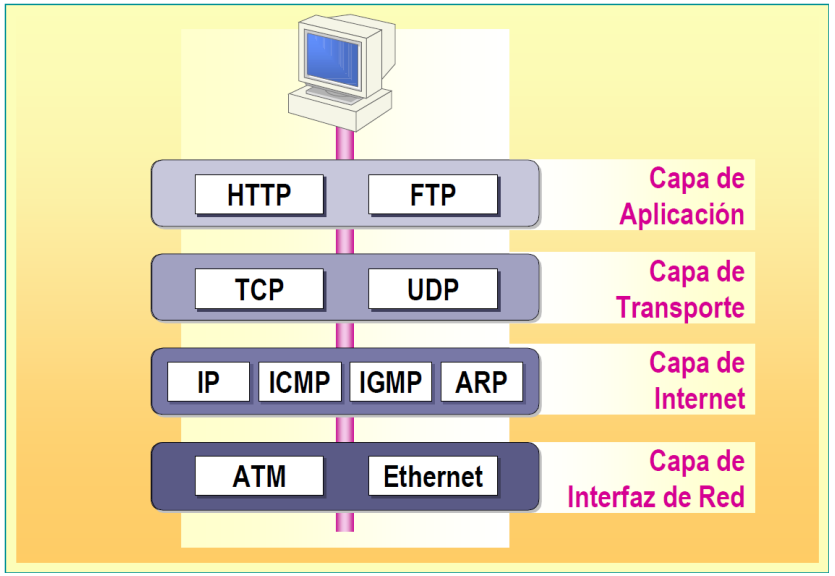
Protocolos TCP y UDP

- TCP y UDP son protocolos de la capa de transporte
  - Garantiza la entrega de mensajes entre dos entidades remotas (programas)
  - Utiliza el protocolo de red (IP)

- **T**ransmission **C**ontrol **P**rotocol
- **U**ser **D**atagram **P**rotocol



Protocolos TCP y UDP: puertos

- IP conecta dos ordenadores remotos
  - Utiliza direcciones IP
  - Sin embargo, el destinatario de un mensaje no es el ordenador, sino cierta aplicación que corre en el ordenador
- TCP y UDP conectan dos aplicaciones remotas
  - Utilizan **puertos**

Protocolo UDP

- User Datagram Protocol
- Sólo envía paquetes (datagramas) entre una aplicación y otra
  - Los datagramas pueden llegar en un orden diferente al enviado (si IP elige rutas distintas para ellos)
  - El emisor no tiene la seguridad de que los datagramas llegan al receptor
  - No se utilizan conexiones. Cada datagrama se envía de forma independiente.

Protocolo UDP

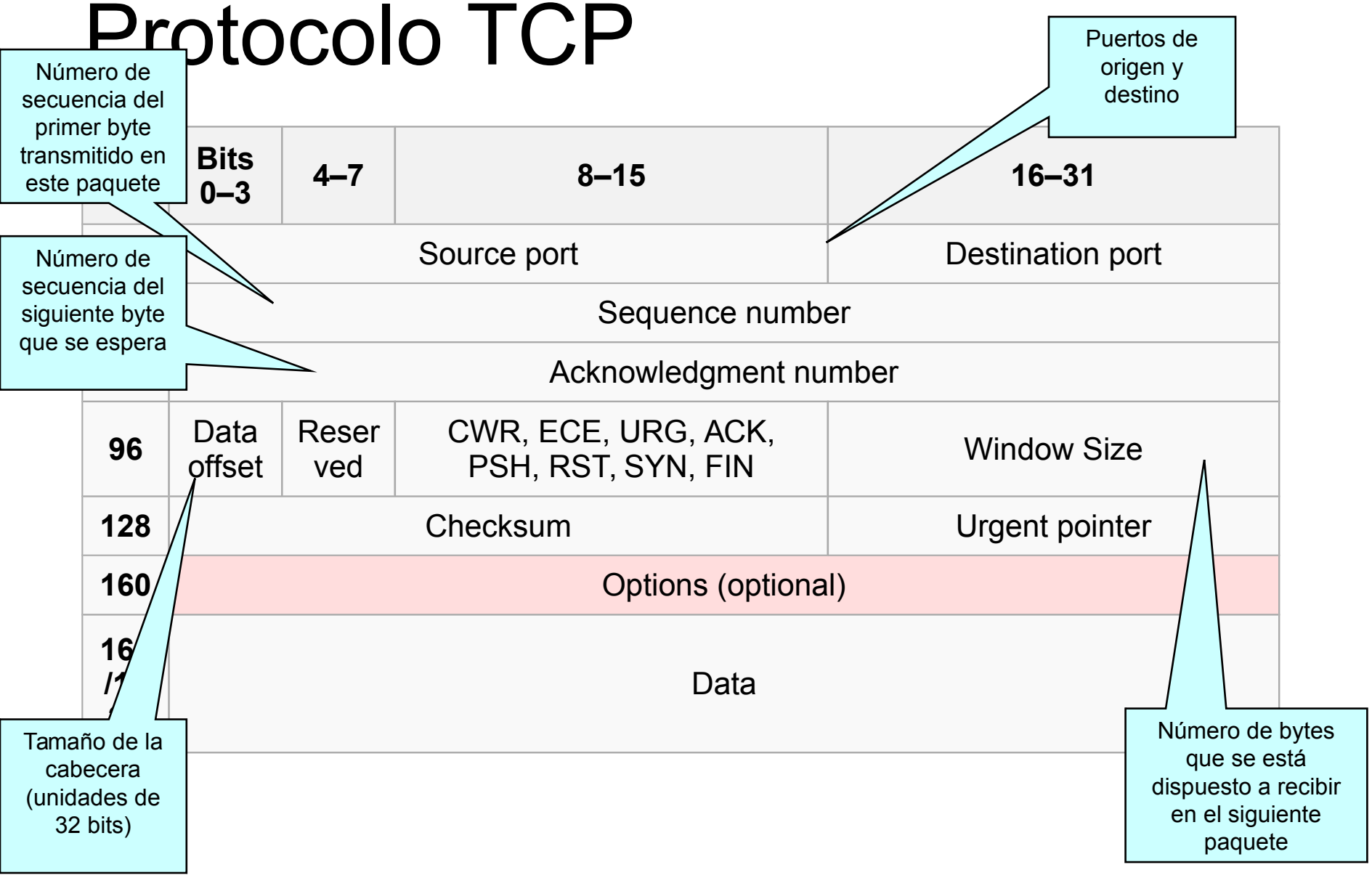
	Bits 0 - 15	16 - 31
0	Source Port	Destination Port
32	Length	Checksum
64	Data	

- Preguntas
  - ¿Cuántos puertos hay?
  - ¿Cuál es el tamaño máximo de un paquete UDP?

Protocolo TCP

- Transmission Control Protocol
- Asegura que la transmisión se realiza por un medio fiable
  - Garantiza la recepción de los mensajes en orden correcto
  - Garantiza al emisor que los mensajes llegan correctamente al receptor
  - Utiliza conexiones (o sesiones), que deben mantenerse activas durante la comunicación

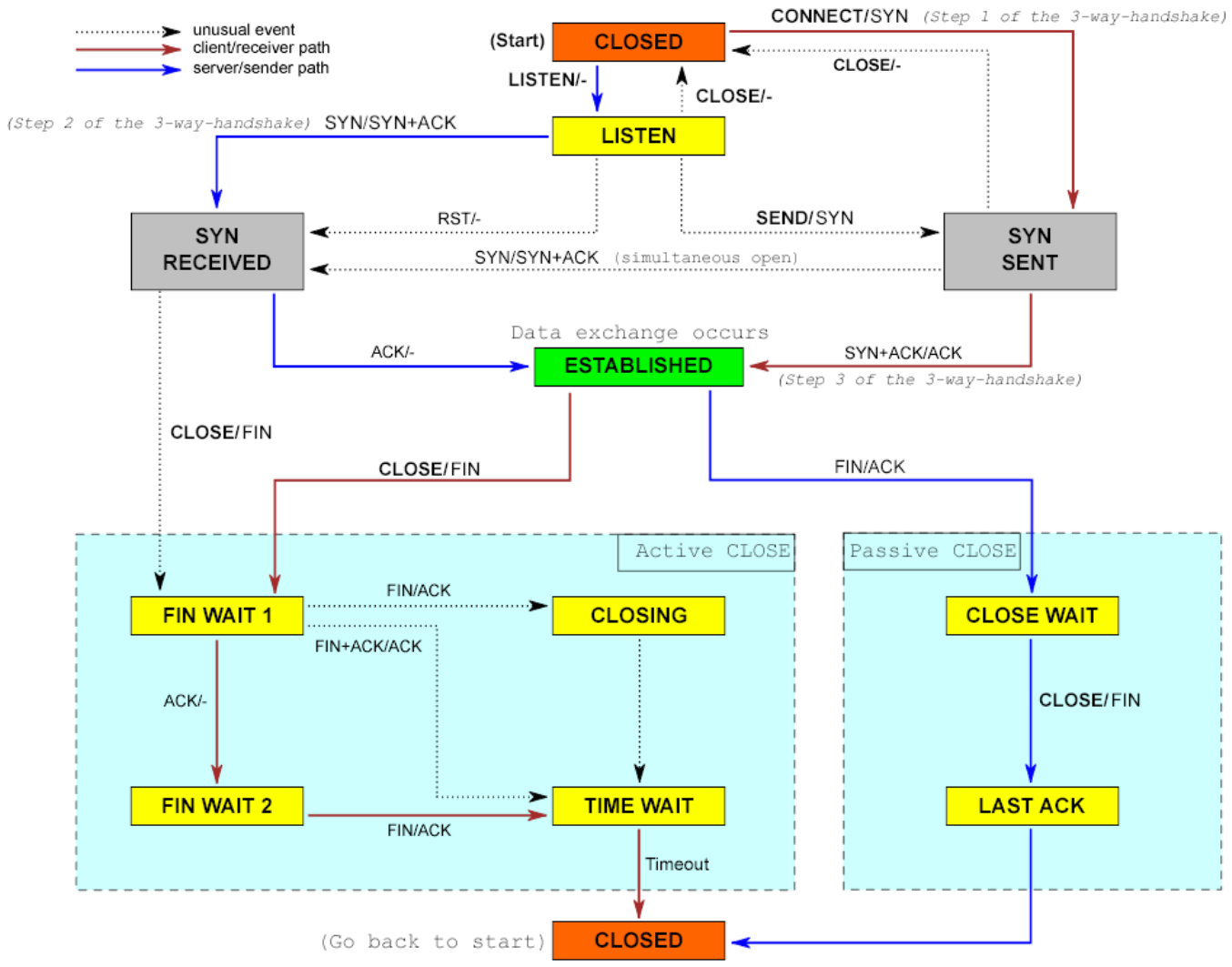
Protocolo TCP



Estados TCP

- Las aplicaciones que usan TCP necesitan sincronizarse para
  - Iniciar la comunicación: mientras un programa *"escucha"* otro se conecta
  - Terminar la comunicación: aunque un sentido de la comunicación esté cortado, el otro puede seguir activo
  - Saber qué tramas ha recibido el otro extremo, cuales deben volver a enviarse,…

Estados TCP

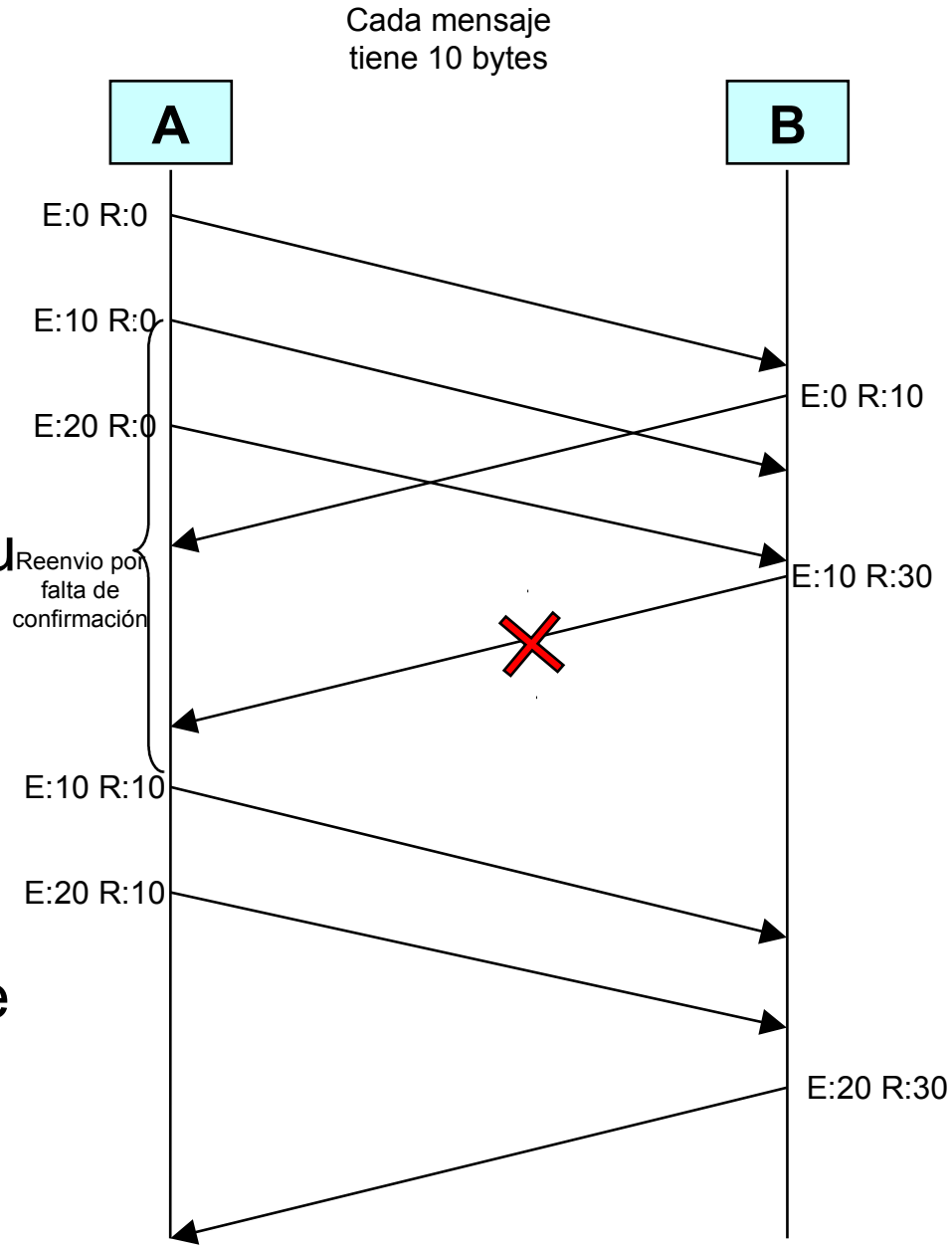

unusual event
client/receiver path
server/sender path
(Start)
CLOSED
CONNECT/SYN
(Step 1 of the 3-way-handshake)
CLOSE/-
LISTEN/-
CLOSE/-
(Step 2 of the 3-way-handshake)
SYN/SYN+ACK
LISTEN
RST/-
SEND/SYN
SYN RECEIVED
SYN/SYN+ACK
(simultaneous open)
SYN SENT
Data exchange occurs
ACK/-
ESTABLISHED
SYN+ACK/ACK
(Step 3 of the 3-way-handshake)
CLOSE/FIN
CLOSE/FIN
FIN/ACK
Active CLOSE
Passive CLOSE
FIN/ACK
FIN WAIT 1
CLOSING
CLOSE WAIT
FIN+ACK/ACK
ACK/-
CLOSE/FIN
FIN WAIT 2
TIME WAIT
LAST ACK
FIN/ACK
Timeout
(Go back to start)
CLOSED

Reenvíos TCP

- Los extremos de TCP llevan la cuenta del número de secuencia del siguiente byte que deben recibir y del siguiente byte que van a enviar
- Además, en cada paquete envían
  - El nº de secuencia del primer byte del paquete
  - El nº de secuencia del siguiente byte a recibir
- De esta forma los dos extremos de la comunicación pueden conocer qué bytes ha recibido/enviado el otro extremo y pueden reenviar información perdida

Reenvíos TCP

- TCP puede enviar varios paquetes sin confirmar (ventana)
  - El otro extremo confirma sólo el último paquete recibido con su siguiente transmisión
- A veces se envían paquetes vacíos, sólo para confirmar al otro extremo
  - Por ejemplo, cuando se lleva un segundo sin emitir nada, pero recibiendo bytes



TCP vs UDP

- TCP es un medio de transmisión asegurado
  - Las aplicaciones que usan TCP no envían ni reciben paquetes, sino bytes. TCP decide cuando enviar un paquete (las aplicaciones pueden *"opinar"*)
  - Requiere sesión, por lo que se puede *identificar* aproximadamente al emisor y receptor.
- UDP es más eficiente
  - No necesita mantener conexión, ni reordenar paquetes, ni retransmitir paquetes
  - Las aplicaciones son *"conscientes"* de que se envían paquetes, no bytes.
  - En redes con pocos errores, puede ser más adecuado
  - Interesante cuando se necesita mucho ancho de banda pero no importa perder algún paquete (voz, vídeo)

¿Cómo se programa esto?: Servidor

```
main(){
    int sd, psd;
    struct sockaddr_in name;
    char buf[1024];
    int cc;

    sd = socket (AF_INET,SOCK_STREAM,0);
    name.sin_family = AF_INET;
    name.sin_addr.s_addr = htonl(INADDR_ANY);
    name.sin_port = htons(12345);

    bind( sd, (SA *) &name, sizeof(name) );
    listen(sd,1);
    psd = accept(sd, 0, 0);
    close(sd);

    for(;;) {
        cc=recv(psd,buf,sizeof(buf, 0) ;
        if (cc == 0) exit (0);
        buf[cc] = NULL;
        printf("message received: %s\n", buf);
    }
}
```

¿Cómo se programa esto?: Cliente

```
main(..)
{
    int sd;
    struct sockaddr_in server;
    struct hostent *hp, *gethostbyname();

    sd = socket (AF_INET,SOCK_STREAM,0);

    server.sin_family = AF_INET;
    hp = gethostbyname(argv[1]);
    bcopy ( hp->h_addr, &(server.sin_addr.s_addr), hp->h_length);
    server.sin_port = htons(12345);

    connect(sd, (SA *) &server, sizeof(server));

    for (;;) {
        send(sd, "HI", 2 );
        sleep(2);
    }
}
```

Puertos TCP/UDP

- Cada extremo de una conexión TCP o datagrama UDP tiene un puerto asignado
- Los puertos se asignan de la siguiente forma
  - El proceso servidor activa la escucha en un puerto conocido (80 para HTTP, 25 para SMTP,…)
  - El cliente inicia una conexión a dicho proceso. El sistema le asigna un puerto no utilizado cualquiera

Conexiones abiertas: **netstat**

- El comando **netstat** controla las conexiones TCP/UDP abiertas en el sistema
  - Puertos utilizados, procesos conectados, estado de la conexión,…
- Ejemplos
  - **Netstat /?**: Ayuda
  - **Netstat -b**: Conexiones actuales y el proceso que las maneja en el ordenador
  - **Netstat –a**: Conexiones actuales y puertos a la escucha en el ordenador