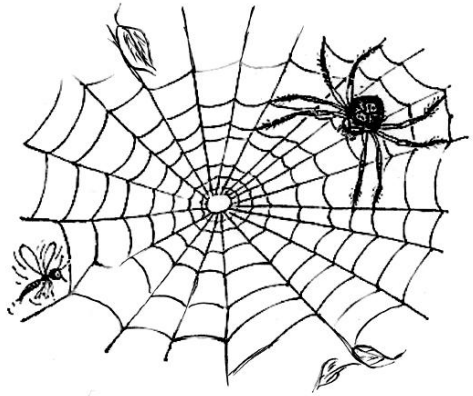
Modulo 5
Protocolos de Transporte



Índice

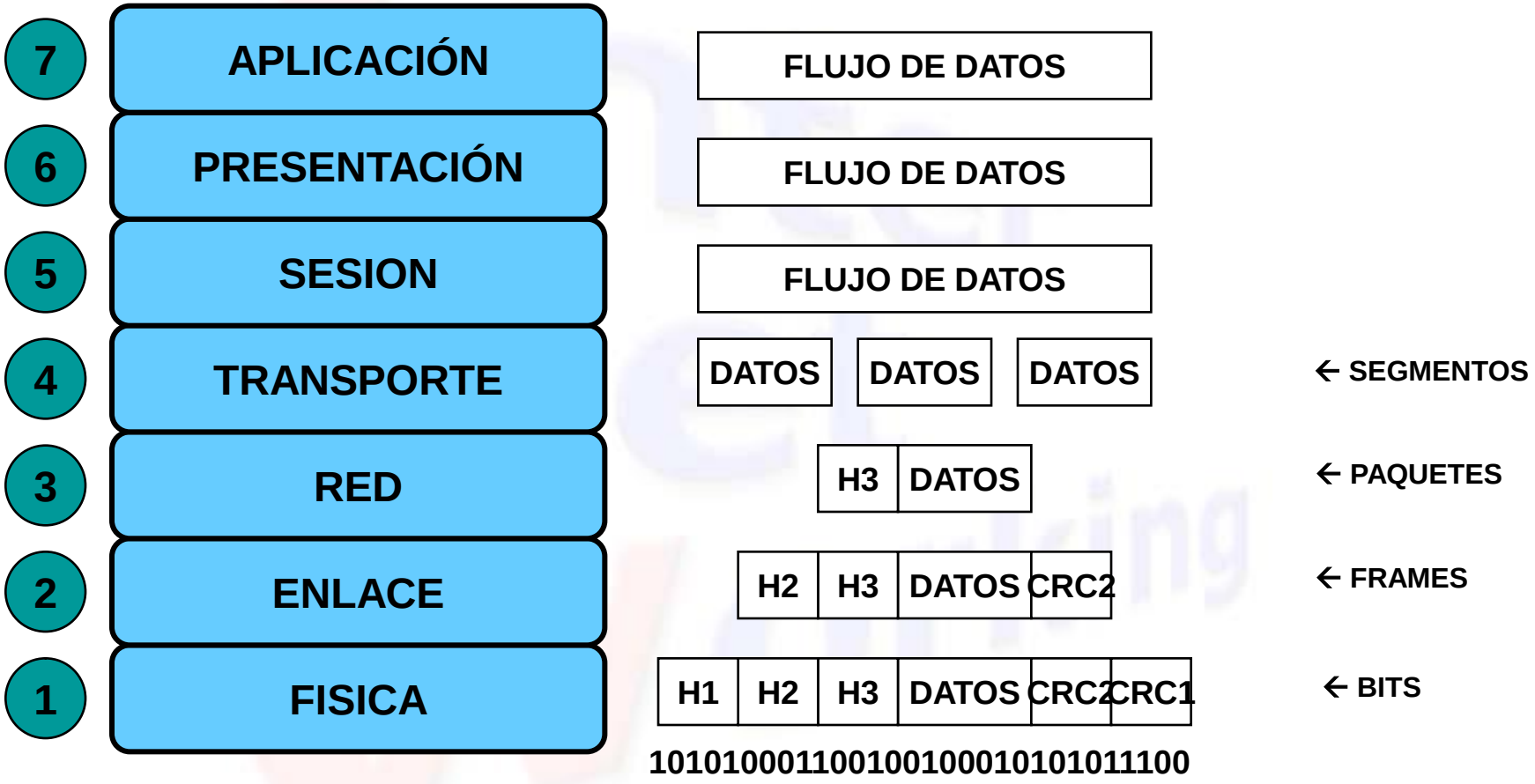
1. Introducción
2. TCP
3. UDP
4. Socket y Puertos

Introducción

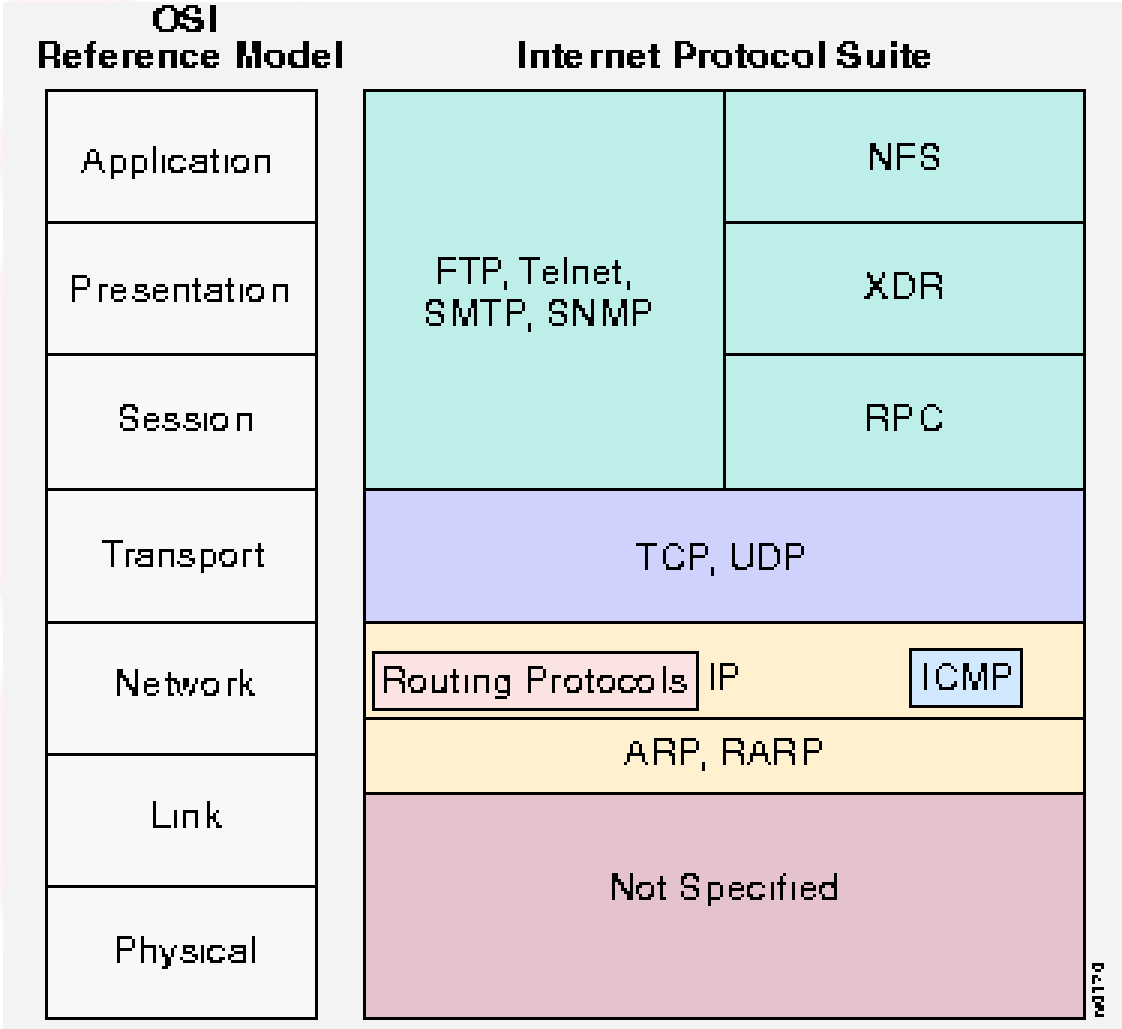
Capa de Transporte

- Confiabilidad del transporte de los datos.
- Administración de circuitos virtuales (creación, mantención, ter minación).
- Detección y recuperación de errores.
- Control de flujo de la información
- Multiplexión

Encapsulamiento de Datos



Comparación con Modelo OSI

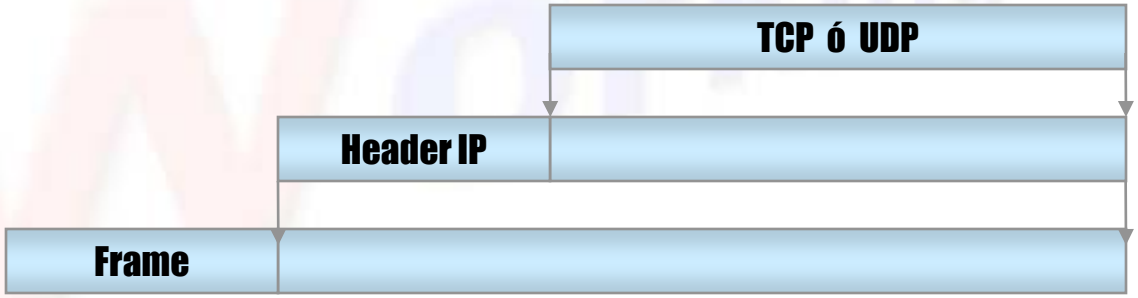


Tipos de Protocolos 1

- Orientados a la Conexión
  - Alta confiabilidad
  - Requieren más procesamiento
- No Orientados a la Conexión
  - Virtualmente no ofrecen confiabilidad
  - Requiere un que la transmisión sea confiable
  - Es muy rápido

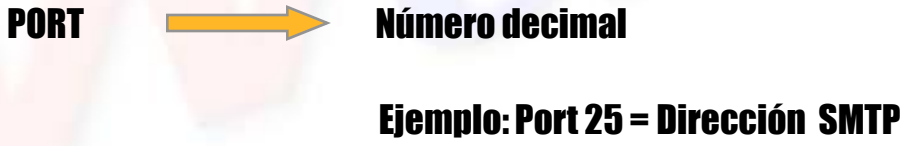
Tipos de Protocolos 2

- Tipos
  - TCP: Transmision Control Protocol
  - UDP: User Datagram Protocol
- Ambos se encapsulan en paquetes IP para proporcionar acceso a servicios que se hayan en hosts remotos

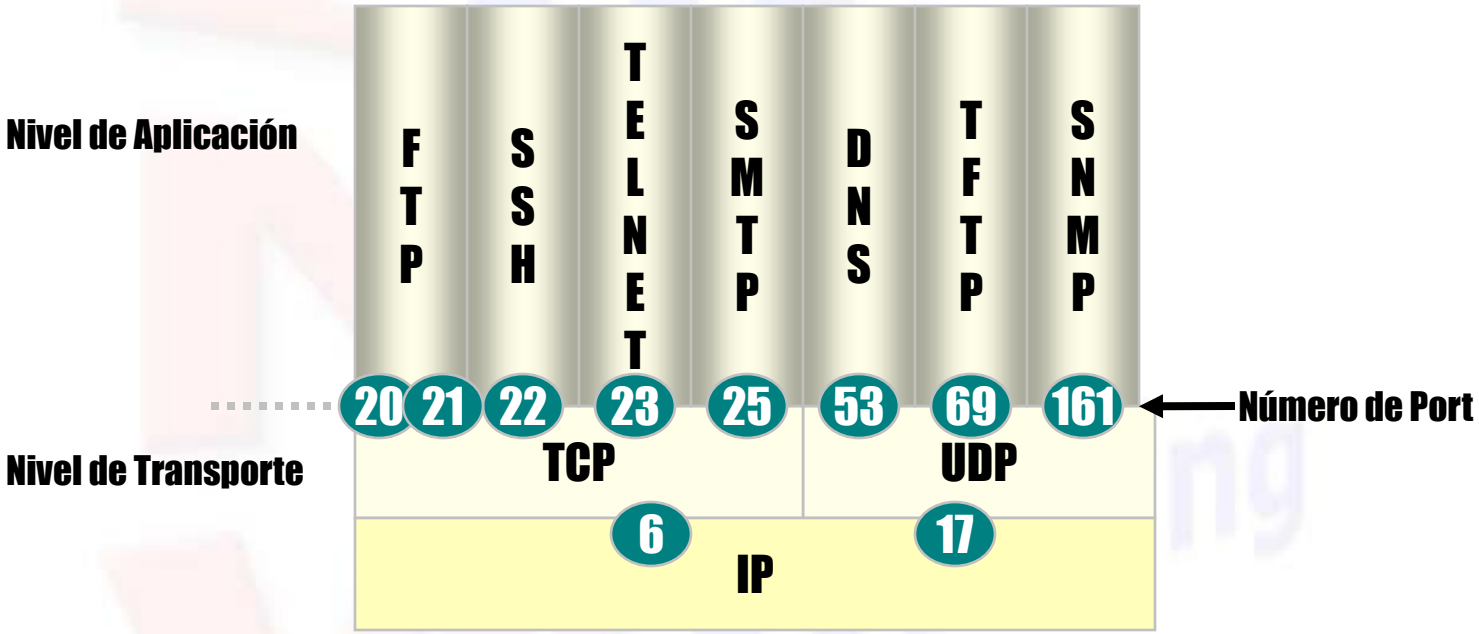


PORT

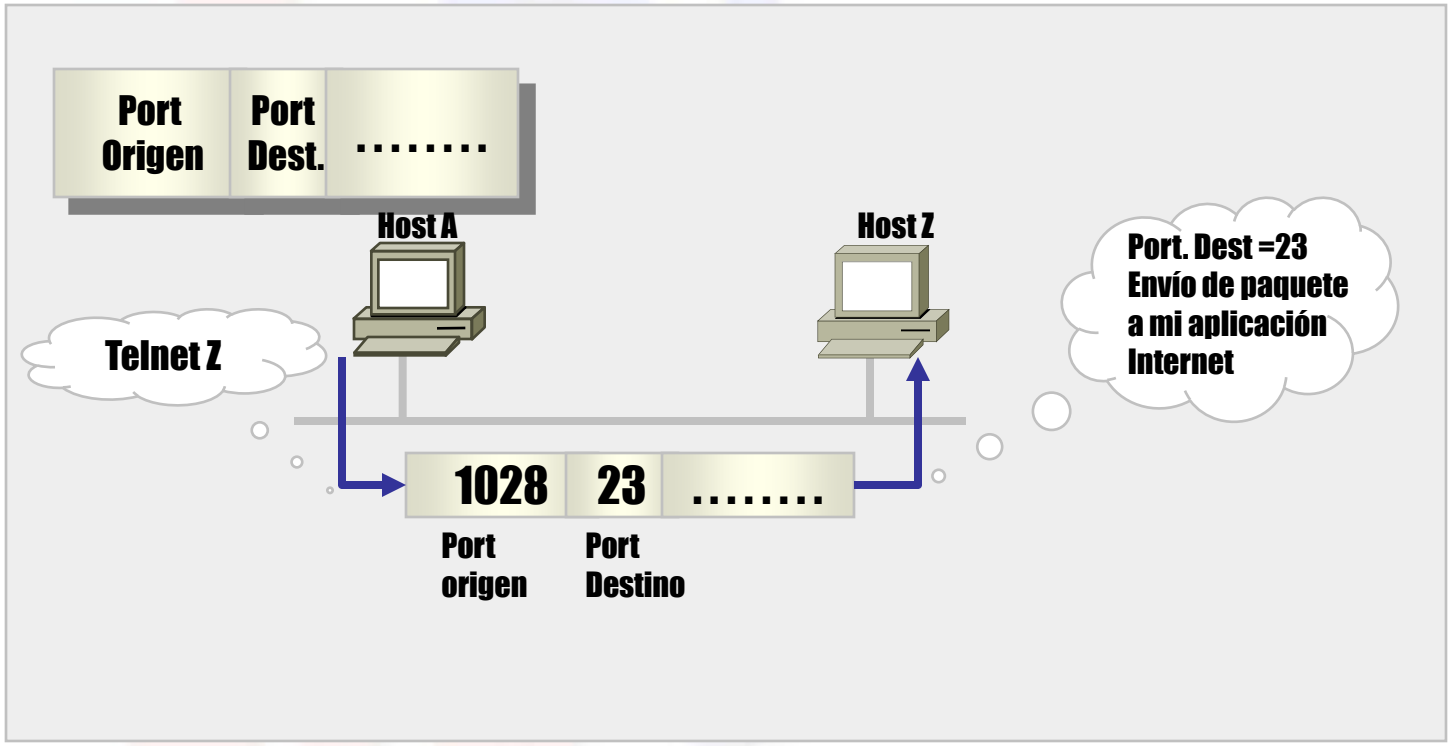
- Para poder ejecutar multiples aplicaciones es un mismo server y distingir los datos entre ellas se requiere otro nivel de direcionamiento
- Cada servicio dispone una dirección única llamada “port”.
- Los números de port éstan en los headers de los paqueter TCP y UDP



Número de Port



Ejemplo de Conexión

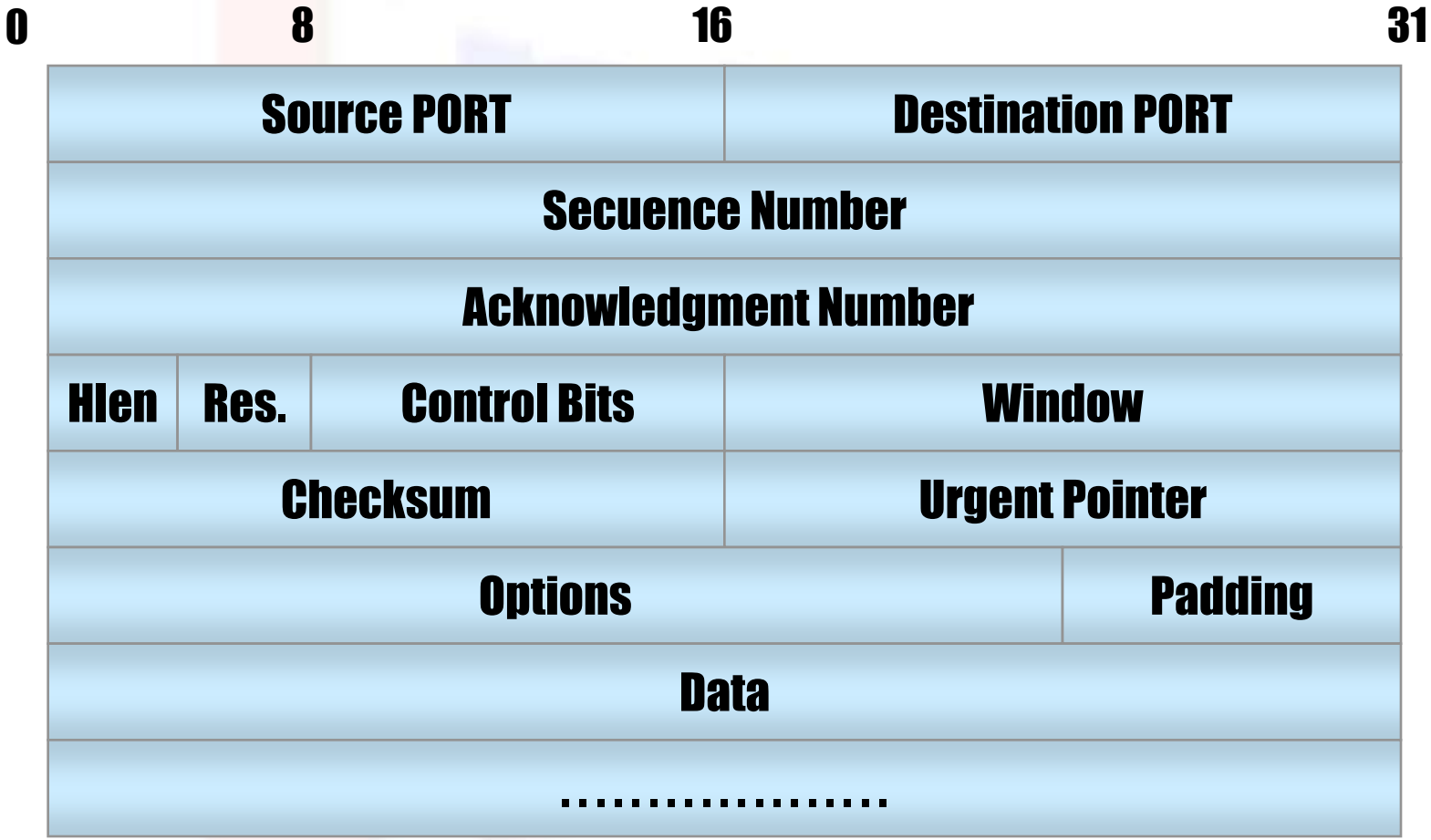


TCP

TCP - Trasport Control Protocol

- Orientado a la Conexión.
- Permite el flujo bidireccional de datos libre de errores.
- Permite Control de Congestión de la Conexión.
- Servicio de Flujo de Datos entre aplicaciones corriendo en diferentes maquinas.
- Permite múltiples conexiones simultáneas
- Confiable, asegura integridad de los datos

Header TCP


0
8
16
31
Source PORT
Destination PORT
Secuence Number
Acknowledgment Number
Hlen
Res.
Control Bits
Window
Checksum
Urgent Pointer
Options
Padding
Data
.....

Descripción de Campos 1

- Source / Destination PORT. Numero de puerto Origen/Destino. 16 bit.
- Secuence number. Número de secuencia del primer byte de la data en este segmento (excepto en SYN). Cuando SYN está presente SN es el ISN (Initial Secuence Number y el primer byte es ISN+1
- Acknowledgment Number. Si el bit ACK está presente indica el próximo SN que se espera recibir.
- HLEN. Largo del header TCP en palabras de 32 bit.

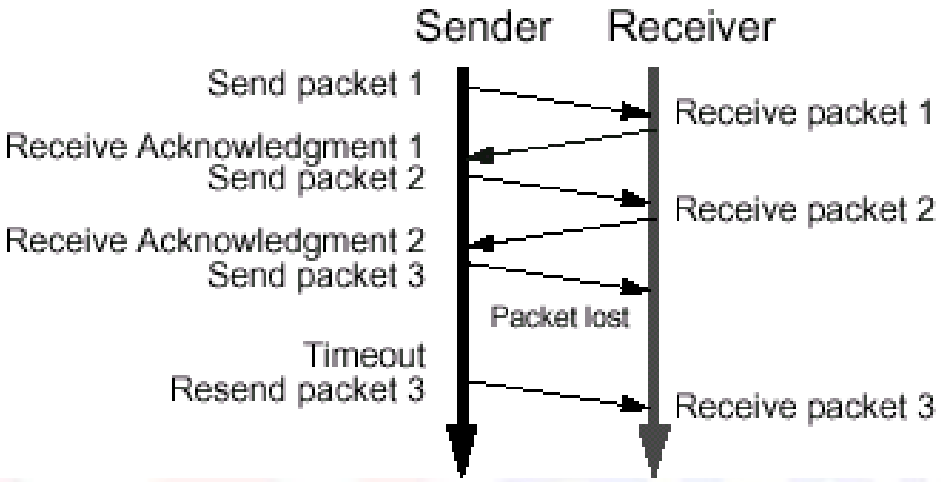
Descripción de Campos 2

- Control Bits. Bits de Control.
  - URG: Urgent Pointer field significant
  - ACK: Acknowledgment field significant
  - PSH: Push Function
  - RST: Reset the connection
  - SYN: Synchronize sequence numbers
  - FIN: No more data from sender
- Window. Número bytes que están en el aire esperando ser aceptados.
- Urgent Pointer. Largo del header TCP en palabras de 32 bit.

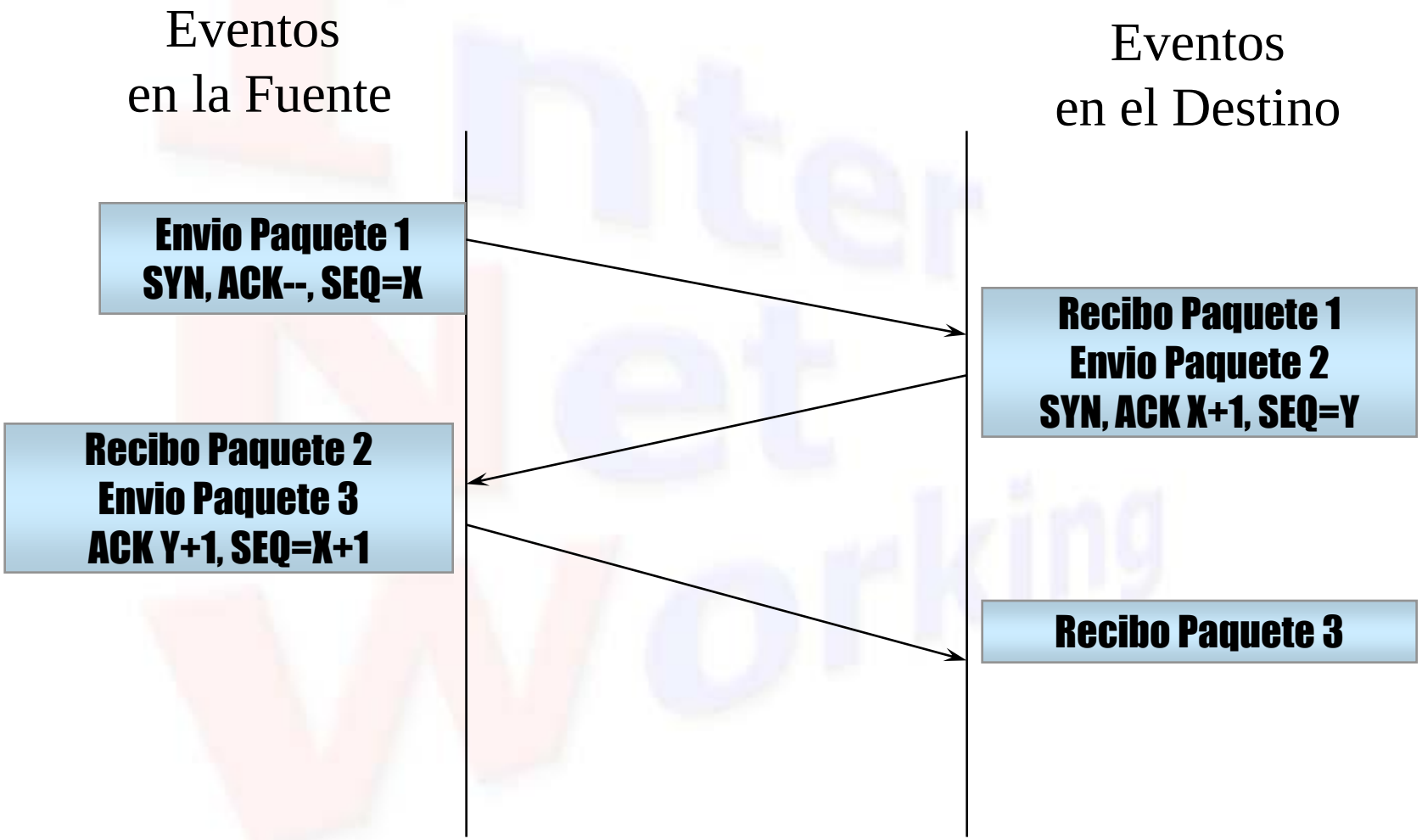
Control de Flujo y Congestión

- ACK con retransmisión.
- Control de flujo
  - Sliding Windows protocols.
  - Stop-and-wait protocols.
- Control de congestión
  - Ventana de Congestión.

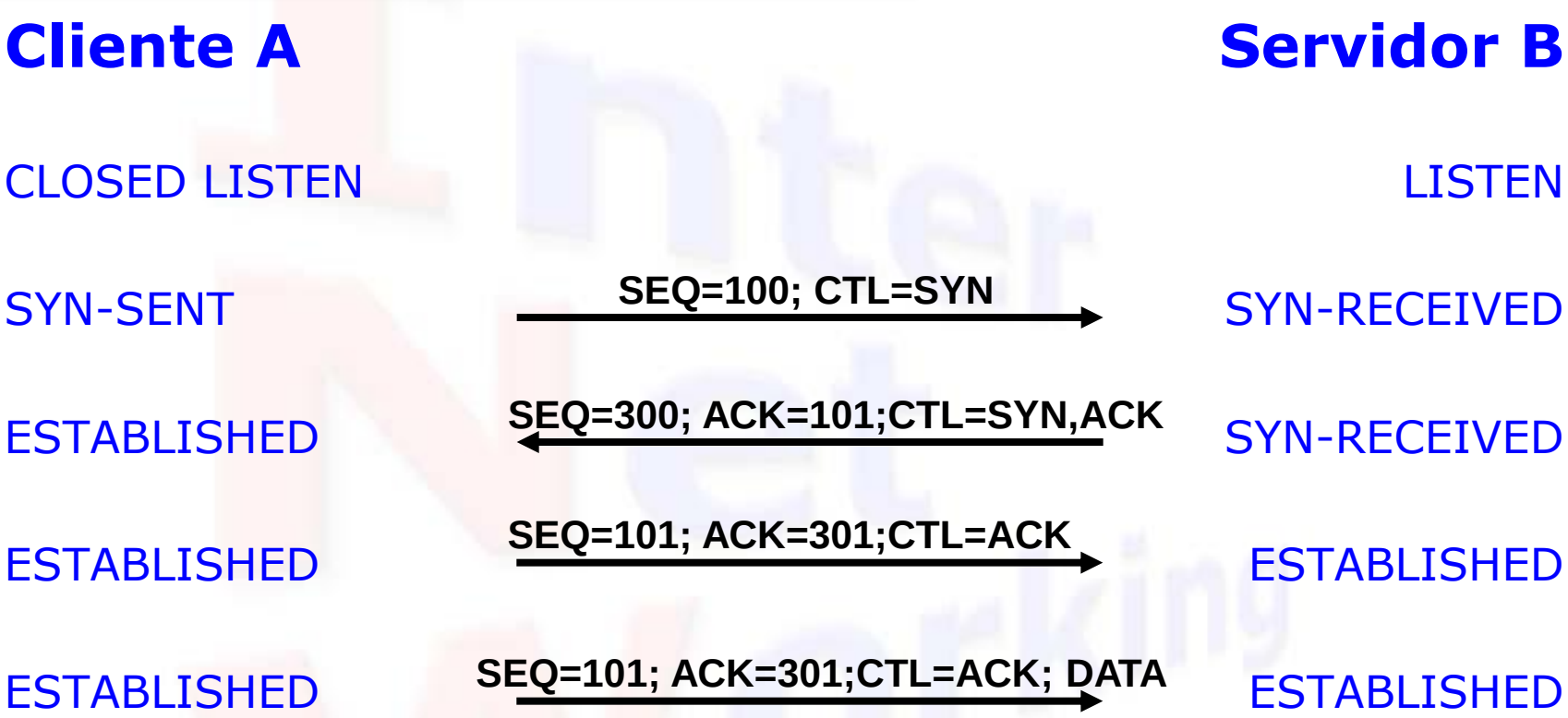
Confiabilidad



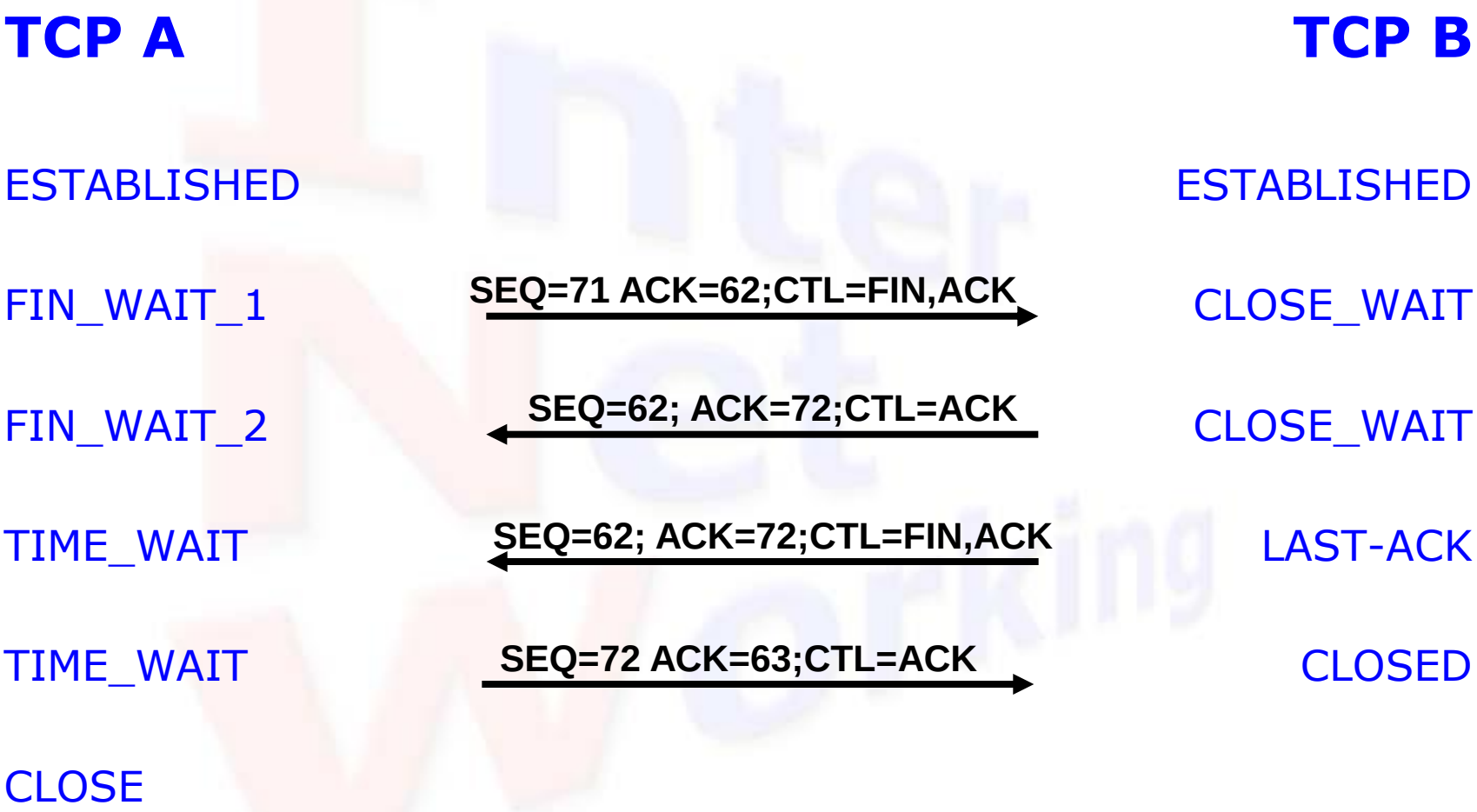
Inicio Conexión TCP: 3 way handshake



Secuencia Normal SYN



Secuencia Normal de FIN

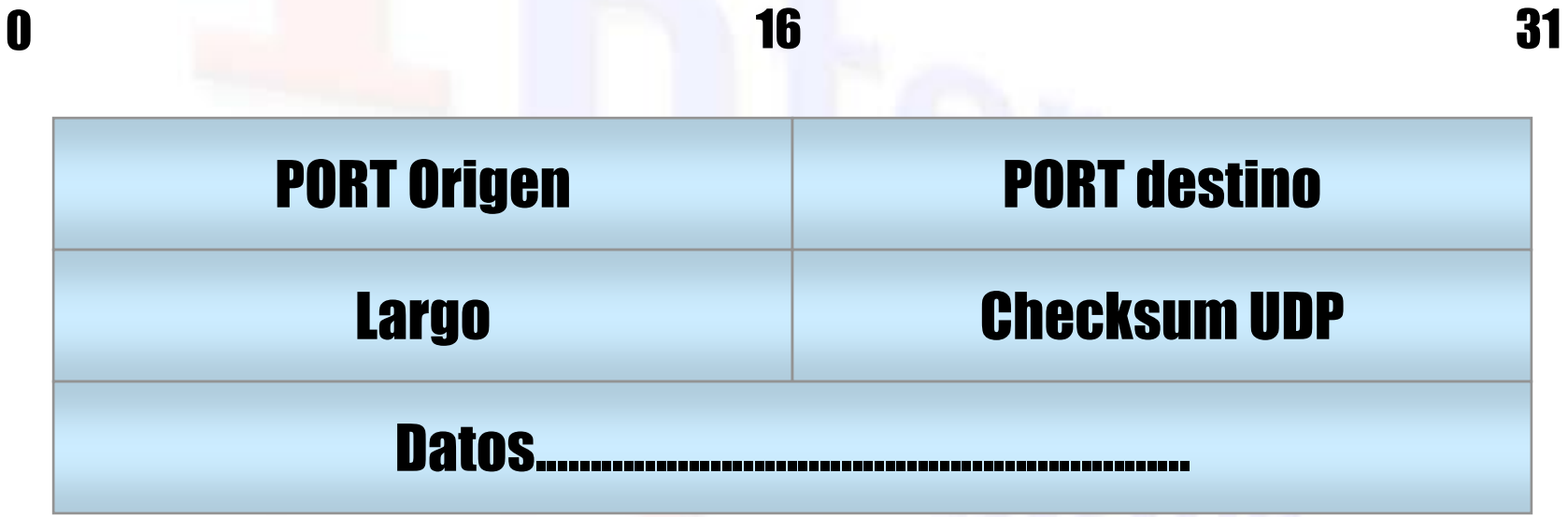


UDP

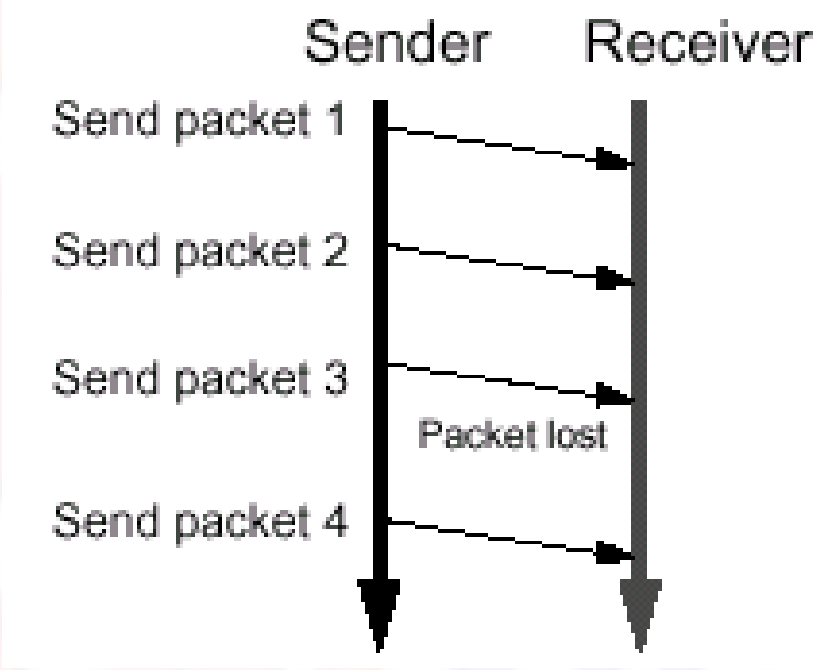
UDP – User Datagram Protocol

- No orientado a la Conexión
- permite flujo bidireccional de datos
- permite múltiples conexiones simultáneas
- Acceso directo de las aplicaciones al datagrama.
- Mínimo overhead.
- No Realiza Corrección de Errores
- No Hay Control de Flujo ni Congestión

Header UDP

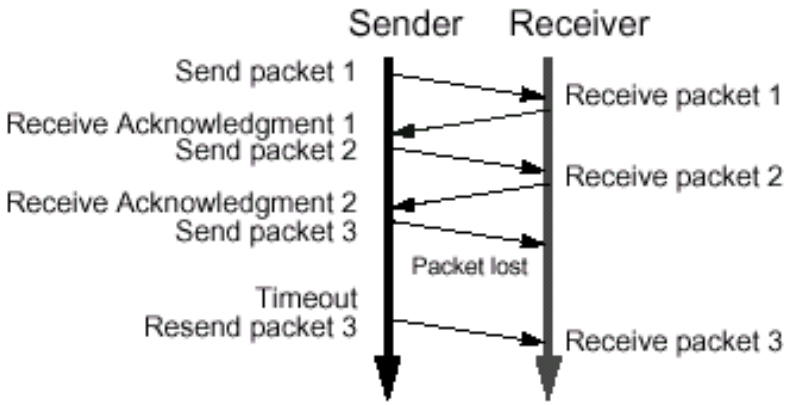


Confiabilidad

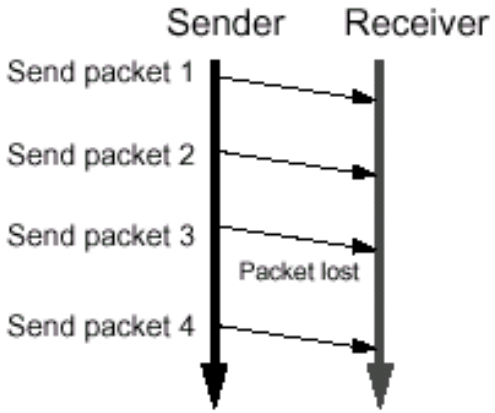


Comparación

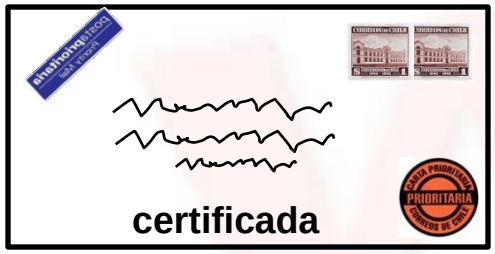
Confiabilidad TCP vs UDP

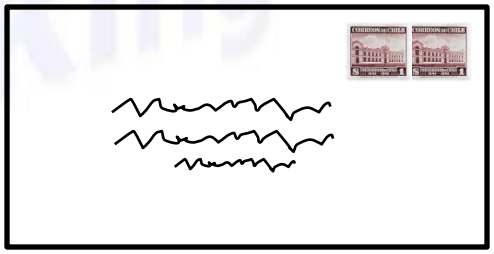


A TCP Reliable Protocol



A UDP Unreliable Protocol





Protocolos Basados en Datagramas

- Imponen un tamaño máximo de datos a ser enviados en cada transmisión.
  - Ejemplos : Ethernet, IP y UDP
- Existe un límite en el monto a ser transmitidos en cada transmisión

Protocolos Orientados a flujo (Stream)

- Ejemplo : TCP, No es necesario preocuparse del tamaño máximo de datos a transmitir.
- TCP divide en trozos más pequeños la transmisión, retransmite partes perdidas, re-ordena los datos distribuidos fuera de orden.
- El overhead necesario para soportar TCP es proporcionalmente mayor que el de UDP.
- Una aplicación que use TCP requiere más memoria y más ancho de banda para asegurar que la transmisión se complete apropiadamente.

Protocolos Confiables v/s No Confiables

- UDP
  - Protocolo no confiable o protocolo de mejor esfuerzo (Best effort).
  - Una aplicación que use UDP, es la responsable de realizar las retransmisiones, filtrar duplicidades, etc.
  - Ejemplos : UDP, Ethernet, IP

Protocolos Confiables v/s No Confiables (cont)

- TCP
  - Protocolo confiable.
  - TCP fragmenta y reemplaza el flujo de datos (para que los datos se adapten al máximo permitido por IP), retransmite paquetes perdidos, filtra datos duplicados, maneja un control de flujo velocidades entre computadores de diferentes
  - Las aplicaciones que invocan sesiones, generalmente usan TCP para la Transferencia de datos.

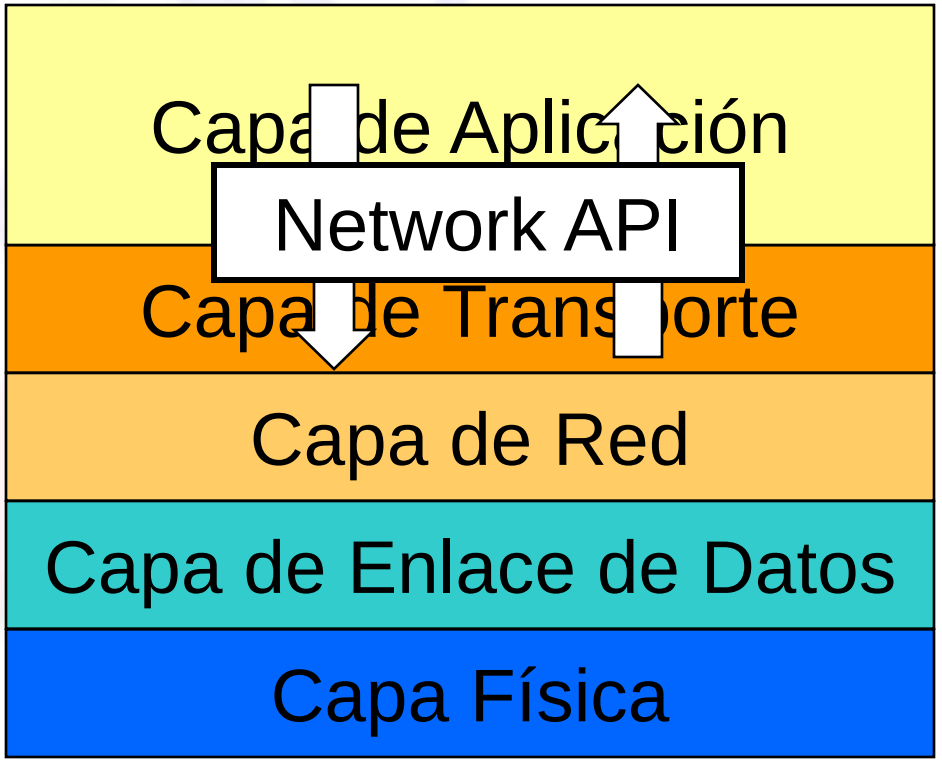
Socket

Socket

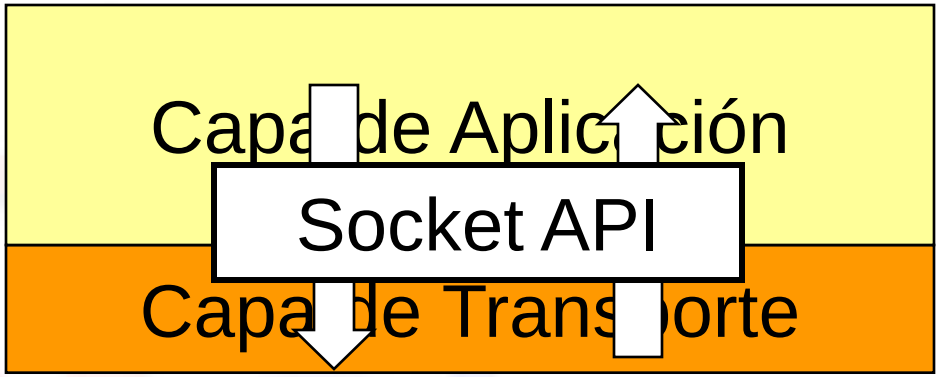
- Un socket es una representación abstracta de un endpoint de comunicación
- Los sockets UNIX trabajan con servicios I/O tal como los archivos, pipes y FIFOs
- Sockets (claramente) necesitan:
  - establecer una conexión
  - especificar las direcciones de los endpoints

Socket

La mayoría de los sistemas han adoptado ***Socket API*** (disponible en la mayoría de los S.O., p.e. Linux, UNIX, Windows)



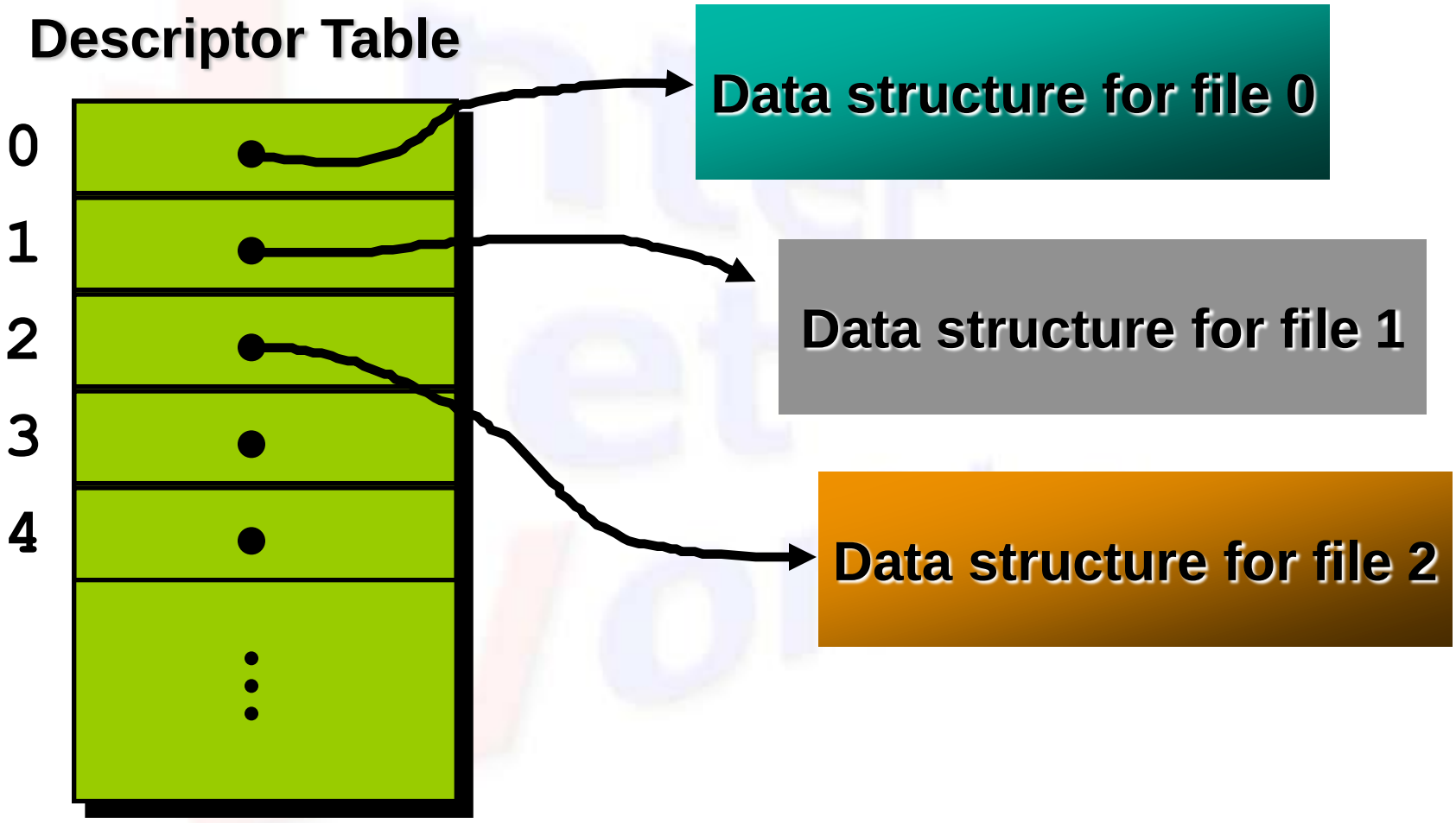
Socket



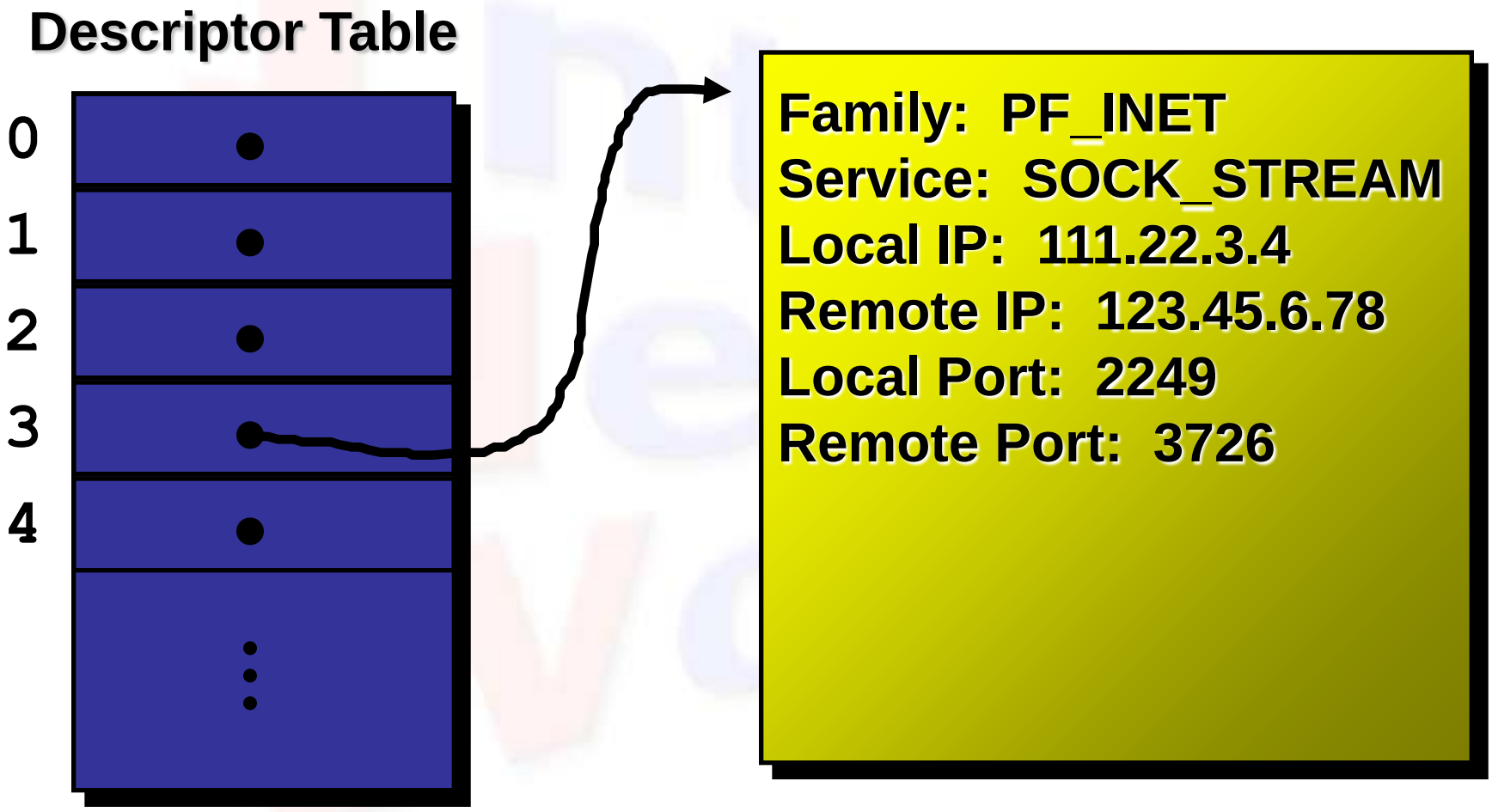
◆ En algunos S.O. las funciones de Socket API están integradas en el S.O. (**p.e. BSD UNIX**)

◆ En otros, una biblioteca entrega las funciones necesarias para comunicar la Aplicación con la Capa de Transporte (**p.e. *socket.h***).

Unix Descriptor Table

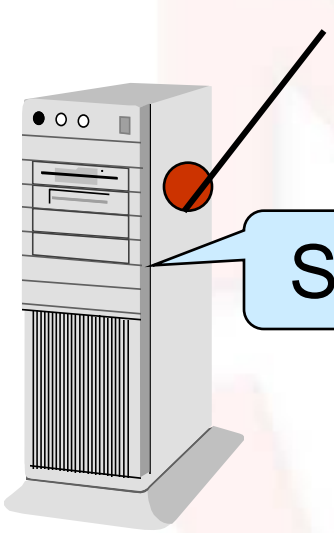


Socket Descriptor Data Structure



Socket API (sockets)

- Los sockets operan según el paradigma *open-read-write-close* (UNIX I/O).



Análogamente al trabajo con archivos, ***open*** retorna un descriptor a un socket.

Creando un Socket

- int socket(int family,int type,int proto);

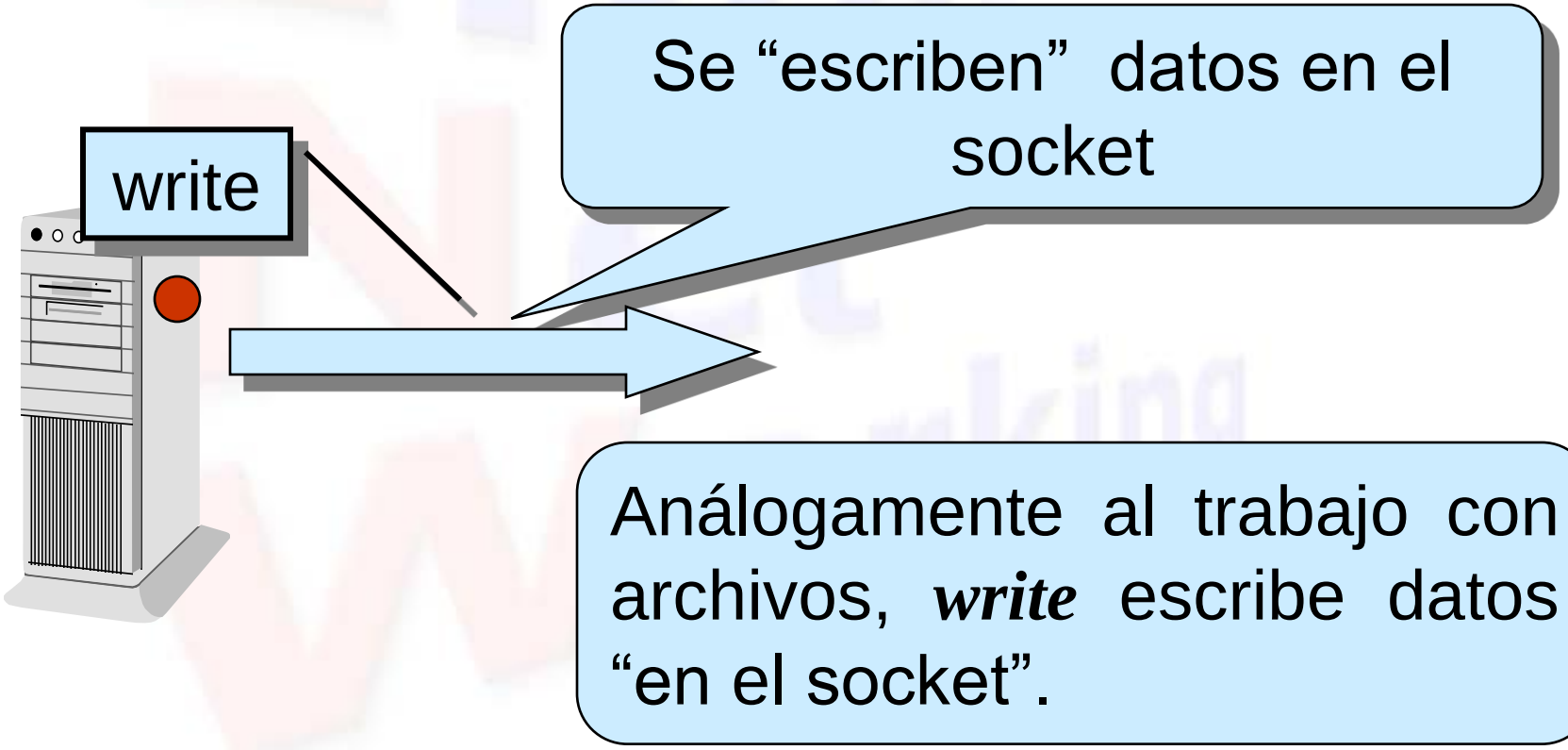
- family especifica la familia de protocolos (PF_INET para TCP/IP).
- type especifica el tipo de servicio (SOCK_STREAM, SOCK_DGRAM).
- protocol especifica el tipo de protocolo (0 usualmente significa default).

Socket API (socket)

Los sockets operan según el paradigma *open-read-write-close* (UNIX I/O).

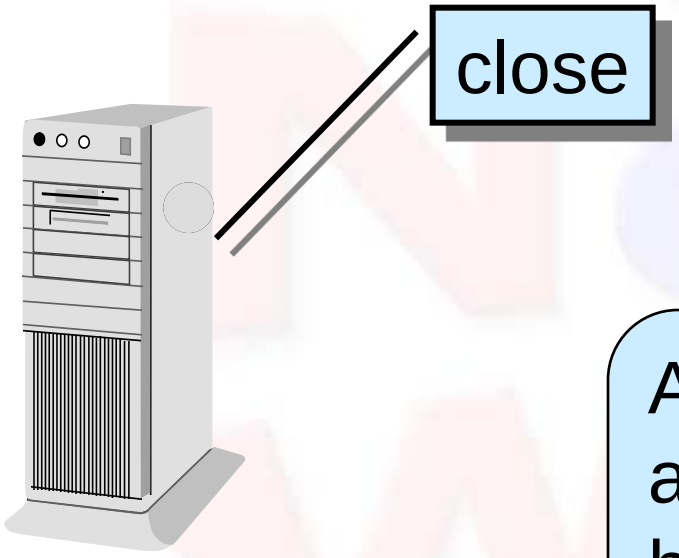
Socket API (socket)

Los sockets operan según el paradigma *open-read-write-close* (UNIX I/O).



Socket API (socket)

Los sockets operan según el paradigma *open-read-write-close* (UNIX I/O).



Análogamente al trabajo con archivos, ***close*** indica que se ha terminado de usar el socket.

Socket API (socket)

- En realidad, la comunicación usando sockets es algo más complicada que open-read-write-close.

- Por ejemplo, es necesario especificar:
  - protocolo de transporte
  - dirección destino
  - tipo de dirección destino
  - si se trata de un servidor, puerto por el que recibe las peticiones
  - etc.

Comunicación usando sockets

Ambos crean un ***socket*** para comunicarse a través de la red

Servidor

Familia de protocolos a usar en la com

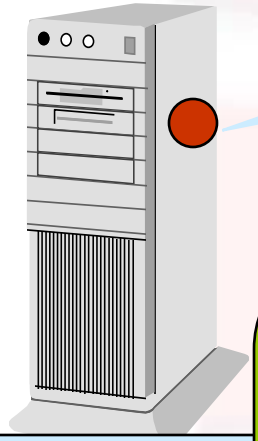
descriptor =

AF_UNIX: Comunicación interna
AF_INET: TCP/IP
AF_APPLETALK: Apple Talk
AF_NS: Xerox NS
AF_CCITT: Protocolos CCITT, X25, etc.
AF_SNA: IBM SNA
AF_DECnet: DECnet

Comunicación usando sockets

Ambos crean un ***socket*** para comunicarse a través de la red

Servidor



Tipo de comunicación que usará el socket

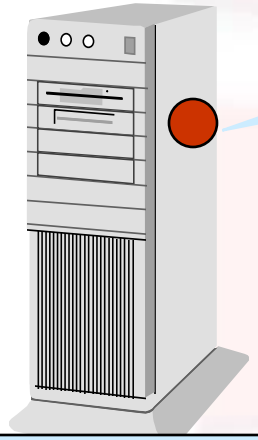
descript

SOCK_DGRAM: Modo no conectado
SOCK_STREAM : Modo conectado
SOCK_RAW : Acceso a protocolos de más bajo nivel (p.e. IP). Superusuario.
SOCK_SEQPACKET: Para comunica-ciones con Xerox NS.

Comunicación usando sockets

Ambos crean un ***socket*** para comunicarse a través de la red

Servidor



Protocolo de transporte que usará el socket

nte

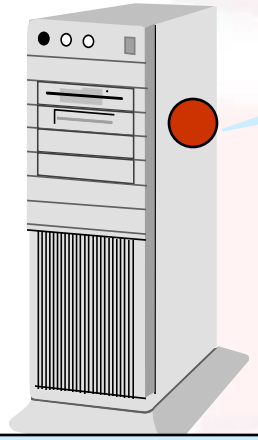
descriptor = socket (protofamily, type, protocol)

0: protocolo por defecto

Comunicación usando sockets

Ambos crean un ***socket*** para comunicarse a través de la red

Servidor



Cli nte

descript ***fa***

Modo conectado

Familia TCP/IP

Protocolo de transporte por defecto (TCP)

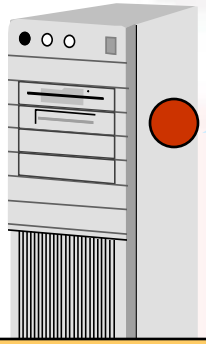
Por ejemplo:

sock=socket(PF_INET,SOCK_STREAM,0)

Comunicación usando sockets

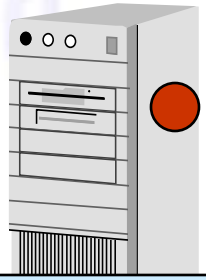
El servidor asocia el descriptor del socket al **puerto** por el que recibirá las peticiones y a la **dirección local**

Servidor



Descriptor del socket retornado por la función socket

Cliente



bind (descriptor, dir_local, largo_dir_local)

Comunicación usando sockets

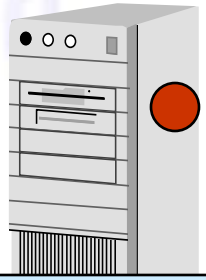
escriptor
r el que
y a la

Estructura que contiene la dirección local:

0	16	31
Familia	Dirs.	Dir_Octetos 0 -1
	Octetos 2 -5	
	Octetos 6 -9	
	tetos 10 - 13	

```
struct sockaddr {
        u_short     sa_family;
        char        sa_data[14];
};
```

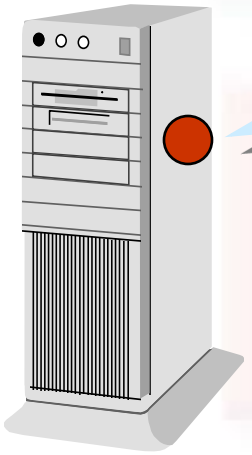
Cliente



dir_local)

Comunicación usando sockets

Servidor



El servidor asocia el descriptor del socket al **puerto** por el que recibirá las peticiones y a la **dirección local**

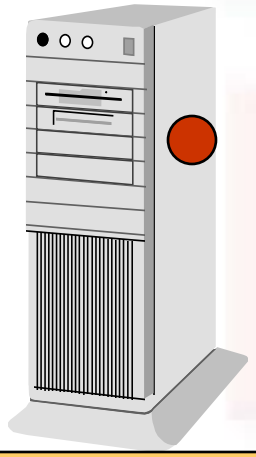
Largo de la dirección local (medido en bytes)

bind (descriptor, dir_local, largo_dir_local)

Comunicación usando sockets

El servidor queda en espera...

Servidor



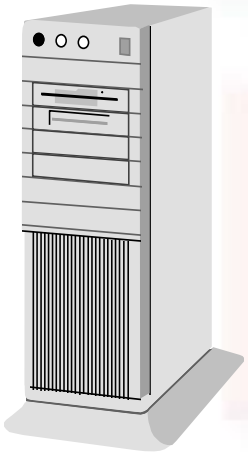
Especifica el largo de la fila de peticiones de clientes esperando ser atendidas

Descriptor del socket

listen(descriptor, largo_cola)

Comunicación usando sockets

Servidor



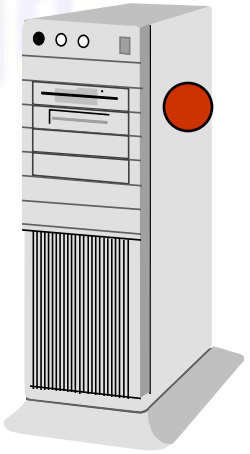
Resumen actividades hechas por el servidor (hasta ahora):

- Crea un socket (***socket***)
- Le asocia un puerto y una dirección local (***bind***)
- Queda en espera (***listen***)

Comunicación usando sockets

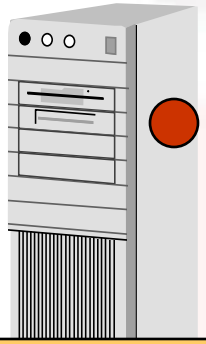
Y el cliente... una vez creado el socket...

Cliente



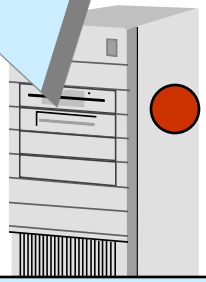
Comunicación usando sockets

Servidor



Se conecta (sólo si se trata de un servicio SOCK_STREAM) al servidor a través de su socket. Para esto le asocia el puerto del servidor y su dirección .

ente



Descriptor del socket retornado por la función socket

connect (descriptor, dir_destino, largo_dir_destino)

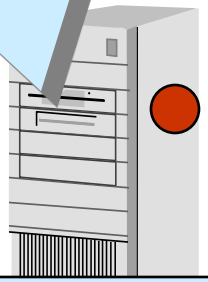
Comunicación usando sockets

Estructura que contiene la dirección destino:

0	16	31
Familia de Dirs.	Dir_Octetos 0 -1	
Dir_Octetos 2 -5		
Dir_Octetos 6 -9		
Dir_Octetos 10 - 13		

ata de un
AM) al
socket.
uerto del
ón.

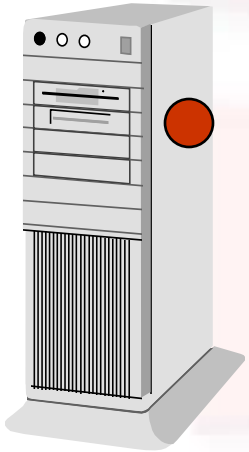
iente



connect (descriptor, dir_destino, largo_dir_destino)

Comunicación usando sockets

Servidor



Se conecta (sólo si se trata de un servicio SOCK_STREAM) al servidor a través de su socket. Para esto le asocia el puerto del servidor y su dirección.

Largo de la dirección destino(medido en bytes)

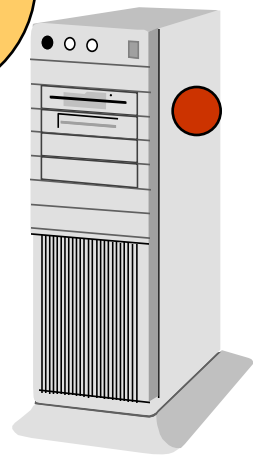
connect (descriptor, dir_destino, largo_dir_destino)

Comunicación usando sockets

Resumen actividades hechas por el cliente (hasta ahora):

- Crea un socket (***socket***)
- Le asocia un puerto y una dirección destino (***connect,*** sólo en caso de un servicio con conexión)

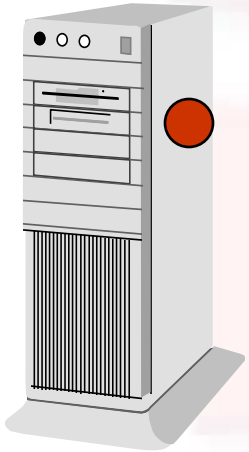
Cliente



Comunicación usando sockets

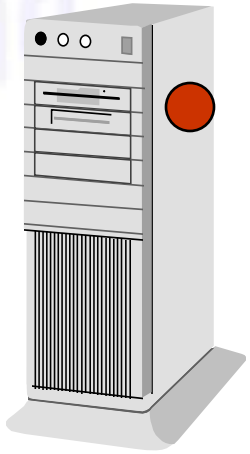
Se inicia la transferencia de información

Servidor



read, readv, recv, recvfrom, recvmsg

Cliente

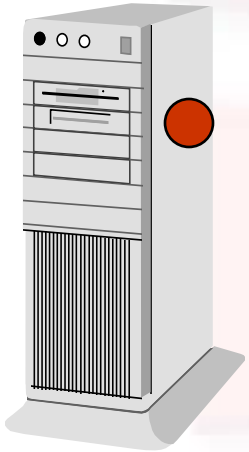


write, writev, send, sendto, sendmsg

Comunicación usando sockets

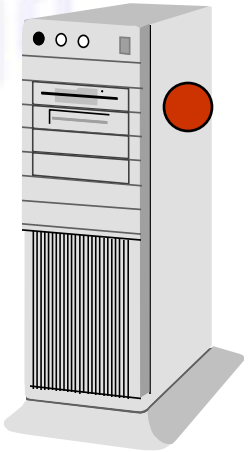
Se inicia la transferencia de información

Servidor



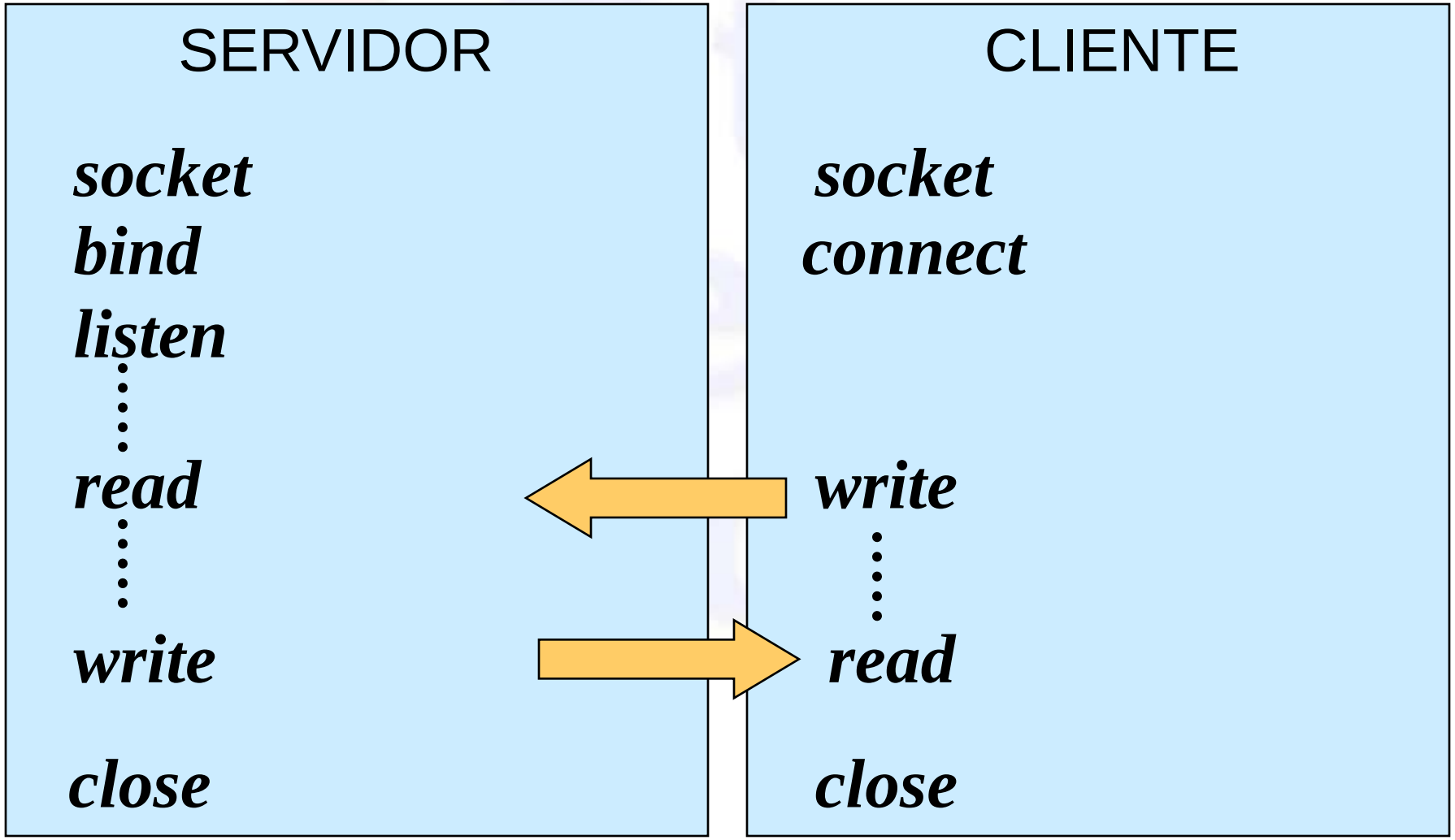
write, writev, send, sendto, sendmsg

Cliente



read, readv, recv, recvfrom, recvmsg

Resumen de Conexión



./etc/services (primeras líneas)

```
tcpmux 1/tcp
echo                    7/tcp
echo                    7/udp
discard                 9/tcp           sink null
discard                 9/udp           sink null
systat                  11/tcp          users
daytime                 13/tcp
daytime                 13/udp
netstat                 15/tcp
chargen                 19/tcp          ttytst source
chargen                 19/udp          ttytst source
ftp-data                20/tcp
ftp                     21/tcp
telnet                  23/tcp
smtp                    25/tcp          mail
```

Preguntas