	Puerto	Servicio	Puerto
ftp	21	dns	53
telnet	22	http	80
ssh	23	pop3	110
smtp	25	https	443

  - 1024-65535: uso libre
    - Son los usados con los clientes al establecer conexiones
    - Suelen asignarse de forma aleatoria

Interfaz de Sockets

- **Direccionamiento:** Para comunicarse con otro proceso usando sockets debe conocerse:
  1. *Dirección IP* (32 bits) de la máquina donde se ejecuta el proceso
     - Alternativamente, su nombre para consultar servicio de nombre DNS (traducción dominio IP)
  2. *No. De puerto* (TCP o UDP) que utiliza el proceso en su máquina
- Por lo tanto, la interfaz de sockets no ofrece transparencia de localización

Implementación de sockets

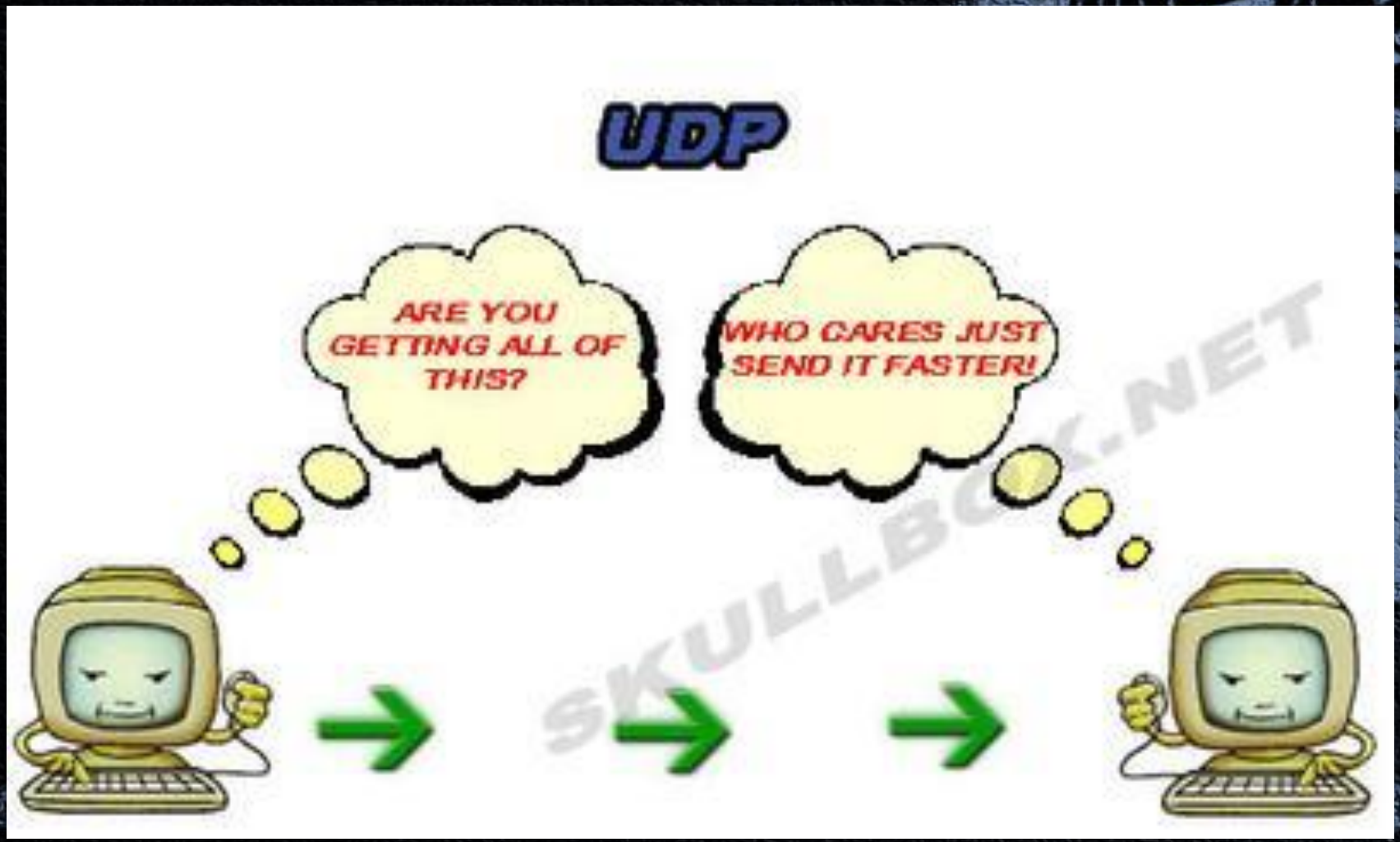


Sockets bloqueantes

Sockets Orientados a Conexión	Sockets No Orientados a Conexión
TCP	UDP
Flujo (stream)	Datagrama

Diferencia entre los tipos de sockets

Sockets UDP


UDP
ARE YOU GETTING ALL OF THIS?
WHO CARES JUST SEND IT FASTER!
SKULLBOX.NET

Sockets UDP

- En las comunicaciones basadas en datagramas (p. ej. UDP), el paquete de datagramas contiene el **número de puerto** de su destino y **UDP** encamina el paquete a la aplicación apropiada.
- El API de Java para UDP proporciona una abstracción del «paso de mensajes».
- Esto hace posible a un proceso emisor transmitir un único mensaje a un proceso receptor.
- Los paquetes independientes que contienen esos mensajes se denominan **datagramas**.

Sockets UDP

- Un datagrama enviado mediante UDP es transmitido desde un proceso emisor a un proceso sin reconocimiento de recomprobaciones.
- Si tiene lugar un fallo, el mensaje puede no llegar.
- Un datagrama es transmitido entre procesos cuando un proceso lo envía y otro proceso lo recibe.
- Cualquier proceso que necesite enviar o recibir mensajes debe en primer lugar crear un socket a una dirección de Internet y a un puerto local.

Sockets UDP

- Un servidor enlazará ese socket a un puerto servidor (uno que es conocido por los clientes de manera que puedan enviarle mensajes).
- Un cliente enlaza su socket a cualquier puerto local libre.
- El método receptor devuelve la dirección de Internet y el puerto del emisor, además del mensaje, permitiendo a los receptores enviar una respuesta.

Sockets UDP

- Tarea

Sockets UDP

- En el caso de Java, proporciona tres clases para dar soporte a la comunicación por medio de datagramas UDP (import java.net.*;)
  - DatagramSocket
  - DatagramPacket
  - MulticastSocket

DatagramPacket

- Proporciona constructores para crear instancias a partir de los datagramas recibidos e instancias de datagramas que serán enviados.

Constructores para datagramas que serán **enviados**

- DatagramPacket(byte[] buf, int length)
- DatagramPacket(byte[] buf, int length, InetAddress address, int port)

Estos constructores crean una instancia de datagrama compuesta por:

  - una cadena de bytes que almacena el mensaje
  - Longitud del mensaje
  - Dirección de Internet
  - Número de puerto local del destinatario

DatagramPacket

Constructores para datagramas **recibidos**

- DatagramPacket(byte[] buf, int offset, int length)
- DatagramPacket(byte[] buf, int offset, int length, InetAddress address, int port)

Estos constructores nos permiten crear instancias de los datagramas recibidos, especificando la cadena de bytes en la que alojar:

  - El mensaje
  - Su longitud
  - Offset* dentro de la cadena

* Este campo soluciona el problema de la secuenciación de fragmentos, indicándole al dispositivo receptor donde debe ser colocado cada fragmento en particular del mensaje original.

DatagramPacket

En esta clase hay métodos para obtener los diferentes componentes de un datagrama, tanto recibido como enviado:

- ✓ getData(): obtiene el mensaje contenido en el datagrama
- ✓ getAddress(): obtiene la dirección IP
- ✓ getPort(): obtiene el puerto

DatagramSocket

- Maneja sockets para enviar y recibir datagramas UDP y proporciona tres constructores:

1. **DatagramSocket()**: constructor sin argumentos que permite que el sistema elija un puerto entre los que estén libres y selecciona una de las direcciones locales.
2. **DatagramSocket(int port)**: constructor que toma un número de puerto como argumento; es apropiado para los procesos que necesitan un número de puerto (servicios).
3. **DatagramSocket(int port, InetAddress laddr)**: constructor que toma como argumentos el número de puerto y una determinada dirección local.

DatagramSocket

La clase DatagramSocket proporciona varios métodos, los más utilizados son:

- send(DatagramPacket p) y receive(DatagramPacket p): transmiten datagramas entre un par de conectores.
  - El argumento de *send* es una instancia de DatagramPacket conteniendo el mensaje y el destino.
  - El argumento de *receive* es un DatagramPacket vacío en el que se coloca: el mensaje, su longitud y su origen
- setSoTimeout(int timeout): permite establecer un tiempo de espera límite
  - Cuando se fija un límite, el método receive se bloquea y luego lanza una excepción.

DatagramSocket

- Connect(InetAddress address, int port): se utiliza para conectarse a un puerto remoto y a una dirección de Internet específicos, en cuyo caso el conector sólo podrá enviar y recibir mensajes de esa dirección.

Serialización