

Tema 5

La Capa de Transporte

*Amelia Zafra Gómez
Dpto. Informática y Análisis Numérico*



Tema 5: La Capa de Transporte

1. Introducción
2. Elementos de los protocolos de transporte
3. Un protocolo de transporte sencillo
4. El protocolo de transporte UDP
5. El protocolo de transporte TCP
6. Aspectos de desempeño
7. Bibliografía



1. Introducción

1.1 Servicios de la capa de Transporte

1.2 Primitivas del servicio de transporte

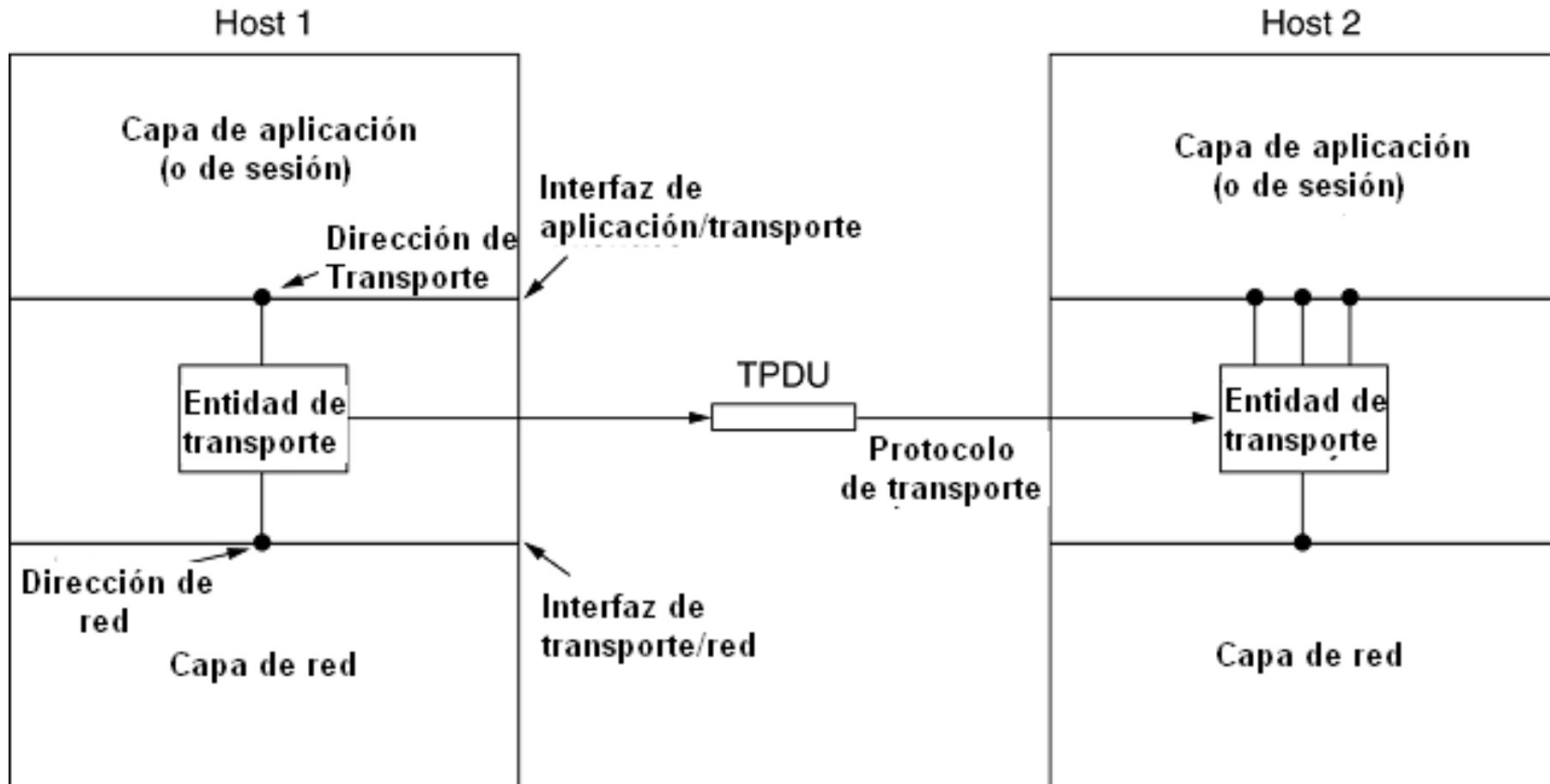
1.3 Sockets de Berkeley

- La **capa de transporte** se encarga de:
 1. Proporcionar un transporte de datos **confiable, eficiente y económico** de la máquina de origen a **destino**, independientemente de la red o redes físicas en uso.
 2. Para lograr esto utiliza los servicios proporcionados por la capa de red.

- El hardware o software de la capa de transporte que se encarga del trabajo se llama entidad de transporte y puede estar:
 1. En el kernel o núcleo del sistema operativo.
 2. En un proceso de usuario independiente.
 3. En un paquete de librería que forma parte de las aplicaciones de red o en la tarjeta de red.
- Hay dos tipos de servicios en la capa de transporte: orientado a la conexión y no orientado a la conexión

La capa de Transporte

1.1 Servicios de la capa de Transporte

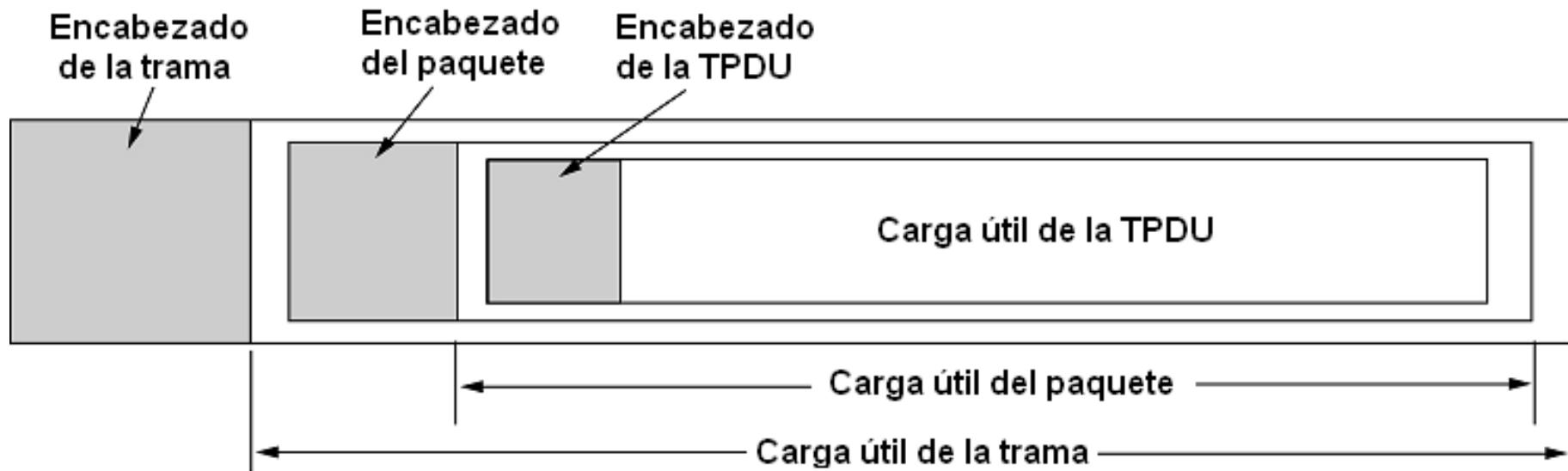


Las capas de red, transporte y aplicación

- El código de transporte se ejecuta por completo en las máquinas de los usuarios, pero la capa de red se ejecuta, normalmente, en los enrutadores.
- Si la capa de red ofrece un servicio no confiable o los enrutadores se caen de vez en cuando, los usuarios no tienen control sobre la capa de red.
- La única posibilidad es poner por encima de la capa de red, otra capa que mejore la calidad del servicio.
- Gracias a la capa de transporte es posible escribir aplicaciones usando un conjunto estándar de primitivas y que éstos programas funcionen en una amplia variedad de redes.

- Si la capa de red fuese perfecta seguramente no haría falta la capa de transporte.
- Se hace una distinción entre las capas:
 - Las capas inferiores a las de transporte pueden verse como **proveedoras del servicio de transporte**
 - Las capas por encima de la de transporte pueden verse como **usuarios del servicio de transporte**
- La capa de transporte constituye el límite principal entre el proveedor y el usuario del servicio confiable de transmisión de datos.

- Para acceder al servicio de transporte, la capa de transporte debe proporcionar algunas operaciones a sus usuarios (normalmente, los programas de aplicación), o sea una interfaz del servicio de transporte.
- Se diferencian dos protocolos de transporte
 - Orientado a conexión, en el que se garantiza la entrega de los datos al destinatario, sin errores, pérdidas ni datos duplicados.
 - No orientado a conexión, un servicio menos fiable en el que los mensajes se envían sin pedir confirmación, de forma independiente unos de otros.
- El servicio de transporte es usado por la mayoría de los usuarios para construir sus aplicaciones, por ello debe ser fácil de entender.



Anidamiento de las TPDUs, los paquetes y las tramas

TPDU: Unidad de Datos del Protocolo de Transporte

- Ejemplo de una interfaz de transporte sencilla, que muestra la esencia de lo que debe hacer una interfaz de transporte orientada a la conexión: permite que los programas de aplicación establezcan, usen y liberen conexiones, lo cual es suficiente para muchas aplicaciones.

Primitiva	Paquete enviado	Significado
LISTEN	(ninguno)	Se bloquea hasta que algún proceso intenta la conexión
CONNECT	CONNECTION REQ.	Intenta activamente establecer una conexión
SEND	DATA	Envía información
RECEIVE	(NINGUNO)	Se bloquea hasta que llega un paquete DATA
DISCONNECT	DISCONNECTION REQ.	Este lado quiere liberar la conexión

Primitivas de un servicio de transporte sencillo

Primitiva	Significado
SOCKET	Crea un nuevo punto terminal de comunicación
BIND	Adjunta una dirección local a un socket
LISTEN	Anuncia la disposición a aceptar conexiones
ACCEPT	Bloquea el invocador hasta la llegada de un intento de conexión
CONNECT	Intenta establecer activamente una conexión
SEND, WRITE	Envía datos a través de la conexión
RECEIVE, READ	Recibe datos de la conexión
CLOSE	Libera la conexión

Primitivas de socket para TCP.

Ejemplo de programación de Sockets

Programa Cliente

```
/* This page contains a client program that can request a file from the server program
 * on the next page. The server responds by sending the whole file.
 */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define SERVER_PORT 12345           /* arbitrary, but client & server must agree */
#define BUF_SIZE 4096               /* block transfer size */

int main(int argc, char **argv)
{
    int c, s, bytes;
    char buf[BUF_SIZE];           /* buffer for incoming file */
    struct hostent *h;             /* info about server */
    struct sockaddr_in channel;   /* holds IP address */

    if (argc != 3) fatal("Usage: client server-name file-name");
    h = gethostbyname(argv[1]);    /* look up host's IP address */
    if (!h) fatal("gethostbyname failed");

    s = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (s < 0) fatal("socket");
    memset(&channel, 0, sizeof(channel));
    channel.sin_family= AF_INET;
    memcpy(&channel.sin_addr.s_addr, h->h_addr, h->h_length);
    channel.sin_port= htons(SERVER_PORT);

    c = connect(s, (struct sockaddr *) &channel, sizeof(channel));
    if (c < 0) fatal("connect failed");

    /* Connection is now established. Send file name including 0 byte at end. */
    write(s, argv[2], strlen(argv[2])+1);

    /* Go get the file and write it to standard output. */
    while (1) {
        bytes = read(s, buf, BUF_SIZE);      /* read from socket */
        if (bytes <= 0) exit(0);            /* check for end of file */
        write(1, buf, bytes);              /* write to standard output */
    }
}

fatal(char *string)
{
    printf("%s\n", string);
    exit(1);
}
```




Tema 5: La Capa de Transporte

1. Introducción
2. Elementos de los protocolos de transporte
3. Un protocolo de transporte sencillo
4. El protocolo de transporte UDP
5. El protocolo de transporte TCP
6. Aspectos de desempeño
7. Bibliografía



2. Elementos de los Protocolos de Transporte

2.1 Introducción

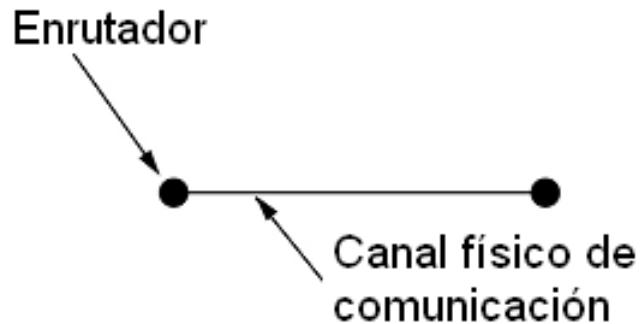
2.2 Direccionamiento

2.3 Establecimiento de la Conexión

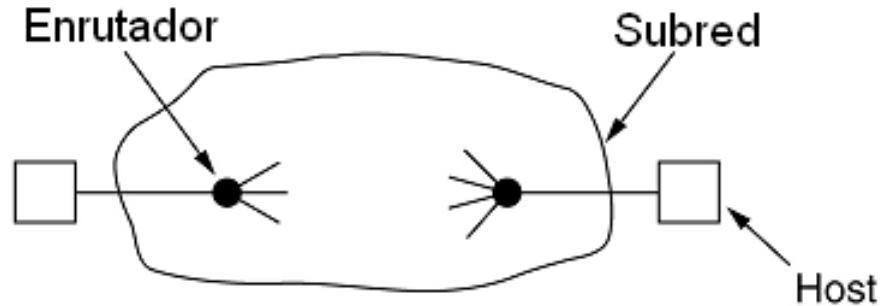
2.4 Liberación de la Conexión

2.5 Control de Flujo y Almacenamiento en
búfer

- El servicio de transporte se implementa mediante un protocolo de transporte entre las 2 entidades de transporte. Se encargan del control de errores y flujo pero en un entorno distinto a la capa de enlace.
- Diferencias entre los dos entornos:
 - Necesita especificar el enrutador con quien quiere comunicarse.
 - El establecimiento de la conexión es menos sencillo.
 - Problema del almacenamiento de paquetes en la subred.
 - Se requieren búferes y control de flujo en ambas capas.



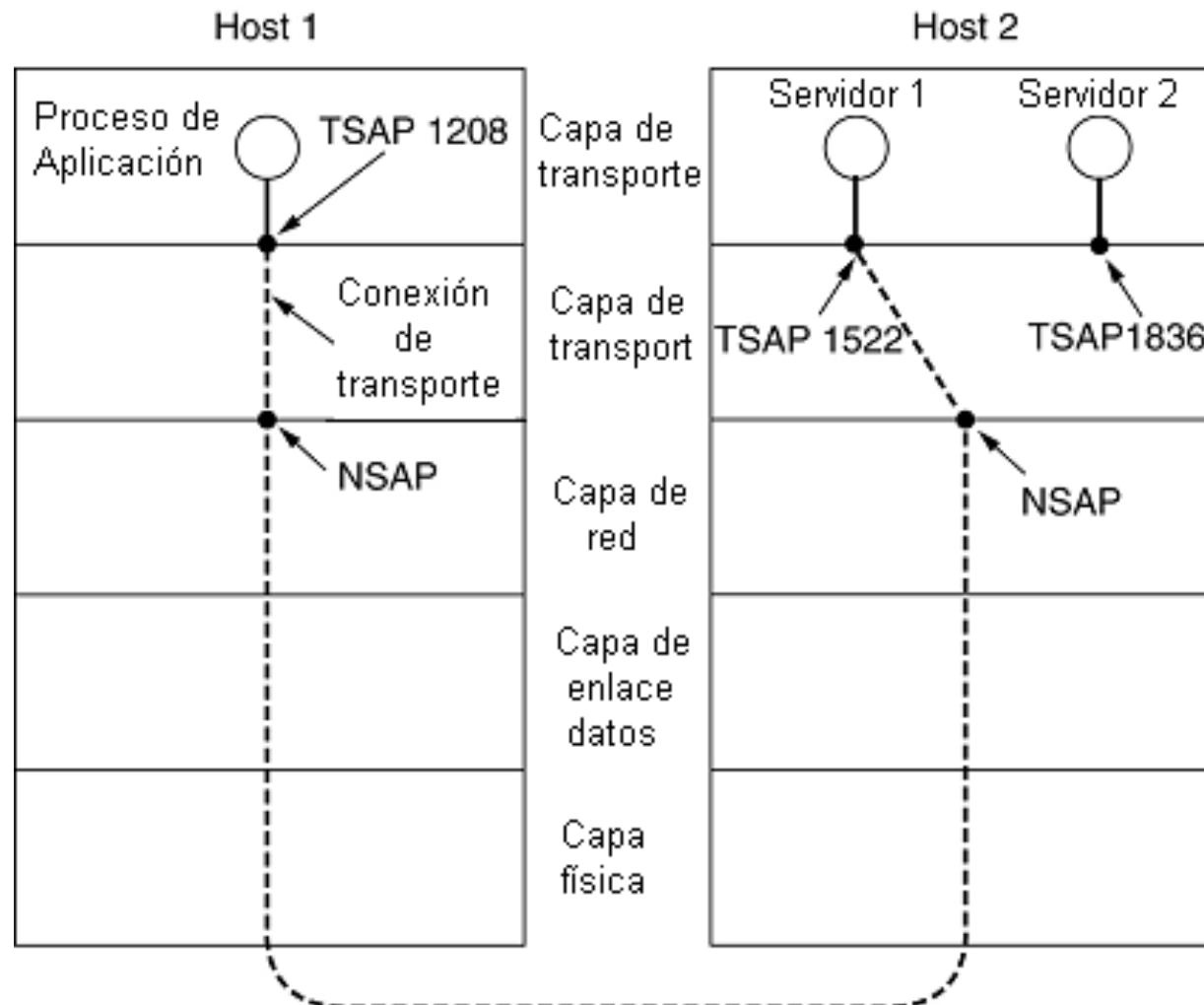
(a)



(b)

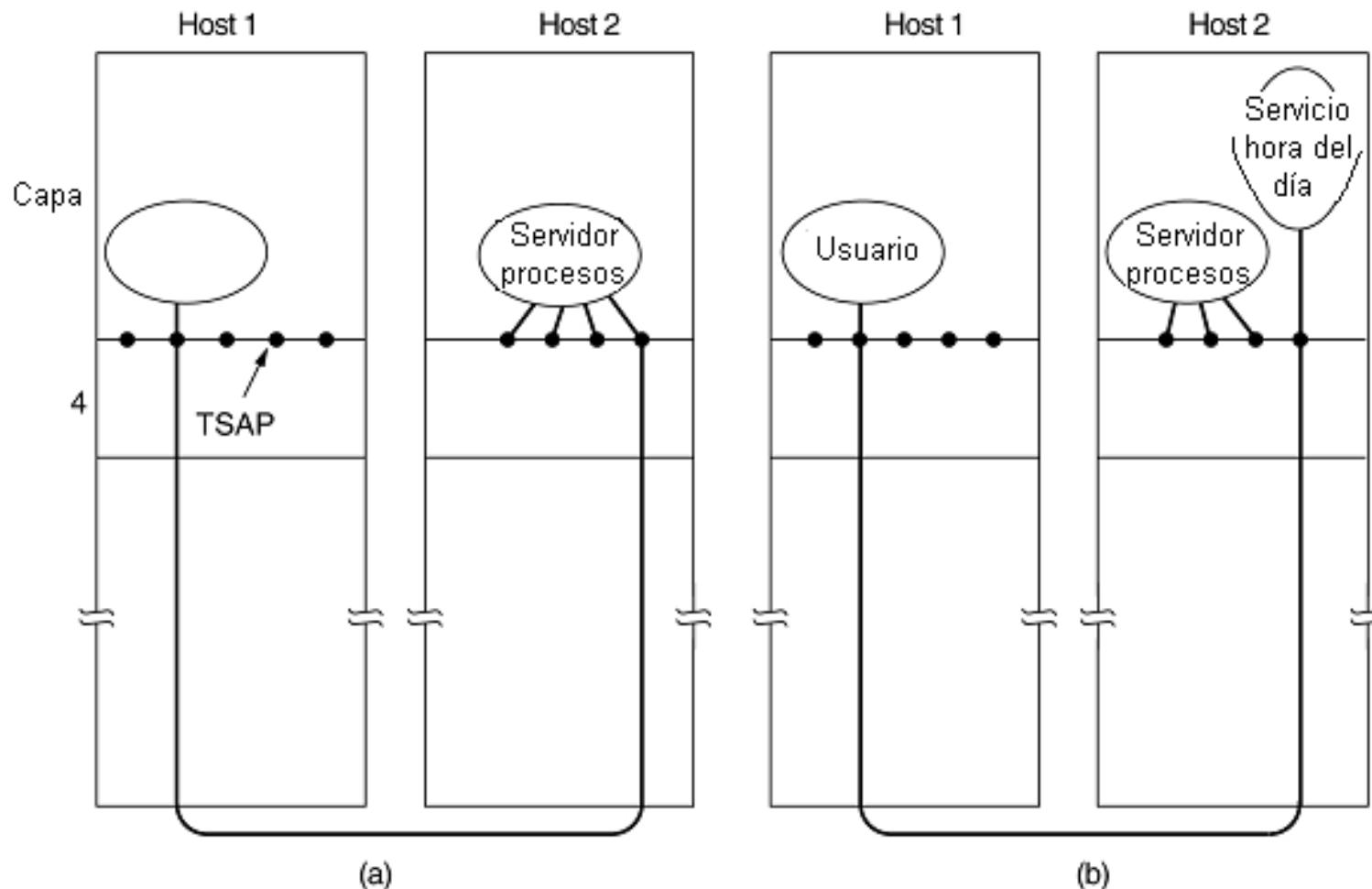
- (a) Entorno de la capa de enlace de datos.
- (b) Entorno de la capa de transporte.

- Cuando un proceso (ej. Un usuario) desea establecer una conexión con un proceso de aplicación remoto debe especificar a cuál se conectará.
- El método que se emplea es definir direcciones de transporte en la que los procesos pueden estar a la escucha de solicitudes de conexión. En Internet estos puntos terminales se llaman puertos, de forma generalizada los llamaremos **TSAP (Punto de Acceso al Servicio de Transporte)**.
- Los 1024 primeros puertos están reservados para servicios estándar a nivel mundial (www.iana.org) y los restantes hasta 65.535 para uso propio siempre y cuando otro proceso no lo esté utilizando.
- Los puntos análogos de la capa de red se llaman **NSAP**. Las direcciones IP son ejemplos de NSAP.



TSAPs, NSAPs y conexiones de transporte.

- Cuando hay muchos procesos de servidor, la mayoría de los cuales se usan pocas veces, es muy ineficiente tenerlos activados todos, escuchando en una dirección TSAP estable todo el día
 - Se emplea en estos casos **protocolo inicial de conexión**: cada máquina que desea ofrecer servicio a usuarios remotos tiene un servidor de procesos especial que actúa como proxy de los servidores de conexión.
- En el caso de servicios que existan independientemente del servidor de procesos
 - Se emplea un proceso especial: **servidor de nombres** o **servidor de directorios**: para localizar el TSAP de un servicio se establece una conexión con este servidor indicándole el nombre del servicio y el servidor de nombres devuelve la dirección TSAP. Se libera esta conexión y se inicia una nueva con el servicio deseado.



Manera en que un proceso usuario del host 1 establece una conexión con un servidor del hora del día en el host 2.

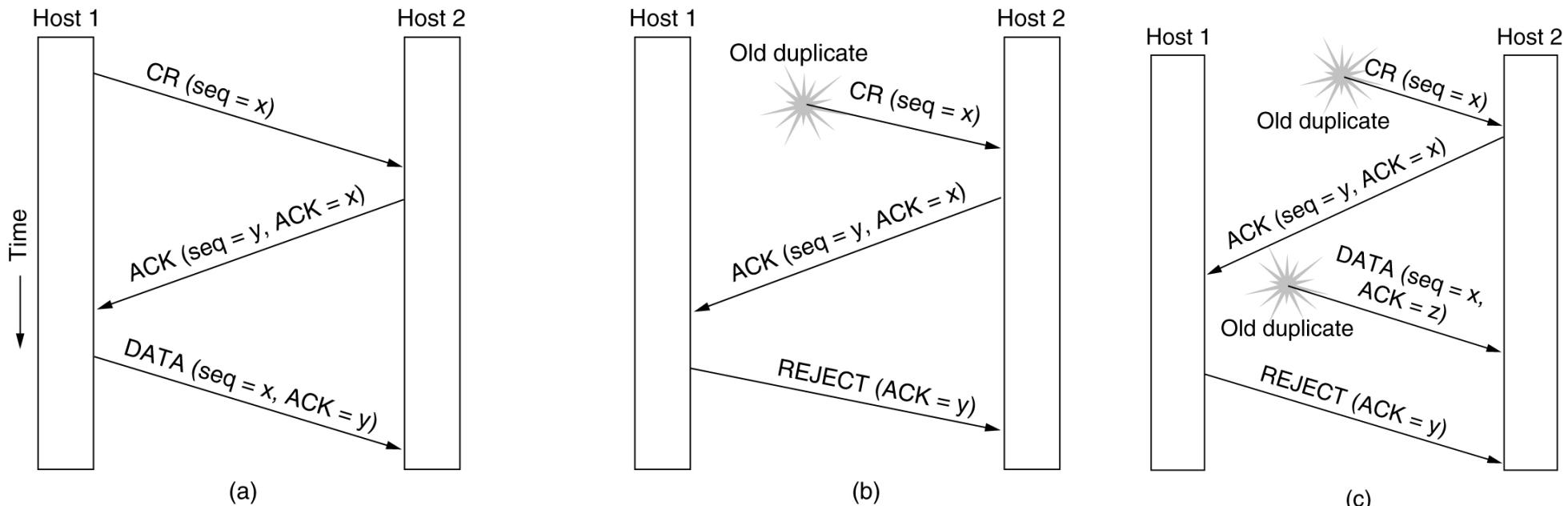
- El establecimiento de conexión a priori parece fácil sería suficiente una **TPDU CONNECTION REQUEST** al destino y esperar una respuesta **CONNECTIO ACCEPTED**. El problema es que se pierden paquetes, se retrasan y se duplican.
- En una subred congestionada en que las confirmaciones de recepción casi nunca regresan a tiempo y cada paquete expira y se retransmite 2 ó 3 veces, el peor problema son los duplicados retrasados.
- Para solucionar el problema de los duplicados retrasados tenemos
 - Usar direcciones de transporte desechables.
 - Dar a cada conexión un identificador de conexión (es decir, un número de secuencia que se incrementa con cada conexión establecida), seleccionado por la parte iniciadora, y ponerlo en cada TPDU, incluida la que solicita la conexión. Tras la liberación de una conexión cada entidad de transporte podría actualizar una tabla que liste conexiones obsoletas.

- Para solucionar el problema de los duplicados retrasados tenemos (continuación).
 - Permitir que los paquetes no vivan eternamente en la subred, diseñar un mecanismo para eliminar a los paquete viejos que aún andan vagando por ahí.

El tiempo de vida de un paquete puede restringirse a un máximo conocido usando una de las siguientes técnicas:

 1. Un diseño de subred restringido.
 2. Colocar un contador de saltos en cada paquete.
 3. Marcar el tiempo en cada paquete.
 - No sólo hay que garantizar que el paquete se elimine, sino que también se eliminan posibles retransmisiones de dicho paquete.

- Limitar el tiempo de vida de los paquetes, permite proponer un mecanismo de controlar los errores para poder establecer una conexión segura. El método se debe a Tomlinson (1975).
- Tomlinson en 1975 propuso una solución llamada **acuerdo de tres vías**:
 1. El host 1 escoge un número de secuencia X y envía al host 2 una TPDU CONNECTION REQUEST que la contiene.
 2. El host 2 responde con TPDU CONNECTION ACCEPTED confirmando la recepción de X y anunciando su propio número de secuencia inicial.
 3. El host 1 confirma la recepción del número de secuencia inicial del host 2 en la primera TPDU de datos que envía.



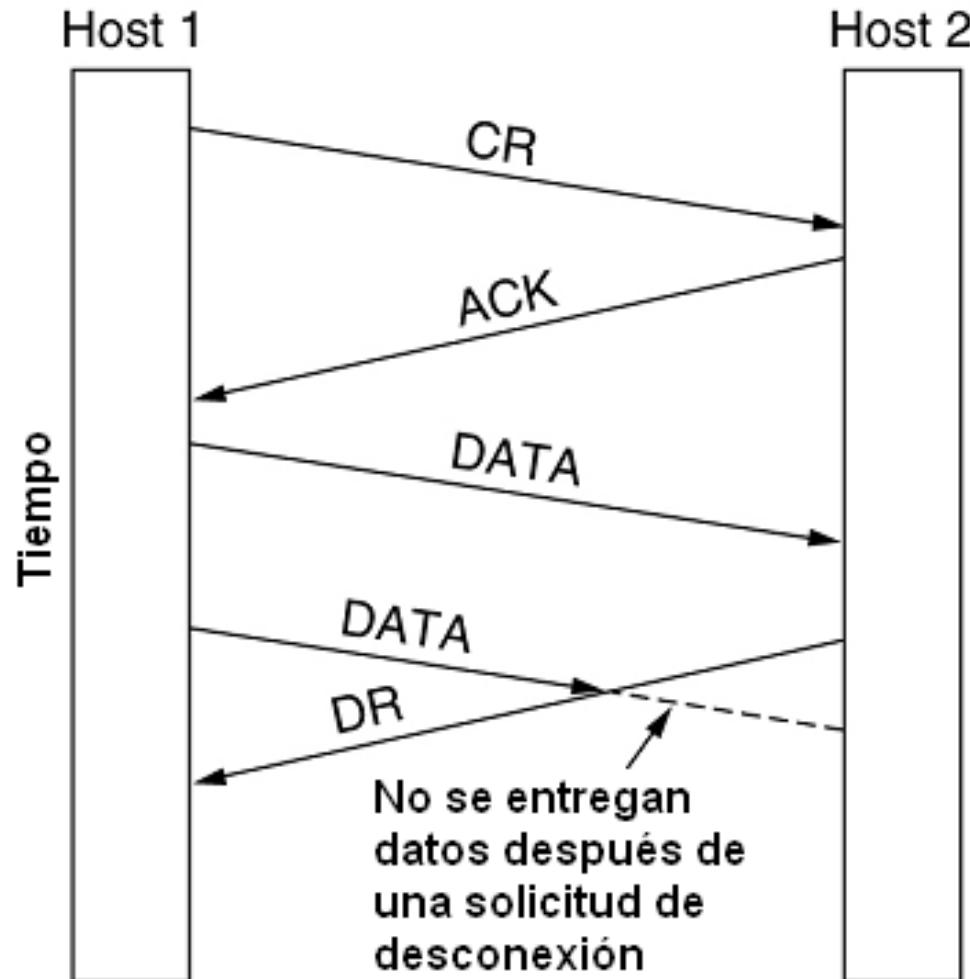
Tres escenarios para establecer una conexión usando un **acuerdo de tres vías**
CR significa CONNECTION REQUEST.

(a) Operación normal,

(b) **CR duplicada** vieja que llega al host 2 y este envía una TPDU ACK al host 1 solicitando la comprobación de que el host1 quiere conectarse, el host 1 la rechaza, el host 2 se da cuenta que era una duplicado con retardo y abandona la conexión

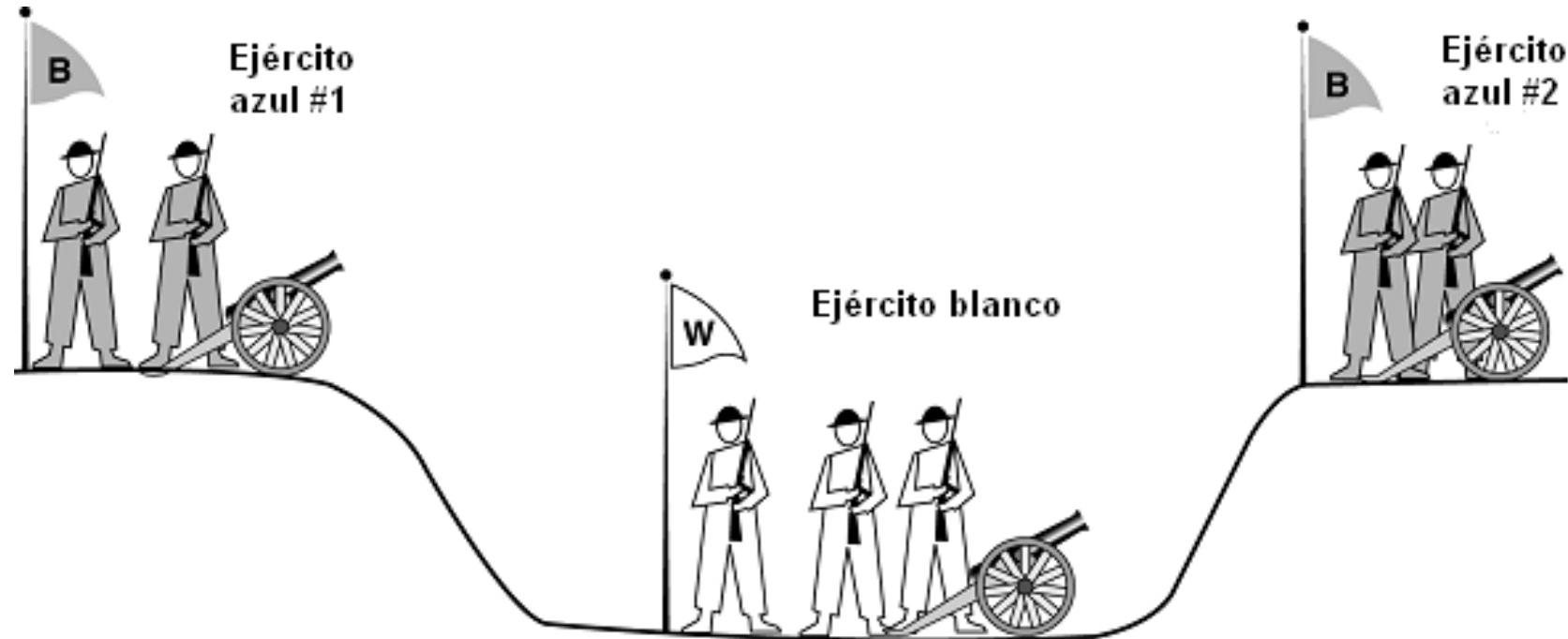
(c) **CR duplicada y ACK duplicado**, la CR vieja se trata igual que antes, el host 2 envió antes Y como número de secuencia inicial para el tráfico del host 2 al 1, con lo cual al recibir una confirmación de Z significa que es un duplicado viejo

- Una conexión puede finalizarse de dos formas:
 1. **Liberación asimétrica:** manera en que funciona el sistema telefónico, cuando una parte cuelga, se interrumpe abruptamente la conexión.
 2. **Liberación simétrica:** trata la conexión como dos conexiones unidireccionales distintas y requiere que cada uno libere por separado.
- La asimétrica es abrupta y **puede resultar en pérdida de datos**, mientras que en la **simétrica** un **host** puede continuar **recibiendo datos** aún después de haber enviado una TPDU **DISCONNECT REQUEST (DR)**.



Desconexión abrupta con pérdida de datos

- Para evitar la pérdida de datos se requiere un protocolo de liberación más refinado. Para esto se utiliza la **liberación simétrica** donde cada parte se libera independientemente de la otra.
- La liberación simétrica es ideal cuando un proceso tiene una cantidad fija de datos por enviar y sabe con certidumbre cuándo los ha enviado.
- Podríamos pensar en un protocolo en el que el host 1 diga: Ya terminé. ¿Terminaste también?. Si el host 2 responde: Ya terminé también. Adiós, la conexión puede liberarse con seguridad.

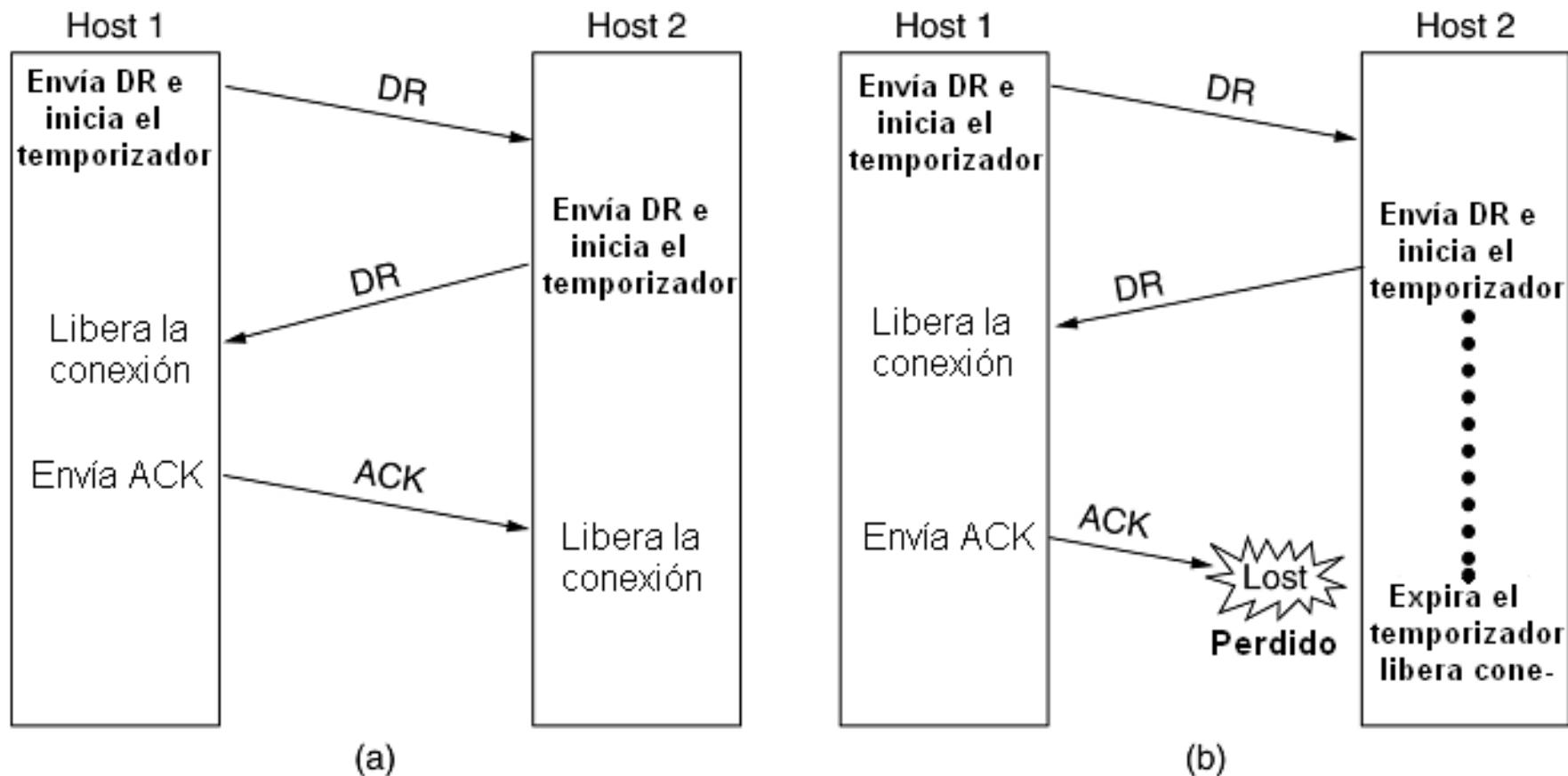


El problema de los dos ejércitos: para sincronizar ataques

- **Acuerdo de dos vías:** el 1 envía un mensaje al 2 y el 2 responde, sin estar seguro de que ha llegado al ejército 1, no ataca.
- **Acuerdo de tres vías:** el 1 envía un mensaje al 2 y el 2 responde, el 1 debe confirmar la recepción de la respuesta del 2, pero no estará seguro si llega.

La capa de Transporte

2.4 Liberación de la conexión

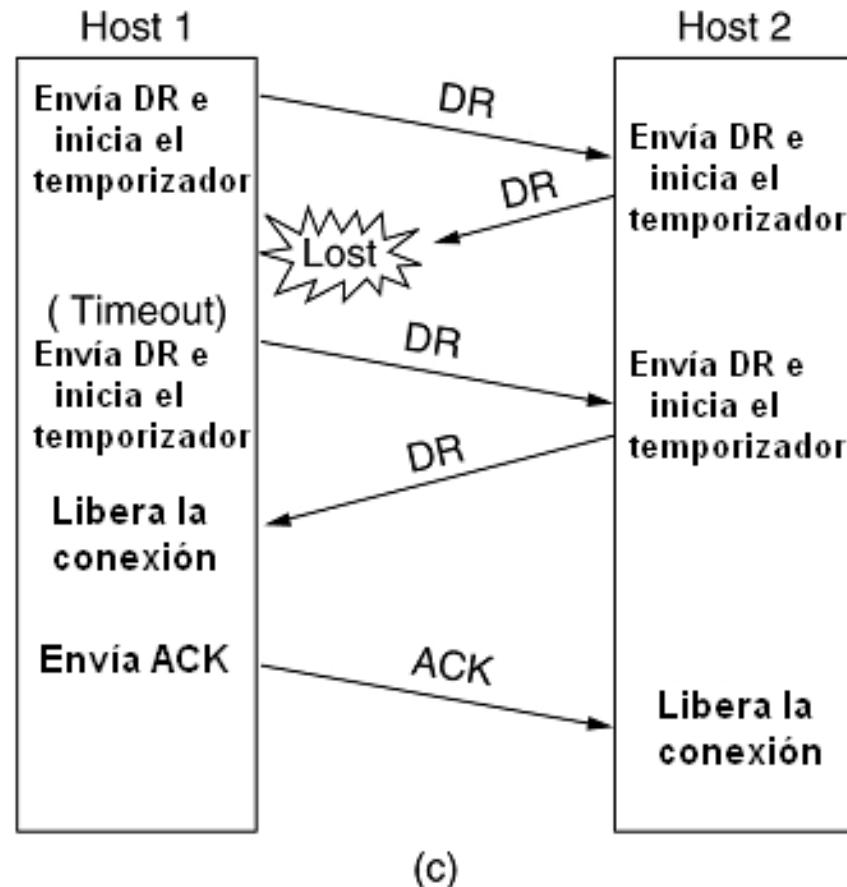


Cuatro escenarios de un protocolo para liberar una conexión. (a) Caso normal del acuerdo de tres vías (b) Se pierde el último ACK, en este caso el temporizador salva la situación. Cuando este expira la conexión se libera de todos modos.

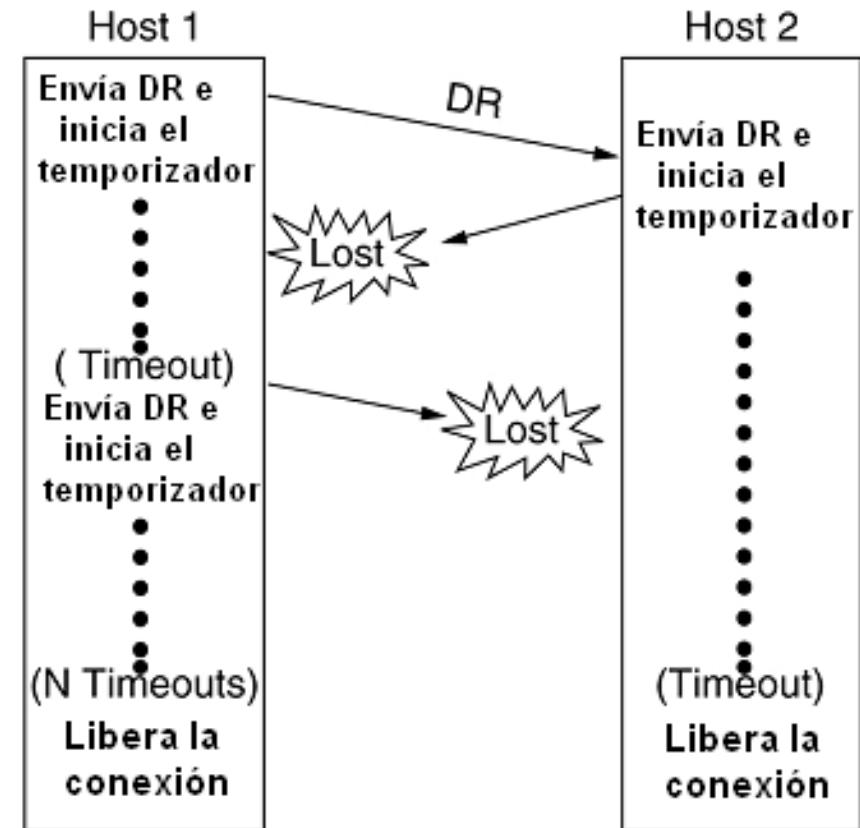
La capa de Transporte

2.4 Liberación de la conexión

Timeout = expira el temporizador



(c)



(d)

(c) Pérdida de segunda DR, el usuario que inicia la desconexión no recibirá la respuesta esperada, su temporizador expira y el proceso comienza de nuevo (d) Caso crítico: todos los intentos repetidos de retransmitir fallan. Tras N reintentos, el emisor se da por vencido y libera la conexión, lo mismo ocurre en el receptor.

- Habiendo visto cómo conectar y desconectar, veamos la manera en que se manejan las conexiones mientras están en uso.
- El método propuesto es similar al que utiliza la capa de enlace de datos, y es el protocolo de **ventana corrediza para evitar que un emisor rápido sature a uno lento, la diferencia es el entorno**, por lo general un enrutador tiene pocas líneas y un host puede tener muchas conexiones.
- Por este motivo la asignación de espacio para buffers en el nivel de transporte tiene dos características singulares que le diferencian del nivel de enlace.
 - En primer lugar el espacio de buffers es común y compartido por todas las conexiones, entrantes y salientes.
 - En segundo lugar, el tráfico que genera cada conexión normalmente es diferente. Por ello, el reparto del espacio entre las conexiones activas debe realizarse de forma dinámica de acuerdo con las necesidades; una conexión con poco tráfico debe recibir menos asignación que una con mucho tráfico.

- A medida que se abren y cierran conexiones y cambia el patrón de tráfico, el emisor y el receptor necesitan ajustar dinámicamente sus asignaciones de búfer.
- En consecuencia el protocolo de transporte debe permitir que el host emisor solicite espacio de búfer en el otro extremo. Como alternativa, el receptor sabiendo su capacidad de manejo de búferes podría indicar al emisor “te he reservado X búferes”
- Una manera general de manejar la asignación dinámica de búferes es desacoplarlos de las confirmaciones de recepción, en contraste con los protocolos de ventana corrediza de capa 2.
- Esto implica una ventana de tamaño variable. Inicialmente el emisor solicita una cierta cantidad de búferes con base en sus necesidades percibidas. El receptor otorga tanto búferes como puede. Cada vez que el emisor envía una TPDU debe disminuir su asignación, deteniéndose al llegar a cero. El receptor incorpora tanto las confirmaciones de recepción como las asignaciones de búfer al tráfico de regreso.

	<u>A</u>	<u>Message</u>	<u>B</u>	<u>Comments</u>
1	→	< solicito 8 búferes>	→	A quiere 8 búferes
2	←	<ack = 15, buf = 4>	←	B sólo otorga los mensajes 0 a 3
3	→	<seq = 0, data = m0>	→	A tiene 3 búferes libres
4	→	<seq = 1, data = m1>	→	A tiene 2 búferes libres
5	→	<seq = 2, data = m2>	•••	Mensaje perdido, pero A piensa que le queda 1
6	←	<ack = 1, buf = 3>	←	B confirma la recepción de 0 y 1, permite 2, 4
7	→	<seq = 3, data = m3>	→	A tiene un búfer libre
8	→	<seq = 4, data = m4>	→	A tiene 0 búferes libres y debe detenerse
9	→	<seq = 2, data = m2>	→	El temporizador de A expira y retransmite
10	←	<ack = 4, buf = 0>	←	Todo confirmado, pero A bloqueado aún
11	←	<ack = 4, buf = 1>	←	A puede enviar 5 ahora
12	←	<ack = 4, buf = 2>	←	B encontró un búfer nuevo
13	→	<seq = 5, data = m5>	→	A tiene un búfer libre
14	→	<seq = 6, data = m6>	→	A está bloqueado nuevamente
15	←	<ack = 6, buf = 0>	←	A aún está bloqueado
16	•••	<ack = 6, buf = 4>	←	Bloqueo irreversible potencial

Asignación dinámica de ventanas con Nº secuencia 4 bits. Las flechas muestran la dirección de la transmisión. Los puntos suspensivos (...) indican una TPDU perdida. Para evitar 16, cada host debe enviar periódicamente una TPDU de control con la confirmación de recepción y estados de búferes de cada conexión. De esta manera el estancamiento se romperá tarde o temprano



Tema 5: La Capa de Transporte

1. Introducción
2. Elementos de los protocolos de transporte
3. Un protocolo de transporte sencillo
4. El protocolo de transporte UDP
5. El protocolo de transporte TCP
6. Aspectos de desempeño
7. Bibliografía

Vemos un ejemplo de **protocolo de transporte sencillo** orientado a la conexión. Cada conexión está siempre en uno de los siguientes siete estados

- IDLE – No se ha establecido todavía una conexión.
- WAITING – Se ha ejecutado un CONNECT, y enviado CALL REQUEST.
- QUEUED – Ha llegado un CALL REQUEST; aún no hay LISTEN.
- ESTABLISHED – Se ha establecido la conexión.
- ENDING – El usuario está esperando un permiso para enviar un paquete.
- RECEIVING – Se ha hecho un RECEIVE.
- DISCONNECTING – Se ha hecho localmente un DISCONNECT

La capa de Transporte

3. Un protocolo de transporte sencillo

```
#define MAX_CONN 32                                /* max number of simultaneous connections */
#define MAX_MSG_SIZE 8192                          /* largest message in bytes */
#define MAX_PKT_SIZE 512                            /* largest packet in bytes */

#define TIMEOUT 20
#define CRED 1
#define OK 0

#define ERR_FULL -1
#define ERR_REJECT -2
#define ERR_CLOSED -3
#define LOW_ERR -3

typedef int transport_address;
typedef enum {CALL_REQ,CALL_ACC,CLEAR_REQ,CLEAR_CONF,DATA_PKT,CREDIT} pkt_type;
typedef enum {IDLE,WAITING,QUEUED,ESTABLISHED,SENDING,RECEIVING,DISCONN} cstate;

/* Global variables. */
transport_address listen_address;                /* local address being listened to */
int listen_conn;                                  /* connection identifier for listen */
unsigned char data[MAX_PKT_SIZE];                /* scratch area for packet data */

struct conn {
    transport_address local_address, remote_address;
    cstate state;                                    /* state of this connection */
    unsigned char *user_buf_addr;                   /* pointer to receive buffer */
    int byte_count;                                /* send/receive count */
    int clr_req_received;                          /* set when CLEAR_REQ packet received */
    int timer;                                     /* used to time out CALL_REQ packets */
    int credits;                                   /* number of messages that may be sent */
} conn[MAX_CONN + 1];                            /* slot 0 is not used */ ;
```

```
void sleep(void);                                /* prototypes */  
void wakeup(void);  
void to_net(int cid, int q, int m, pkt_type pt, unsigned char *p, int bytes);  
void from_net(int *cid, int *q, int *m, pkt_type *pt, unsigned char *p, int *bytes);  
  
int listen(transport_address t)  
{ /* User wants to listen for a connection. See if CALL_REQ has already arrived. */  
    int i, found = 0;  
  
    for (i = 1; i <= MAX_CONN; i++)           /* search the table for CALL_REQ */  
        if (conn[i].state == QUEUED && conn[i].local_address == t) {  
            found = i;  
            break;  
        }  
  
    if (found == 0) {  
        /* No CALL_REQ is waiting. Go to sleep until arrival or timeout. */  
        listen_address = t; sleep(); i = listen_conn ;  
    }  
    conn[i].state = ESTABLISHED;             /* connection is ESTABLISHED */  
    conn[i].timer = 0;                      /* timer is not used */
```

```
listen_conn = 0;
to_net(i, 0, 0, CALL_ACC, data, 0);
return(i);
}

int connect(transport_address l, transport_address r)
{ /* User wants to connect to a remote process; send CALL_REQ packet. */
    int i;
    struct conn *cptr;

    data[0] = r;  data[1] = l;                                /* CALL_REQ packet needs these */
    i = MAX_CONN;                                         /* search table backward */
    while (conn[i].state != IDLE && i > 1) i = i -1;
    if (conn[i].state == IDLE) {
        /* Make a table entry that CALL_REQ has been sent. */
        cptr = &conn[i];
        cptr->local_address = l; cptr->remote_address = r;
        cptr->state = WAITING; cptr->clr_req_received = 0;
        cptr->credits = 0; cptr->timer = 0;
        to_net(i, 0, 0, CALL_REQ, data, 2);
        sleep();                                     /* wait for CALL_ACC or CLEAR_REQ */
        if (cptr->state == ESTABLISHED) return(i);
        if (cptr->clr_req_received) {
            /* Other side refused call. */
            cptr->state = IDLE;                      /* back to IDLE state */
            to_net(i, 0, 0, CLEAR_CONF, data, 0);
            return(ERR_REJECT);
        }
    } else return(ERR_FULL);                                /* reject CONNECT: no table space */
}
```

```
int send(int cid, unsigned char bufptr[], int bytes)
{ /* User wants to send a message. */
    int i, count, m;
    struct conn *cptr = &conn[cid];

    /* Enter SENDING state. */
    cptr->state = SENDING;
    cptr->byte_count = 0;                      /* # bytes sent so far this message */
    if (cptr->clr_req_received == 0 && cptr->credits == 0) sleep();
    if (cptr->clr_req_received == 0) {
        /* Credit available; split message into packets if need be. */
        do {
            if (bytes - cptr->byte_count > MAX_PKT_SIZE) /* multipacket message */
                count = MAX_PKT_SIZE; m = 1; /* more packets later */
            } else {                                /* single packet message */
                count = bytes - cptr->byte_count; m = 0; /* last pkt of this message */
            }
            for (i = 0; i < count; i++) data[i] = bufptr[cptr->byte_count + i];
            to_net(cid, 0, m, DATA_PKT, data, count); /* send 1 packet */
            cptr->byte_count = cptr->byte_count + count; /* increment bytes sent so far */
        } while (cptr->byte_count < bytes);      /* loop until whole message sent */
    }
```

```
    cptr->credits --;                                /* each message uses up one credit */
    cptr->state = ESTABLISHED;
    return(OK);
} else {
    cptr->state = ESTABLISHED;
    return(ERR_CLOSED);                            /* send failed: peer wants to disconnect */
}
}

int receive(int cid, unsigned char bufptr[], int *bytes)
{ /* User is prepared to receive a message. */
    struct conn *cptr = &conn[cid];

    if (cptr->clr_req_received == 0) {
        /* Connection still established; try to receive. */
        cptr->state = RECEIVING;
        cptr->user_buf_addr = bufptr;
        cptr->byte_count = 0;
        data[0] = CRED;
        data[1] = 1;
        to_net(cid, 1, 0, CREDIT, data, 2);      /* send credit */
        sleep();                                /* block awaiting data */
        *bytes = cptr->byte_count;
    }
    cptr->state = ESTABLISHED;
    return(cptr->clr_req_received ? ERR_CLOSED : OK);
}
```

```
int disconnect(int cid)
{ /* User wants to release a connection. */
  struct conn *cptr = &conn[cid];

  if (cptr->clr_req_received) { /* other side initiated termination */
    cptr->state = IDLE;           /* connection is now released */
    to_net(cid, 0, 0, CLEAR_CONF, data, 0);
  } else { /* we initiated termination */
    cptr->state = DISCONN;        /* not released until other side agrees */
    to_net(cid, 0, 0, CLEAR_REQ, data, 0);
  }
  return(OK);
}

void packet_arrival(void)
{ /* A packet has arrived, get and process it. */
  int cid;                      /* connection on which packet arrived */
  int count, i, q, m;
  pkt_type ptype;   /* CALL_REQ, CALL_ACC, CLEAR_REQ, CLEAR_CONF, DATA_PKT, CREDIT */
  unsigned char data[MAX_PKT_SIZE]; /* data portion of the incoming packet */
  struct conn *cptr;

  from_net(&cid, &q, &m, &ptype, data, &count); /* go get it */
  cptr = &conn[cid];
```

```
switch (ptype) {
    case CALL_REQ:                      /* remote user wants to establish connection */
        cptr->local_address = data[0];  cptr->remote_address = data[1];
        if (cptr->local_address == listen_address) {
            listen_conn = cid;  cptr->state = ESTABLISHED;  wakeup();
        } else {
            cptr->state = QUEUED;  cptr->timer = TIMEOUT;
        }
        cptr->clr_req_received = 0;  cptr->credits = 0;
        break;

    case CALL_ACC:                      /* remote user has accepted our CALL_REQ */
        cptr->state = ESTABLISHED;
        wakeup();
        break;

    case CLEAR_REQ:                    /* remote user wants to disconnect or reject call */
        cptr->clr_req_received = 1;
        if (cptr->state == DISCONN) cptr->state = IDLE; /* clear collision */
        if (cptr->state == WAITING || cptr->state == RECEIVING || cptr->state == SENDING) wakeup();
        break;

    case CLEAR_CONF:                  /* remote user agrees to disconnect */
        cptr->state = IDLE;
        break;

    case CREDIT:                      /* remote user is waiting for data */
        cptr->credits += data[1];
        if (cptr->state == SENDING) wakeup();
        break;

    case DATA_PKT:                   /* remote user has sent data */
        for (i = 0; i < count; i++) cptr->user_buf_addr[cptr->byte_count + i] = data[i];
        cptr->byte_count += count;
        if (m == 0 ) wakeup();
    }

}
```

```
}
```

```
void clock(void)
```

```
{ /* The clock has ticked, check for timeouts of queued connect requests. */
```

```
    int i;
```

```
    struct conn *cptr;
```

```
    for (i = 1; i <= MAX_CONN; i++) {
```

```
        cptr = &conn[i];
```

```
        if (cptr->timer > 0) { /* timer was running */
```

```
            cptr->timer--;
```

```
            if (cptr->timer == 0) { /* timer has now expired */
```

```
                cptr->state = IDLE;
```

```
                to_net(i, 0, 0, CLEAR_REQ, data, 0);
```

```
            }
```

```
        }
```

```
    }
```

```
}
```

- Para evitar el hecho de tener que proporcionar y administrar búferes en la entidad de transporte, se usa aquí un mecanismo de control de flujo, diferente de la ventana corrediza tradicional.
- Cuando un usuario llama a RECEIVE, se envía un **mensaje de crédito** especial a la entidad de transporte de la máquina emisora.
- Al llamar al SEND, la entidad de transporte revisa si ha recibido el mensaje de crédito para esa conexión especificada. De ser así, el mensaje se envía (en múltiples paquetes si es necesario) y se disminuye el crédito, en caso contrario la entidad de transporte se pone a dormir en espera de un crédito.
- Este mecanismo garantiza que no se enviará nunca un mensaje a menos que el otro lado ya haya hecho un RECEIVE. Por tanto, siempre que llegue un mensaje habrá un búfer disponible en el cual puede ponerse.



Tema 5: La Capa de Transporte

1. Introducción
2. Elementos de los protocolos de transporte
3. Un protocolo de transporte sencillo
4. El protocolo de transporte UDP
5. El protocolo de transporte TCP
6. Aspectos de desempeño
7. Bibliografía



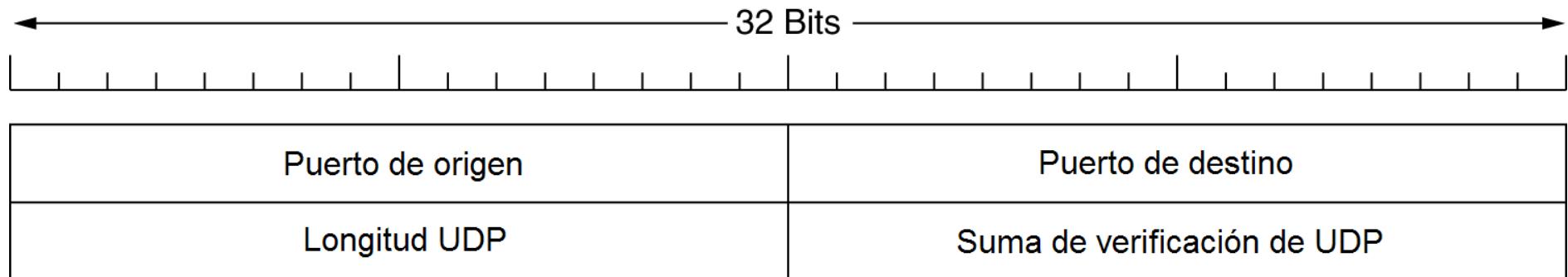
4. *El protocolo de Transporte UDP*

4.1 Introducción a UDP

4.2 Llamada a procedimiento remoto

4.3 El protocolo de transporte en tiempo real

- El conjunto de protocolos de Internet soporta un protocolo no orientado a la conexión, **UDP (Protocolo de Datagramas de Usuario)**.
- Este protocolo proporciona una forma para que las aplicaciones envíen datagramas IP encapsulados sin tener que establecer una conexión (RFC 768).
- Su uso es aconsejable en situaciones cliente-servidor, con solicitudes cortas al servidor donde se espera una respuesta corta.
- UDP transmite **segmentos** que consisten un encabezado de 8 bytes seguido por la carga útil.

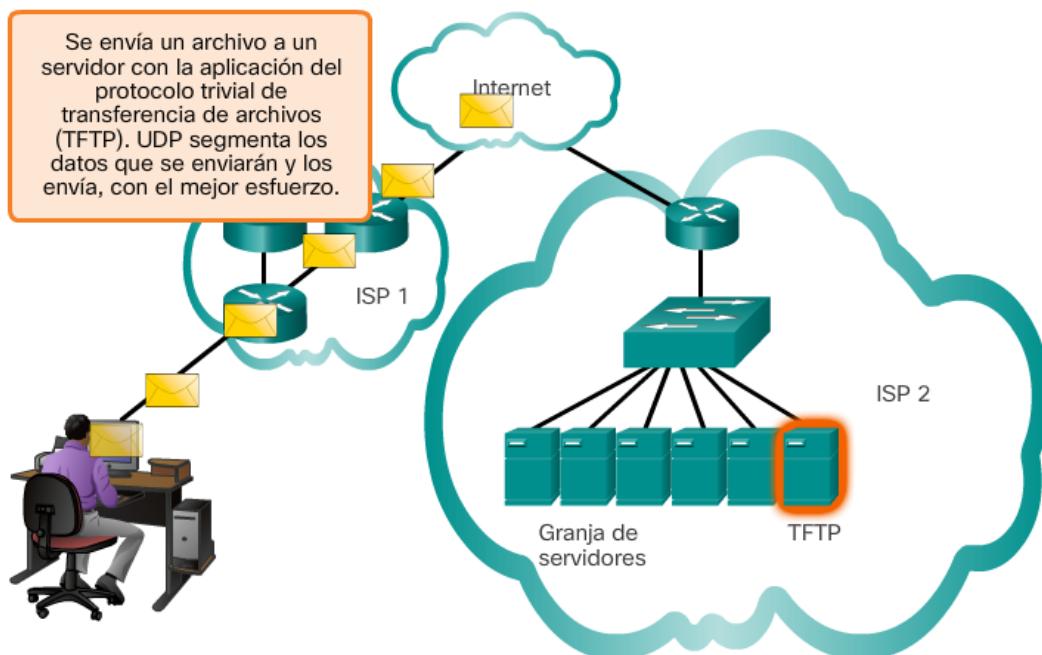


- **Puerto de origen y de destino:** especifica el puerto de la aplicación que genera el mensaje y de la aplicación a la que va dirigido el mensaje, respectivamente.
- **Longitud UDP:** indica la longitud del mensaje, incluyendo los campos de cabecera.
- **Suma de verificación:** es opcional (en IPv4) y obligatorio en IPv6(ya que en este caso se ha suprimido la comprobación a nivel de red).

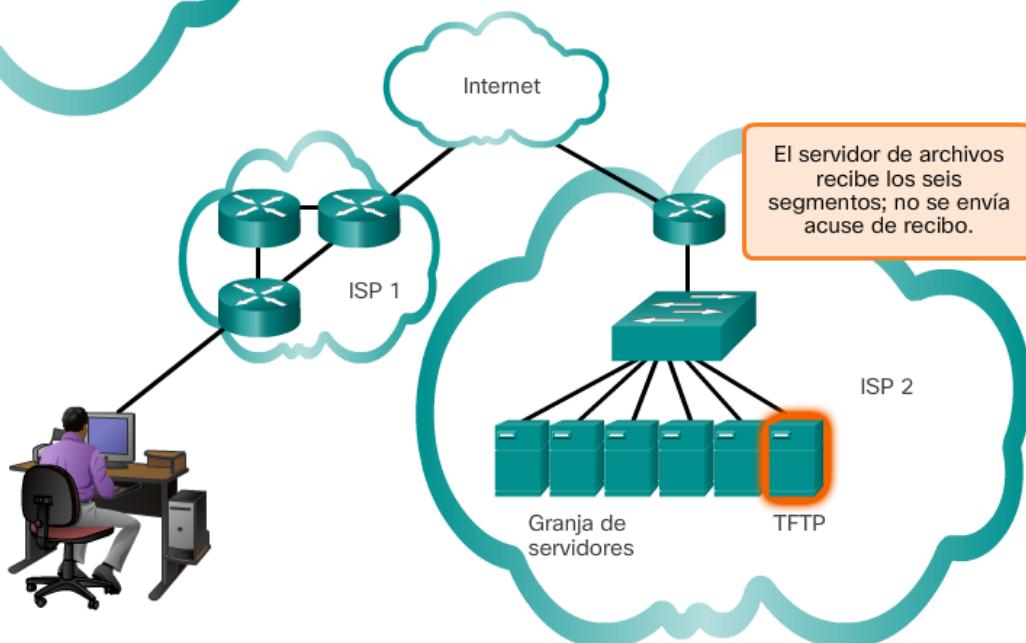
La capa de Transporte

4.1 Introducción

- Algunas aplicaciones no requieren confiabilidad. La confiabilidad genera sobrecarga adicional y posibles demoras en la transmisión.
- Agregar sobrecarga para garantizar la confiabilidad para algunas aplicaciones podría reducir la utilidad de la aplicación e incluso ser perjudicial.

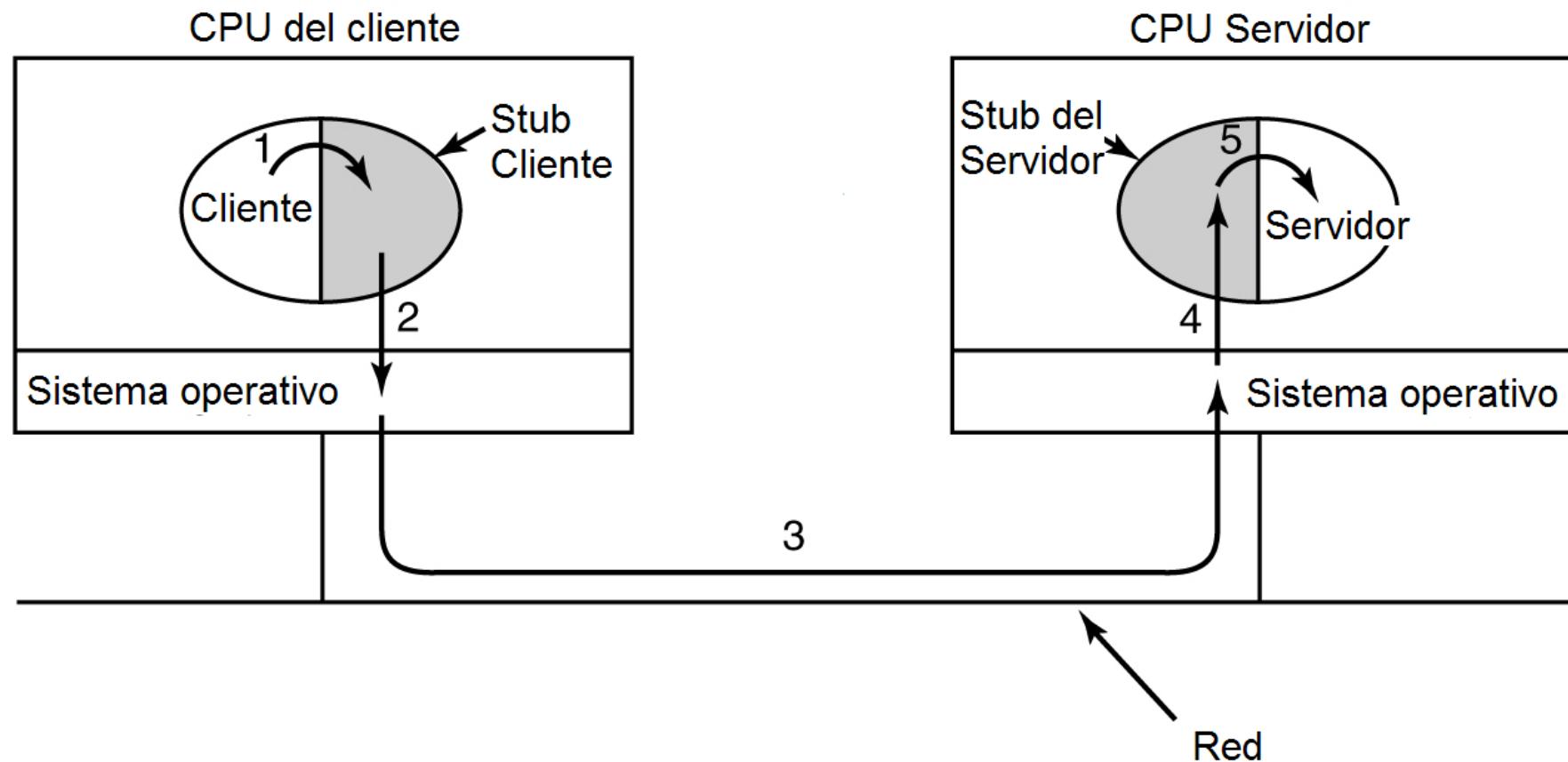


- Si no se requiere confiabilidad, UDP es un mejor protocolo de transporte.
- UDP proporciona las funciones básicas para distribuir segmentos de datos entre las aplicaciones adecuadas, con muy poca sobrecarga y revisión de datos.



- Enviar un mensaje a un host remoto y obtener una respuesta es muy parecido a hacer una llamada a función en un lenguaje de programación, de forma que se ha tratado que las interacciones de solicitud-respuesta en redes se asignen en forma de llamadas a funciones. Ej:
`obt_dirección_IP(nombre_host)` que envía un paquete UDP a un servidor DNS y espera respuesta
- En **1984 Birrell y Nelson** sugirieron que se permitiera que los programas invocaran procedimientos localizados en host remotos.
- Cuando un proceso en la máquina 1 llama a uno en la máquina 2, el proceso invocador de la primera se suspende y la ejecución del procedimiento se lleva a cabo en la máquina 2. El paso de mensajes es transparente para el programador. Esta técnica se llama **RPC (Llamada a Procedimiento Remoto)**
- Al procedimiento invocador se le llama **cliente** y al invocado **servidor**.

- Para llamar a un procedimiento remoto, el programa cliente debe enlazarse con un pequeño procedimiento de biblioteca, llamado **stub del cliente**, que representa al procedimiento servidor en el espacio de direcciones del cliente.
- De forma similar, el servidor se enlaza con una llamada a procedimiento denominada **stub del servidor**.
- La llamada a procedimientos parece local
 - **Paso 1:** el cliente llama al stub del cliente, esta es una llamada a procedimiento local.
 - **Paso 2:** El stub del cliente empaca los parámetros (marshaling) en un mensaje y realiza una llamada de sistema para enviarlo.
 - **Paso 3:** El kernel envía el mensaje del cliente al servidor.
 - **Paso 4:** El kernel pasa el paquete entrante al stub del servidor.
 - **Paso 5:** El stub del servidor desempaqueta y llama al procedimiento servidor con los mismos parámetros.

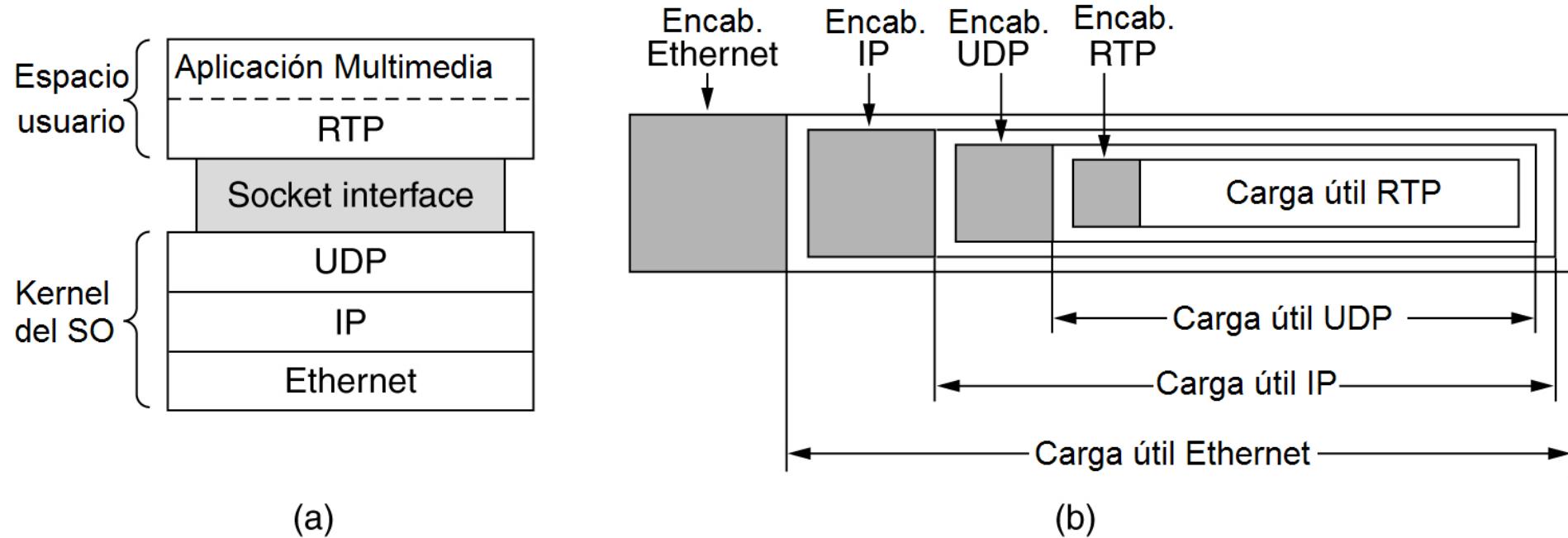


Pasos para realizar una llamada a procedimiento remoto. UDP se utiliza comúnmente en RPC cliente-servidor

Desventajas:

- No se pueden pasar apuntadores, pues el cliente y servidor están en diferentes espacios de direcciones.
- Si el procedimiento utiliza variables globales, el código fallará porque las variables globales ya no se pueden compartir.
- Ejemplos de RPC: NFS (sistema fichero UNIX) y NIS (sistema de contraseñas)

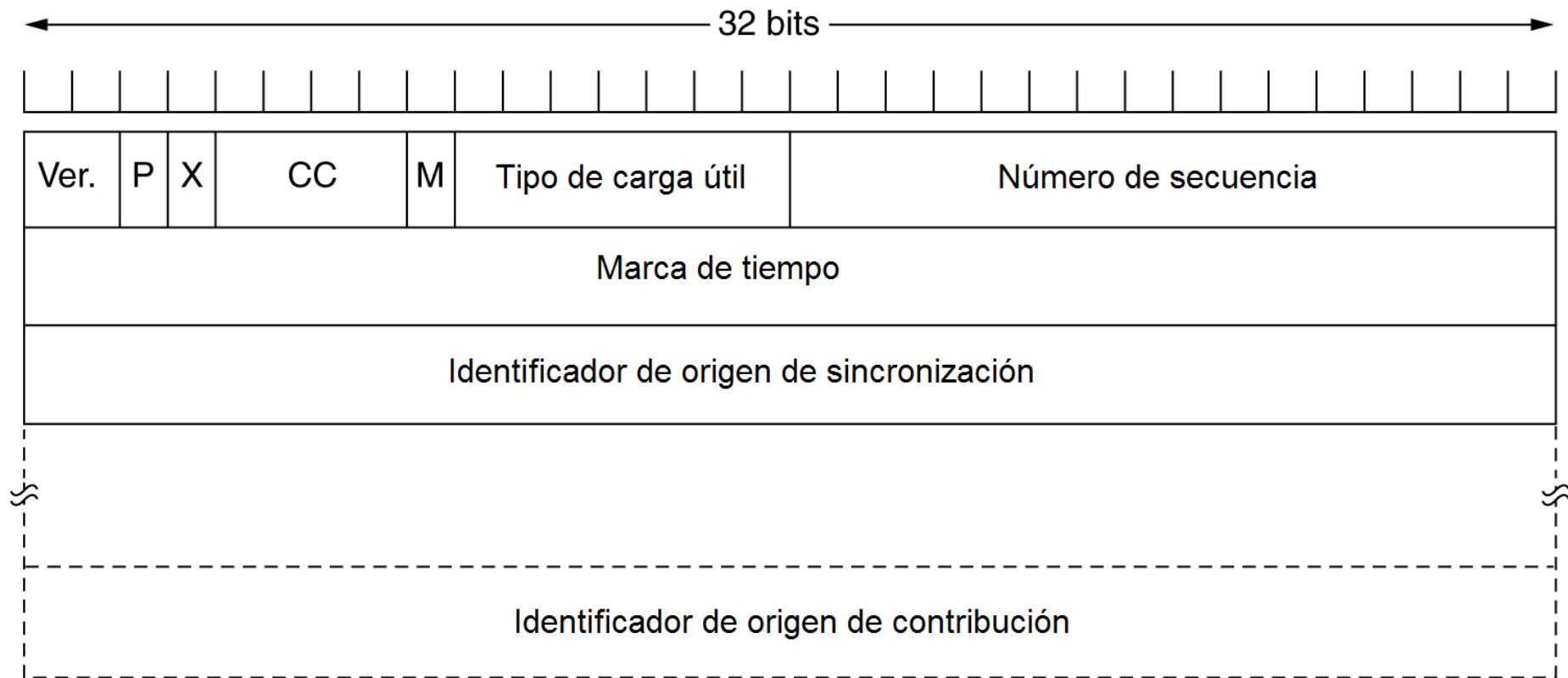
- Otra aplicación de UDP, son las **aplicaciones multimedia en tiempo real**. En particular:
 - radio en Internet
 - telefonía en Internet,
 - Música bajo demanda
 - Videoconferencias
 - Video bajo demanda, etc
- Con el tiempo se trató de unificar el el protocolo de transporte de estas aplicaciones y surgió **RTP (Protocolo de Transporte de Tiempo Real)**. Se describe en RFC 1889 y se utiliza ampliamente.
- La posición de RTP en la pila de protocolos es algo extraña -> está en el espacio de usuario y se ejecuta sobre **UDP**.
- La aplicación multimedia consiste en múltiples flujos de audio, video, texto, entre otros que se entregan al proceso RTP del espacio de usuario.



(a) Posición de RTP en la pila de protocolos. (b) Anidamiento de paquetes

- La biblioteca RTP multiplexa los flujos y los codifica en paquetes RTP, que después coloca en un socket (en el kernel del sistema operativo), los paquetes UDP se generan e incrustan en paquetes IP. Si la computadora está en Ethernet, los paquetes IP se colocan a continuación en tramas Ethernet para su transmisión.
- RTP es un protocolo genérico, independiente de la aplicación que proporciona características de transporte, se puede decir que es un **protocolo de transporte implementado en la capa de aplicación**

- La función básica de RTP es multiplexar varios flujos de datos (audio, video) en tiempo real en un sólo flujo de paquetes UDP. El flujo se puede enviar por unidifusión o multidifusión a uno o varios destinos.
- Como es UDP los paquetes **no se tratan de manera especial** a menos que se especifique.
- Los paquetes se enumeran **correlativamente** para que el destino pueda determinar si falta algún paquete. Si falta alguno se puede aproximar el valor faltante por interpolación.
- Por tanto, RTP **no tiene control de flujo, control de errores, confirmaciones de recepción ni ningún mecanismo para solicitar retransmisiones**.
- Cada carga útil de RTP puede tener varias muestras y cada una se podría codificar de la forma en la que la aplicación necesite.
- Las aplicaciones en tiempo real necesitan la **marcación de tiempo (timestamping)** para asociar una marca de tiempo a la primera muestra de cada paquete, aquí lo importante es evitar las fluctuaciones, por lo que el destino utiliza almacenamiento en búfer y reproduce cada muestra un número exacto de mseg después del inicio del flujo, además permite sincronizar los distintos flujos.



El encabezado RTP (3 palabras de 32 bits y algunas extensiones)

Versión: versión de RTP que es la 2

Tipo de carga útil: indica qué tipo de algoritmo de codificación se ha utilizado, PCM, GSM, MP3, etc

Número de secuencia: para identificar cada paquete enviado

Identificador de origen de sincronización: indica a cuál flujo pertenece el paquete, para multiplexar y demultiplexar



Tema 5: La Capa de Transporte

1. Introducción
2. Elementos de los protocolos de transporte
3. Un protocolo de transporte sencillo
4. El protocolo de transporte UDP
5. El protocolo de transporte TCP
6. Aspectos de desempeño
7. Bibliografía



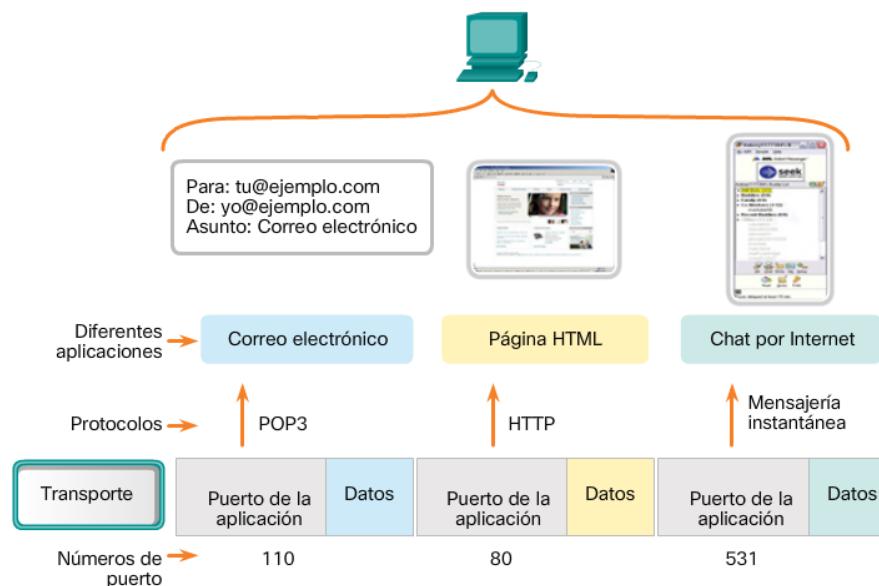
5. *El protocolo de Transporte* **TCP**

- 5.1 Introducción a TCP
- 5.2 El modelo de servicio de TCP
- 5.3 El protocolo TCP
- 5.4 Encabezado del segmento TCP
- 5.5 Establecimiento de la conexión TCP
- 5.6 Liberación de la conexión TCP
- 5.7 Control de congestión en TCP

- **UDP** es un protocolo simple y tiene algunos usos específicos, como interacciones cliente-servidor y multimedia, pero la mayoría de las aplicaciones de Internet necesitan una entrega en secuencia confiable.
- **TCP (Protocolo de Control de Transmisión)** definido en los RFC 793, 1122, 1323, se diseñó específicamente para proporcionar un flujo de bytes confiable de extremo a extremo a través de una interred no confiable.
- TCP tiene un diseño que le permite adaptarse de manera dinámica a las propiedades de la interred.
- Una entidad TCP acepta flujos de datos de usuario de procesos locales, los divide en fragmentos que no excedan los 64 KB (aunque en la práctica son 1460 bytes) de datos que se ajustan en una sola trama Ethernet con los encabezados IP y TCP.

- Cuando los datagramas que contienen datos TCP llegan a una máquina, se pasan a la entidad TCP, la cual reconstruye los flujos de bytes originales.
- La capa IP no proporciona ninguna garantía de que los datagramas se entregarán de manera apropiada, por lo que le corresponde a TCP terminar los temporizadores y retransmitir los datagramas conforme sea necesario.
- Los datagramas que llegan podrían hacerlo en orden incorrecto, también corresponde a TCP reensamblarlos en mensajes en la secuencia correcta.
- En resumen, TCP debe proporcionar la confiabilidad que la mayoría de los usuarios desean y que IP no proporciona.

- El servicio TCP se obtiene al hacer que tanto el servidor como el cliente creen puntos terminales llamados **sockets**, cada socket contiene **un número (dirección)** que consiste en la dirección IP del host y un número de 16 bits, que es local a ese host, llamado puerto.
- **Un puerto** es el nombre TCP para un TSAP. Para obtener el servicio TCP se debe establecer de manera explícita una conexión entre un socket del emisor y otro en el receptor.
- Los números de puerto menores de 1024 se llaman **puertos bien conocidos** y se reservan para servicios estándar (se han asignado alrededor de 300).



La Autoridad de Números Asignados de Internet (IANA) es el organismo de estandarización que se encarga de asignar diversos estándares de direccionamiento, incluidos los números de puerto.

Números de puerto

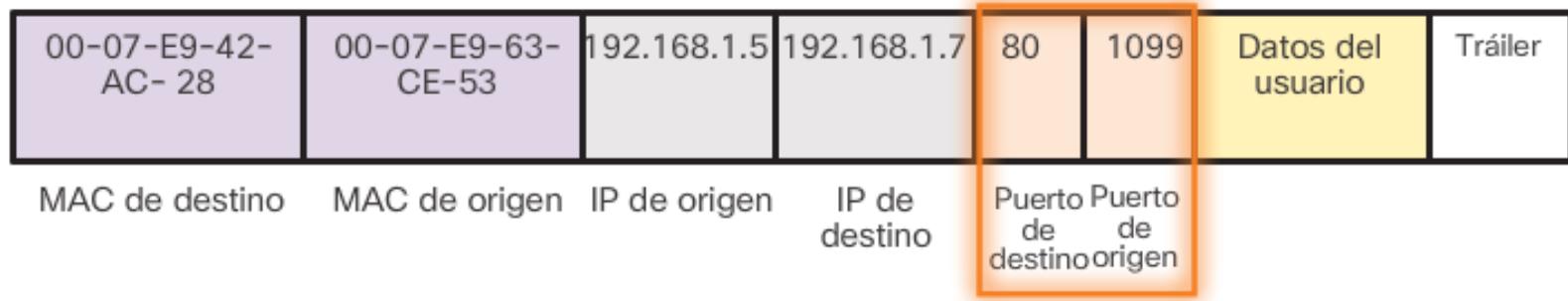
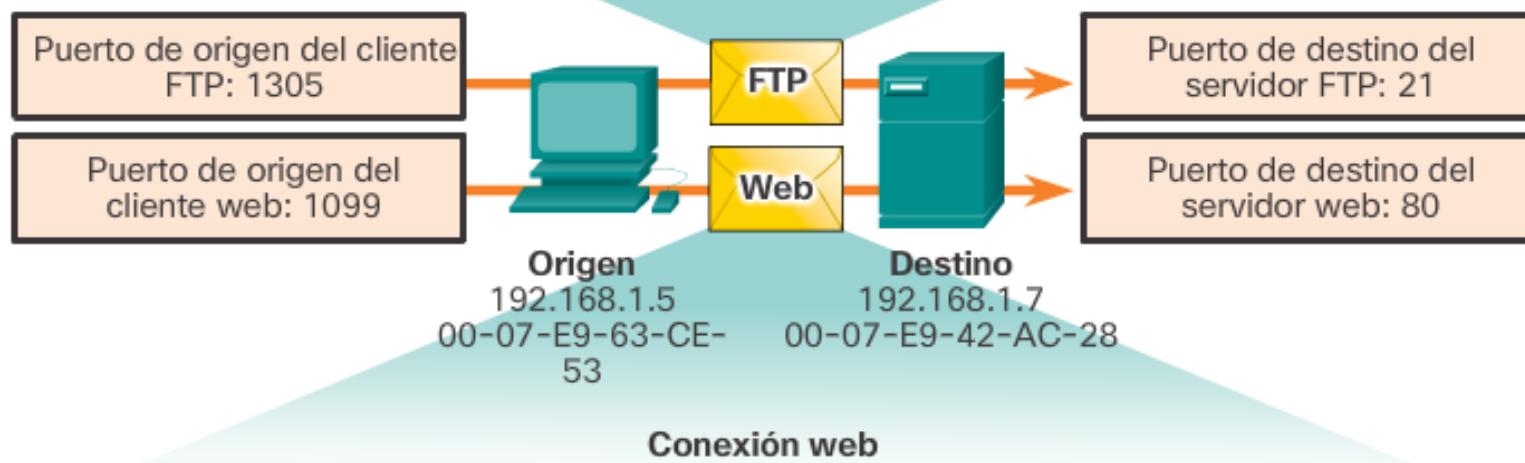
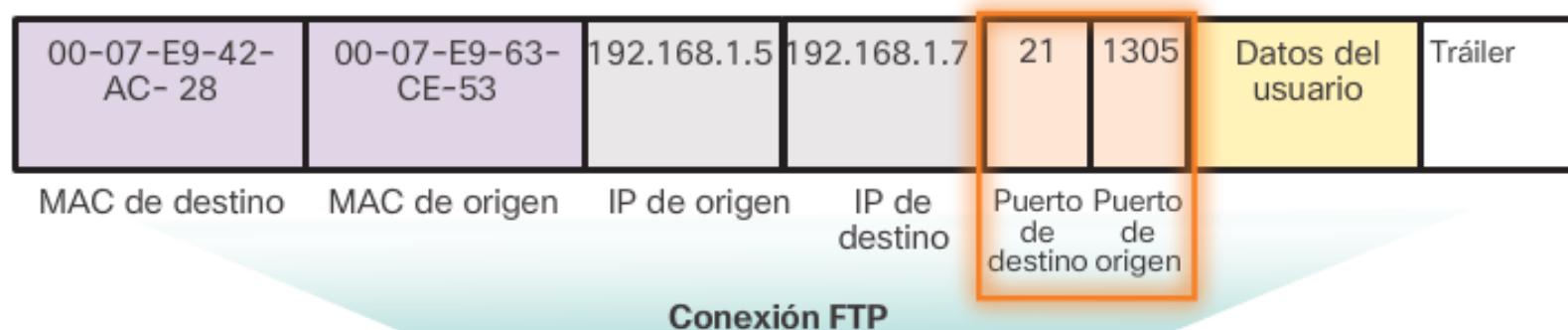
Rango de números de puerto	Grupo de puertos
Entre 0 y 1023	Puertos bien conocidos
de 1024 a 49151	Puertos registrados
de 49152 a 65535	Puertos privados y/o dinámicos

Números de puerto conocidos

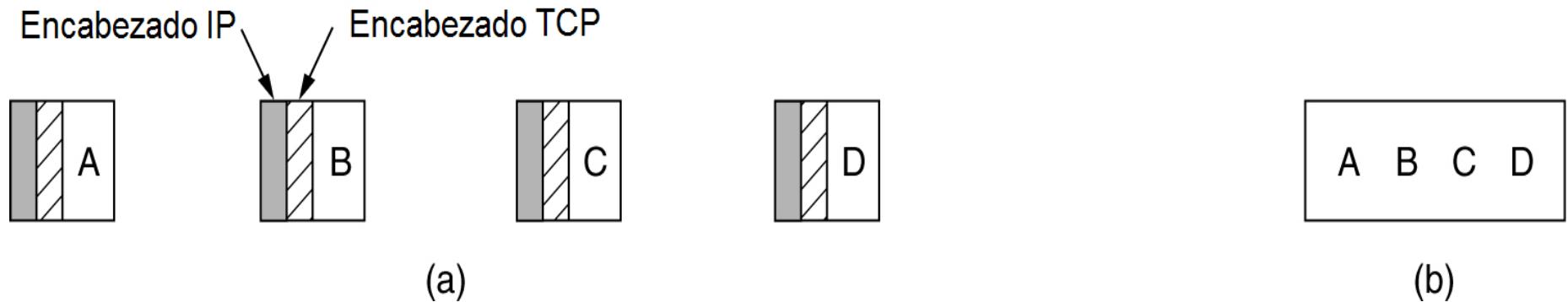
Número de puerto	Protocolo	Aplicación	Acrónimo
20	TCP	Protocolo de transferencia de archivos (datos)	FTP
21	TCP	Protocolo de transferencia de archivos (control)	FTP
22	TCP	Shell Seguro	SSH
23	TCP	Telnet	–
25	TCP	Protocolo simple de transferencia de correo (Simple Mail Transfer Protocol)	SMTP
53	UDP, TCP	Servicio de nombres de dominios	DNS
67	UDP	Protocolo de configuración dinámica de host (servidor)	DHCP
68	UDP	Protocolo de configuración dinámica de host (cliente)	DHCP
69	UDP	Protocolo de transferencia de archivos trivial	TFTP
80	TCP	Protocolo de transferencia de hipertexto	HTTP
110	TCP	Protocolo de oficina de correos versión 3 (Post Office Protocol version 3)	POP3
143	TCP	Protocolo de acceso a mensajes de Internet (Internet Message Access Protocol)	IMAP
161	UDP	Simple Network Management Protocol	SNMP
443	TCP	Protocolo seguro de transferencia de hipertexto	HTTPS

La capa de Transporte

5.2 El modelo de servicio TCP



- Para utilizar los puertos bien conocidos en un servidor es necesario conectar a la aplicación el demonio del puerto específico.
- Una opción podría ser que el demonio del servicio FTP, por ejemplo, se conecte a sí mismo al puerto 21 en tiempo de arranque, el de telnet 23, etc.
- Hacer lo anterior podría llenar la memoria con demonios que están inactivos la mayor parte del tiempo.
- En su lugar, lo que se hace generalmente es que un sólo demonio, llamado **inetd (demonio de Internet)** en UNIX, se conecte a sí mismo a múltiples puertos y esperar las conexiones entrantes, creando un nuevo proceso y ejecutando el demonio específico que lo atenderá.
- TCP es dúplex total y no soporta la difusión ni multidifusión



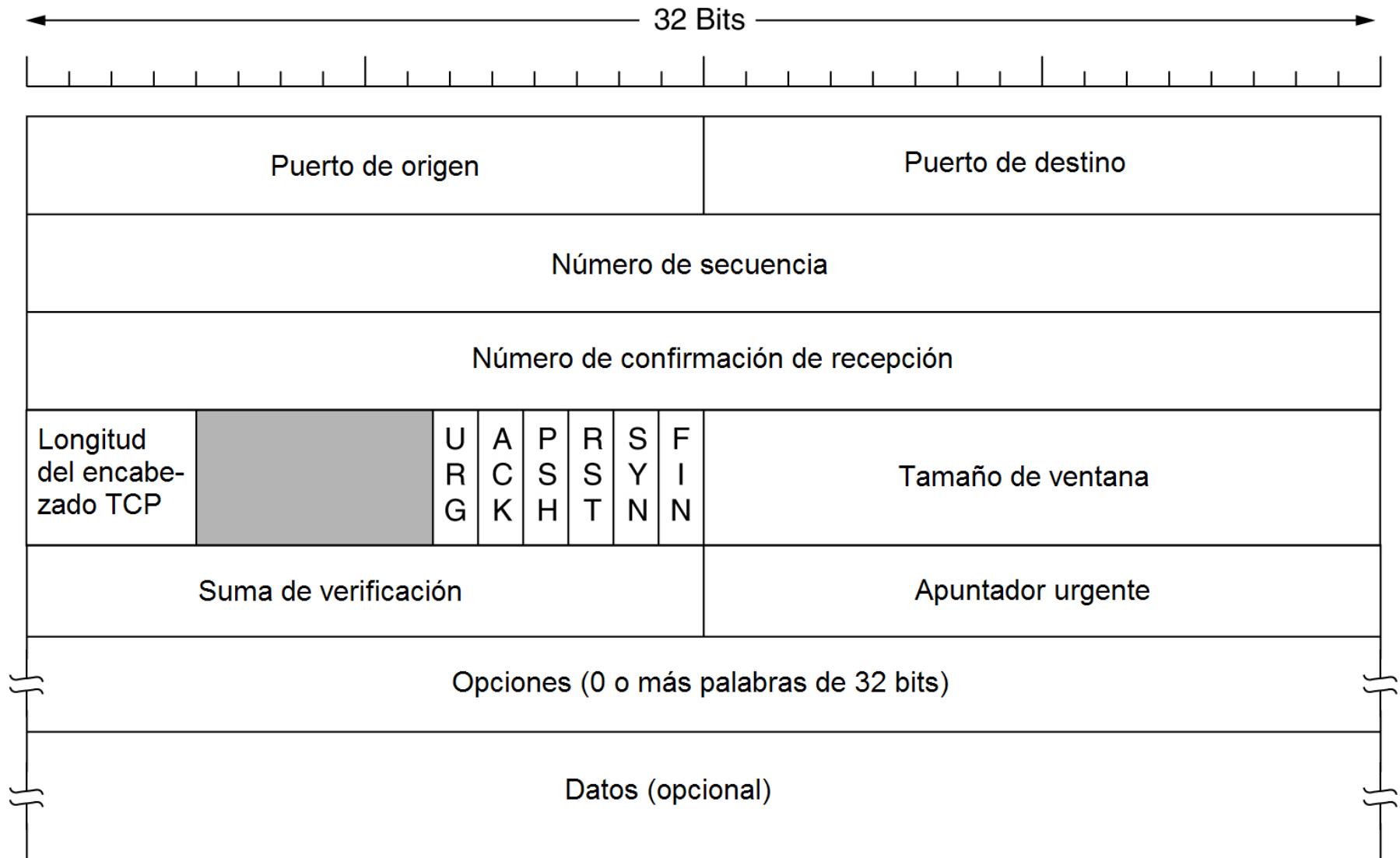
- (a) Cuatro segmentos de 512 bytes enviados como datagramas independientes
- (b) Los 2048 bytes de datos se entregan a la aplicación en una sola llamada a READ

- La entidad TCP emisora y la receptora intercambian datos en forma de segmentos.
- Un **segmento** consiste en un **encabezado TCP fijo de 20 bytes** (más una parte opcional), seguido de cero o más bytes de datos.
- El software de **TCP decide el tamaño de los segmentos**, puede acumular datos de varias escrituras para formar un segmento o dividir los datos de una escritura en varios segmentos.
- Hay dos límites de segmentos
 - Cada **segmento** incluido el encabezado IP, **debe caber en la carga útil de 65.515 bytes del IP**.
 - Cada red tiene una **unidad máxima de transferencia (MTU)** y cada segmento debe caber en la MTU.
 - En la práctica la **MTU es generalmente de 1500 bytes** (el tamaño de la carga útil de Ethernet).

- El protocolo básico usado por las entidades TCP es el protocolo de ventana corrediza.
- Cuando un transmisor envía un segmento, también inicia un temporizador. Cuando llega el segmento al destino la entidad TCP receptora devuelve un segmento (con datos, si existen) que contiene un número de confirmación de recepción igual al siguiente número de secuencia que espera recibir.
- Si el temporizador del emisor expira antes de la recepción de la confirmación, el emisor envía de nuevo el segmento.

La capa de Transporte

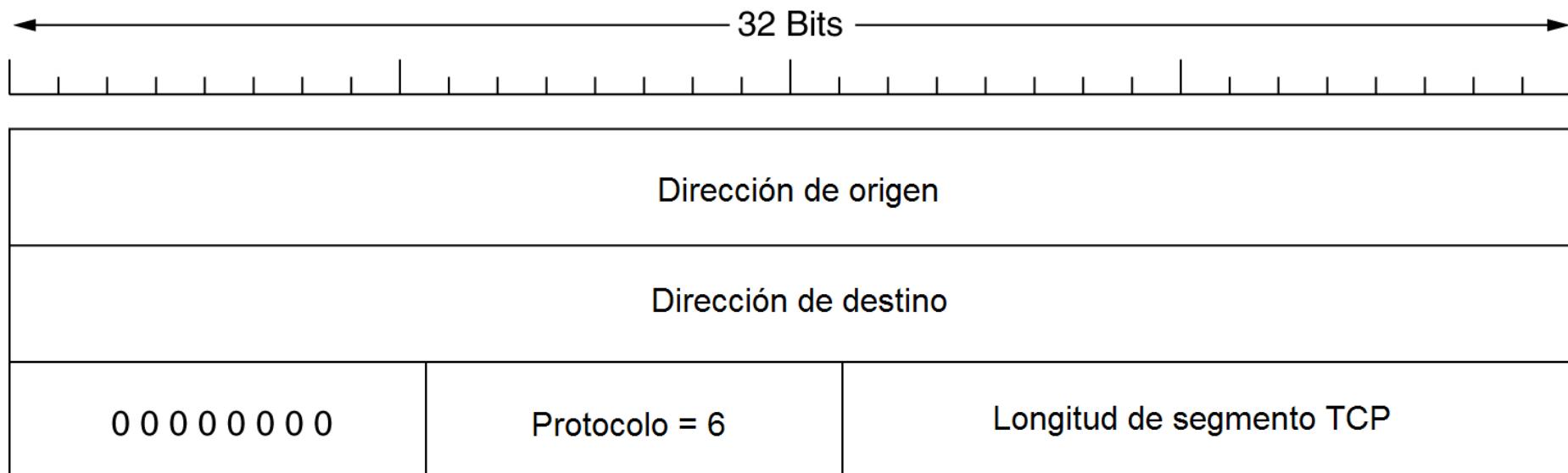
5.4 Encabezado del segmento TCP



Encabezado TCP.

- Puerto de origen y puerto destino: identifican los puntos terminales de conexión. La dirección de un puerto más la dirección IP de su host forman un punto terminal único de 48 bits.
- Número de secuencia y número de confirmación de recepción: el de confirmación especifica el siguiente byte esperado, no el último byte correctamente recibido.
- Longitud del encabezado TCP: indica la cantidad de palabras de 32 bits contenidas en el encabezado.
- Seis indicadores de 1 bit:
 - **URG**: indica si está activo el apuntador urgente, que sirve al receptor para saber a partir de qué nº secuencia los datos son urgentes.
 - **ACK**: si es 0 el segmento no contiene confirmación de recepción.
 - **PSH**: activo indica que se deben transmitir de inmediato y no almacenar en búfer.
 - **SYN**: se usa para solicitar una conexión,
 - SYN = 1, ACK = 0 (CONNECTION REQUEST)
 - SYN = 1, ACK = 1 (CONNECTION ACCEPTED)
 - **FIN**: se usa para liberar una conexión.

- El control de flujo en TCP se maneja usando una ventana corrediza de tamaño variable.
- El campo tamaño de ventana indica la cantidad de bytes que pueden enviarse comenzando por el byte cuya recepción se ha confirmado.
- Suma de verificación: es una suma de verificación del encabezado, los datos y el pseudoencabezado conceptual.
- El pseudoencabezado contiene las direcciones IP de 32 bits de las máquinas de origen y destino , el número de protocolo TCP =6 y la longitud del segmento TCP, incluido el encabezado).
- La inclusión del pseudoencabezado en el cálculo de la suma de verificación ayuda a detectar paquetes mal entregados, pero hacerlo viola la jerarquía de protocolos, puesto que las direcciones IP pertenecen a la capa IP, no TCP.

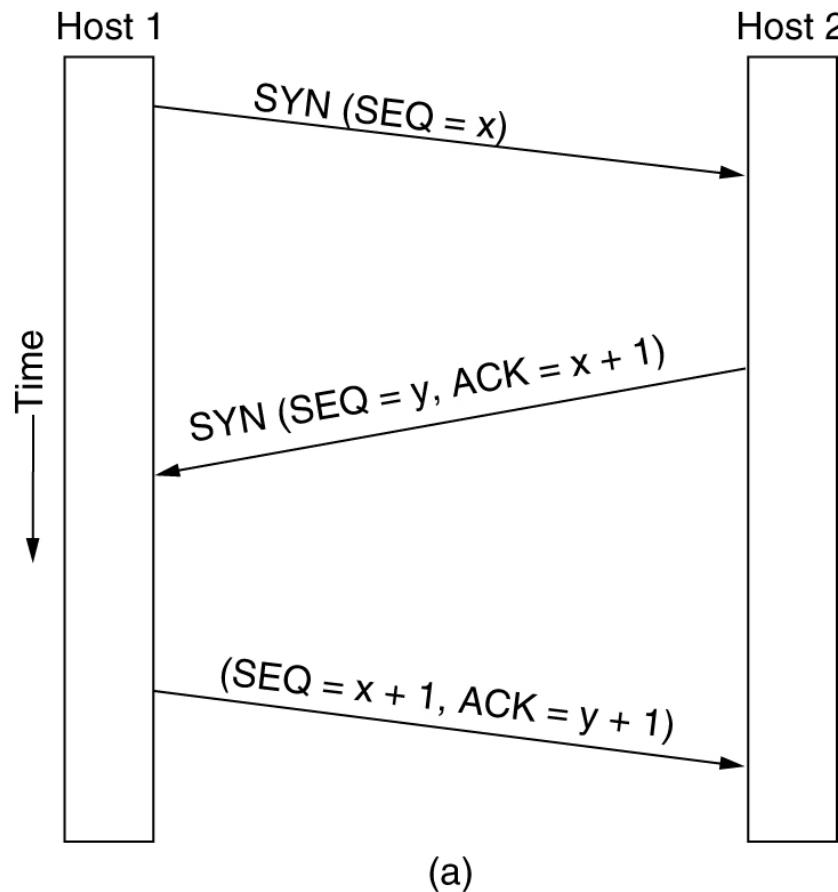


Pseudoencabezado incluido en la suma de verificación del TCP

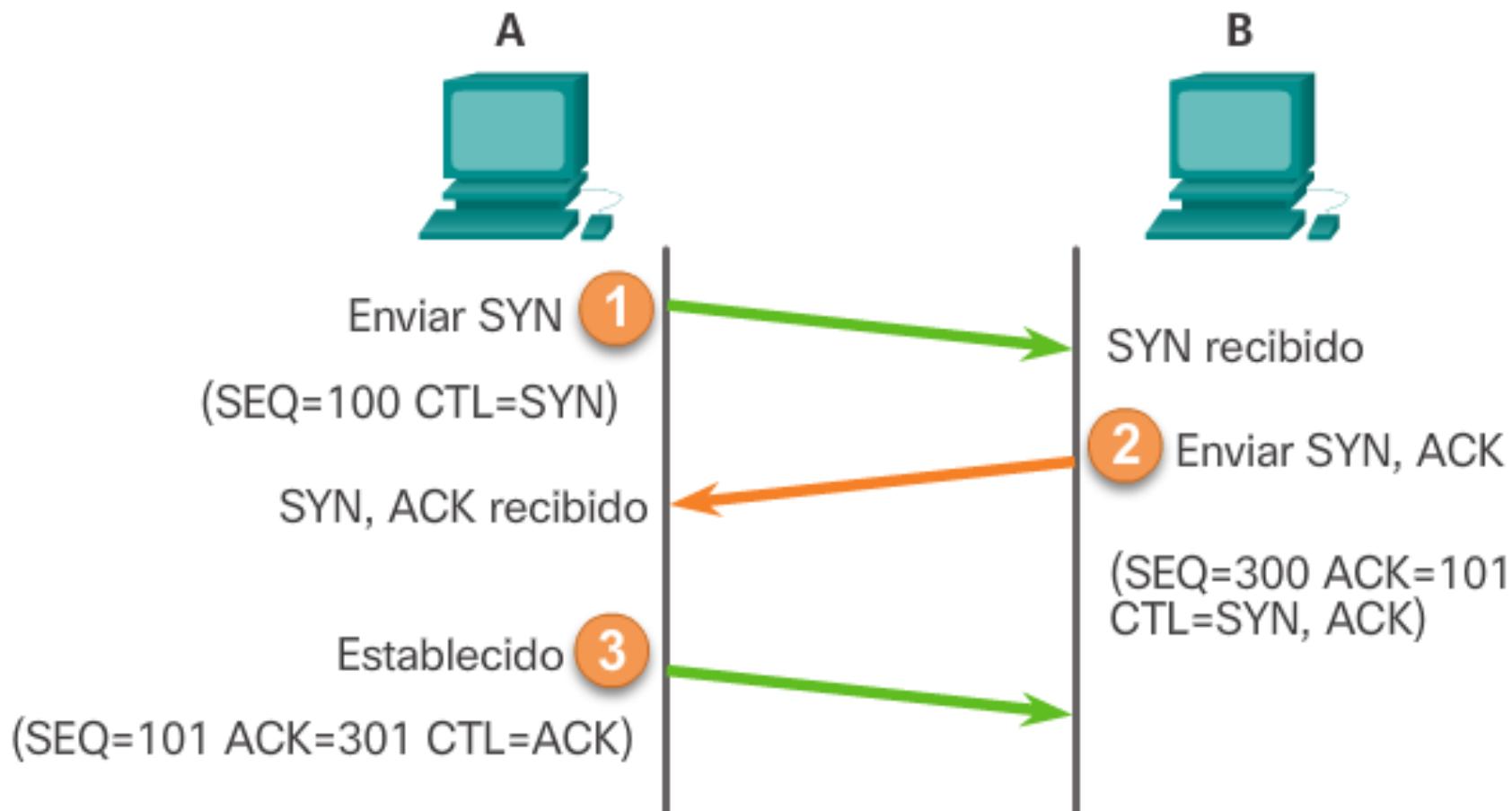
- Volviendo al tamaño de ventana, el **tamaño de ventana de 64 KB como máximo con frecuencia es un problema**: Ejemplo: en una **línea T3 (44.736 Mbps)** se requieren 12 mseg para enviar una ventana completa de 64 KB. Si el retardo de propagación es de 50 mseg (fibra transcontinental), **el emisor estará inactivo $\frac{3}{4}$ del tiempo en espera de confirmación de recepción**. De ahí en RFC1323 se propuso una opción de escala de ventana que permite **desplazar hasta 14 bits a la izquierda el campo Tamaño de ventana (2^{30})**.
- Otra opción propuesta en RFC 1106 es el empleo de la repetición selectiva en lugar de retroceso N, permitiendo al receptor solicitar un segmento/s específico.

Para establecer una conexión se utiliza el acuerdo de tres vías.

- El servidor espera pasivamente una conexión entrante ejecutando las primitivas **LISTEN** y **ACCEPT**.
- Del otro lado el cliente, ejecuta una primitiva **CONNECT** especificando la dirección y el puerto IP con el que desea conectar.
- La primitiva **CONNECT** envía un segmento TCP con el bit **SYN = 1** y el bit **ACK = 0** y espera una respuesta.
- Al llegar el segmento al destino, la entidad TCP ahí revisa si hay algún proceso ejecutando un **LISTEN** en el puerto indicado en el campo de Puerto de destino del segmento. Si no lo hay envía una respuesta con el bit **RST** encendido para rechazar la conexión.
- Si algún proceso está escuchando el puerto, ese **proceso** recibe el segmento TCP entrante y puede entonces aceptar o rechazar la conexión, si la **acepta** se devuelve un segmento de confirmación de recepción (**SYN = 1** y **ACK = 1**).

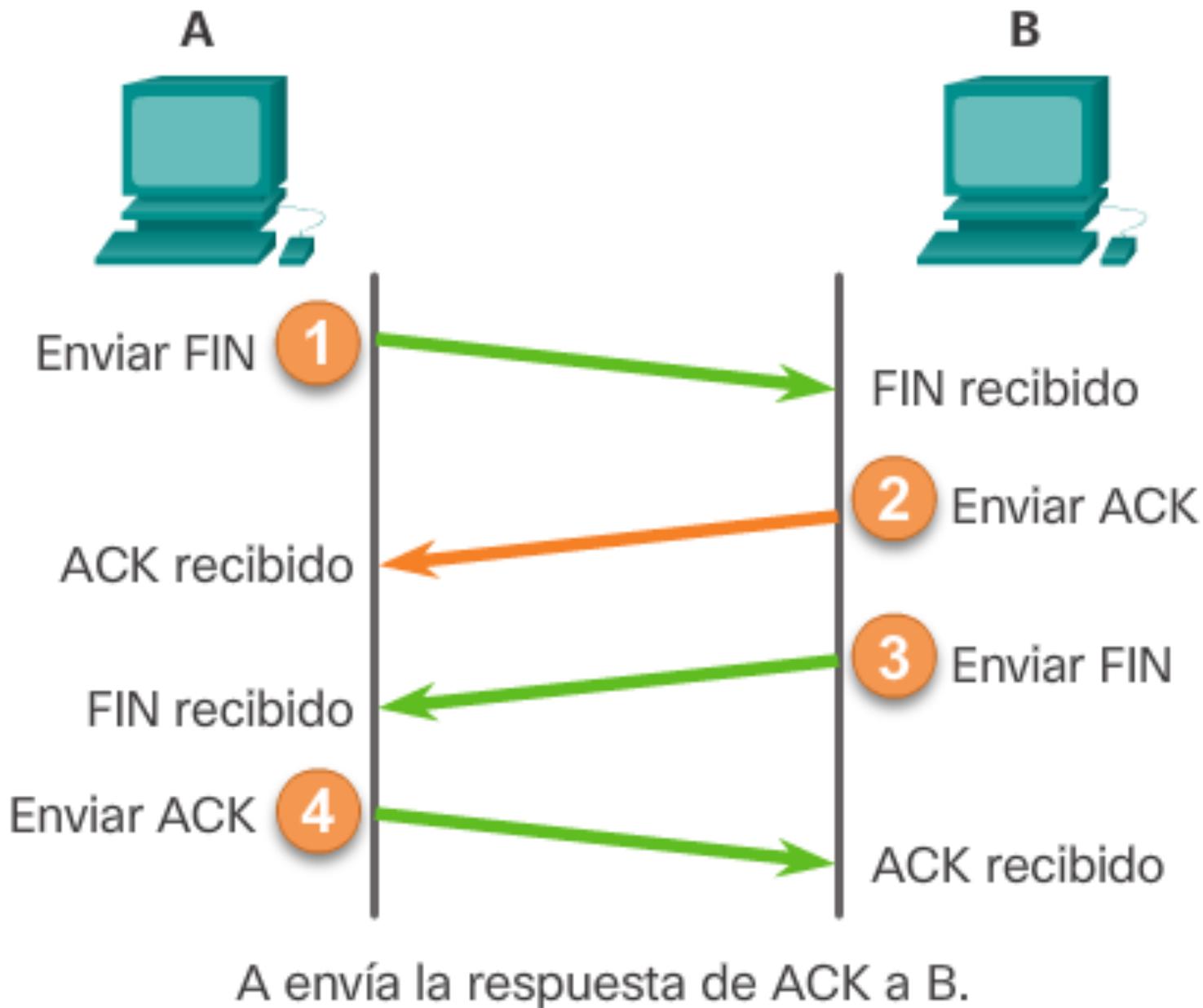


(a) Establecimiento de una conexión TCP en el caso normal

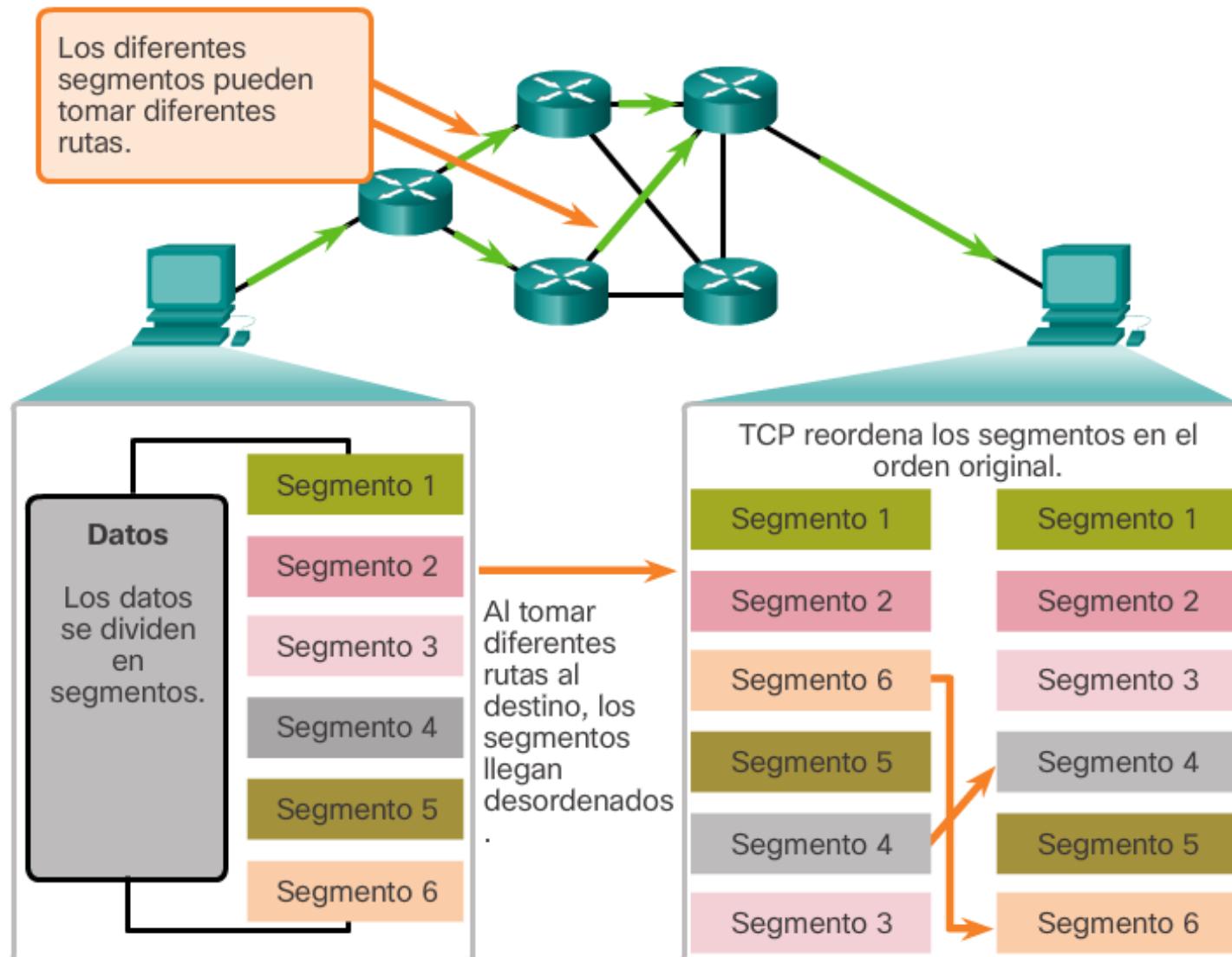


CTL = Cuáles bits de control en el encabezado TCP se establecieron en 1
A envía la respuesta de ACK a B

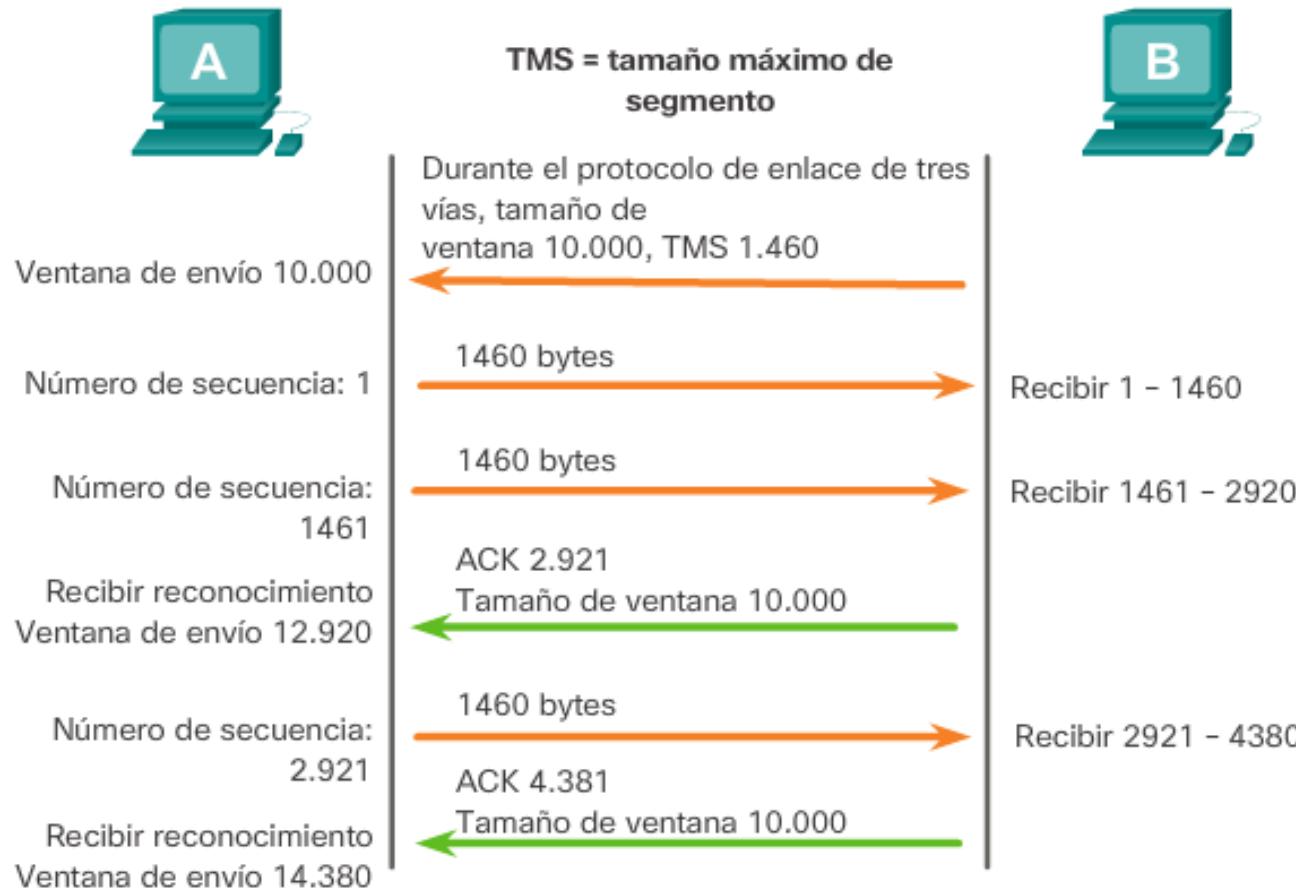
- Aunque las **conexiones TCP son dúplex total**, para entender la manera en que se liberan las conexiones es mejor visualizarlas como un par de conexiones simplex.
- Cada **conexión simplex se libera independientemente** de su igual
- Para **liberar una conexión**, cualquiera de las dos partes puede enviar un **segmento TCP con el bit FIN =1**, lo que significa que no tiene más datos por enviar.
- Al **confirmarse la recepción del FIN, ese sentido se apaga**, sin embargo, puede continuar un flujo de datos indefinido en el otro sentido.
- Para evitar el problema de los dos ejércitos se utilizan **temporizadores**. Si no llega una respuesta a un FIN en un máximo de dos tiempos de vida de paquete, el emisor del FIN libera la conexión. tarde o temprano el otro lado notará que nadie lo escucha y también expira su temporizador, liberando su conexión.



Confiabilidad de TCP: entrega ordenada

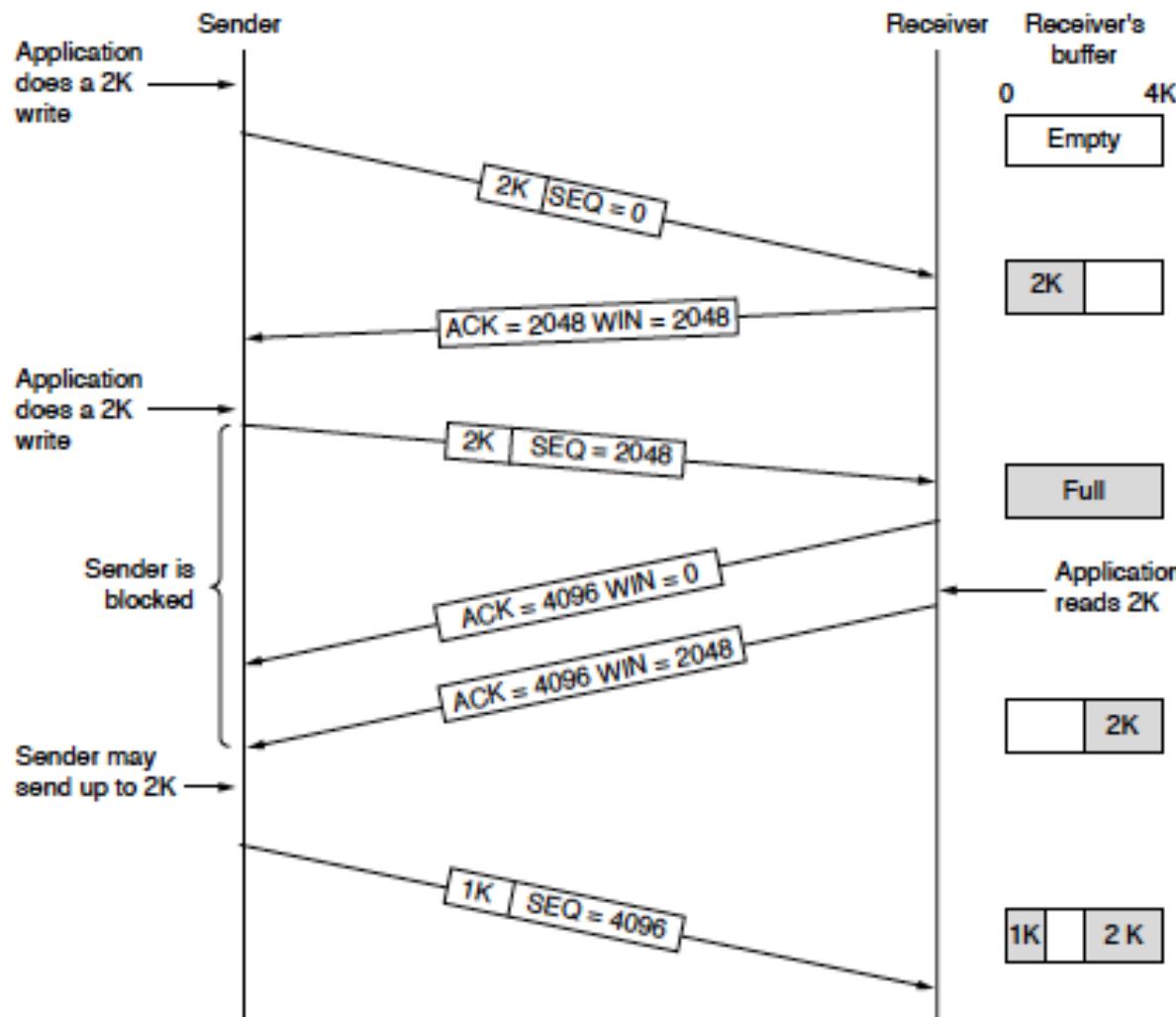


Confiabilidad de TCP: entrega ordenada

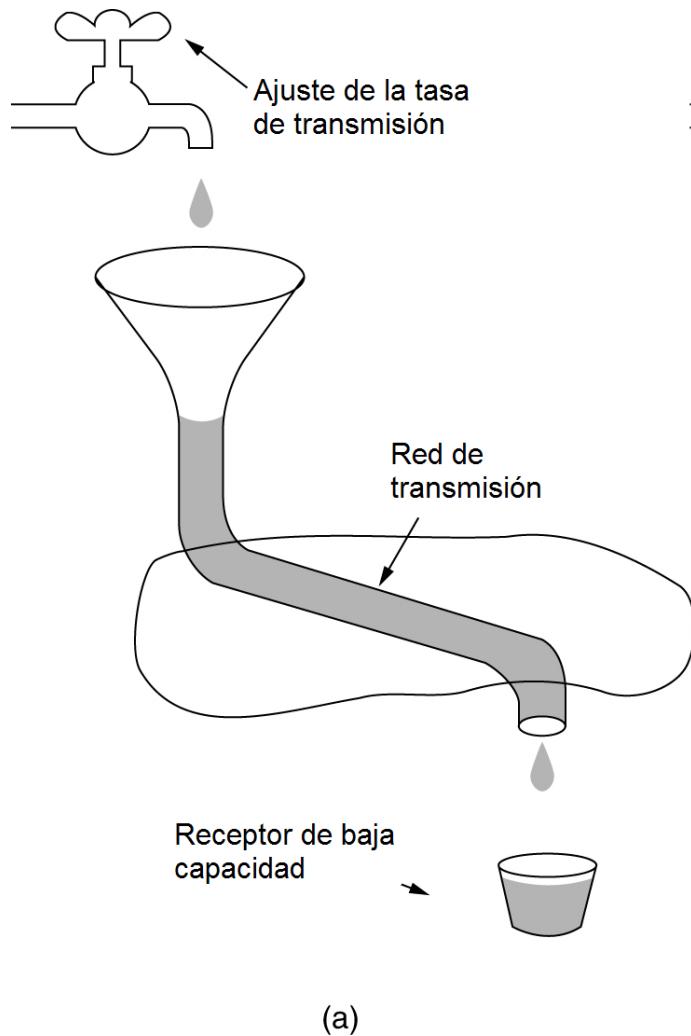


El **tamaño de la ventana** determina la cantidad de bytes que se pueden enviar para recibir un reconocimiento.

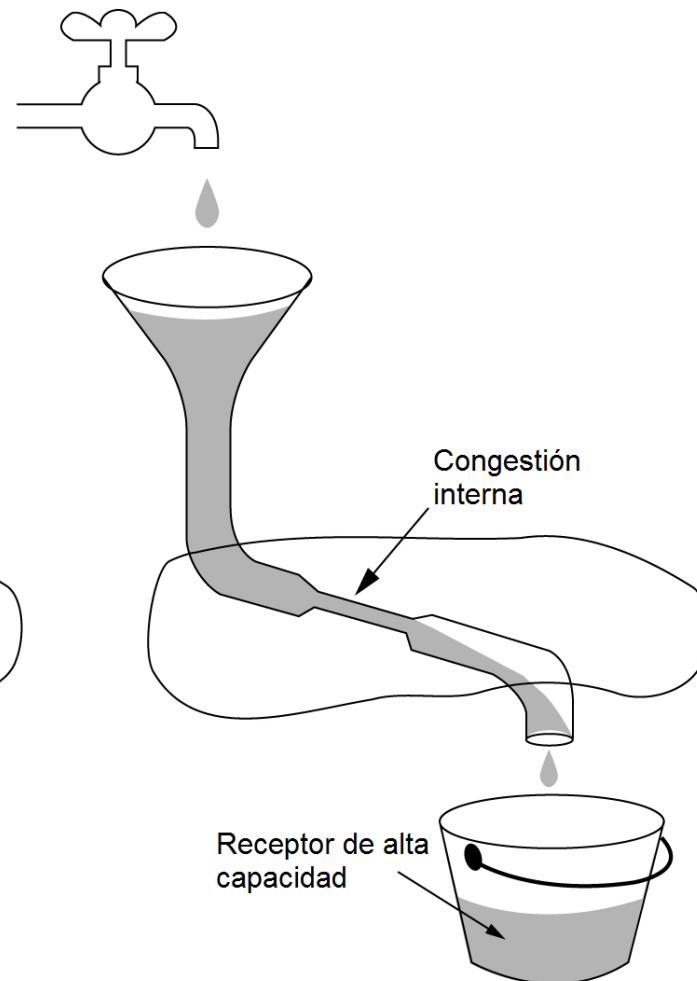
El número de **reconocimiento** es el número del siguiente byte esperado.



- Cuando la carga ofrecida a cualquier red es mayor que la que puede manejar, se genera **una congestión**. Internet no es una excepción.
- Aunque la capa de red también intenta manejarlos, la gran parte del trabajo recae sobre TCP porque la solución real a la congestión es la disminución de la tasa de datos.
- Hoy día la pérdida de paquetes por errores de transmisión es rara debido a que los troncales de larga distancia son fibra (excepto redes inalámbricas), por tanto la mayoría de las expiraciones de tiempo en Internet se deben a congestión.
- Para evitar que ocurra la congestión al establecerse una conexión se debe seleccionar un **tamaño de ventana adecuado**. El receptor puede especificar una ventana con base a su tamaño de búfer, si el emisor se ajusta a ese tamaño, no ocurrirán problemas de desbordamiento en la terminal receptora, pero aún pueden ocurrir debido a la **congestión interna de la red**.



(a)



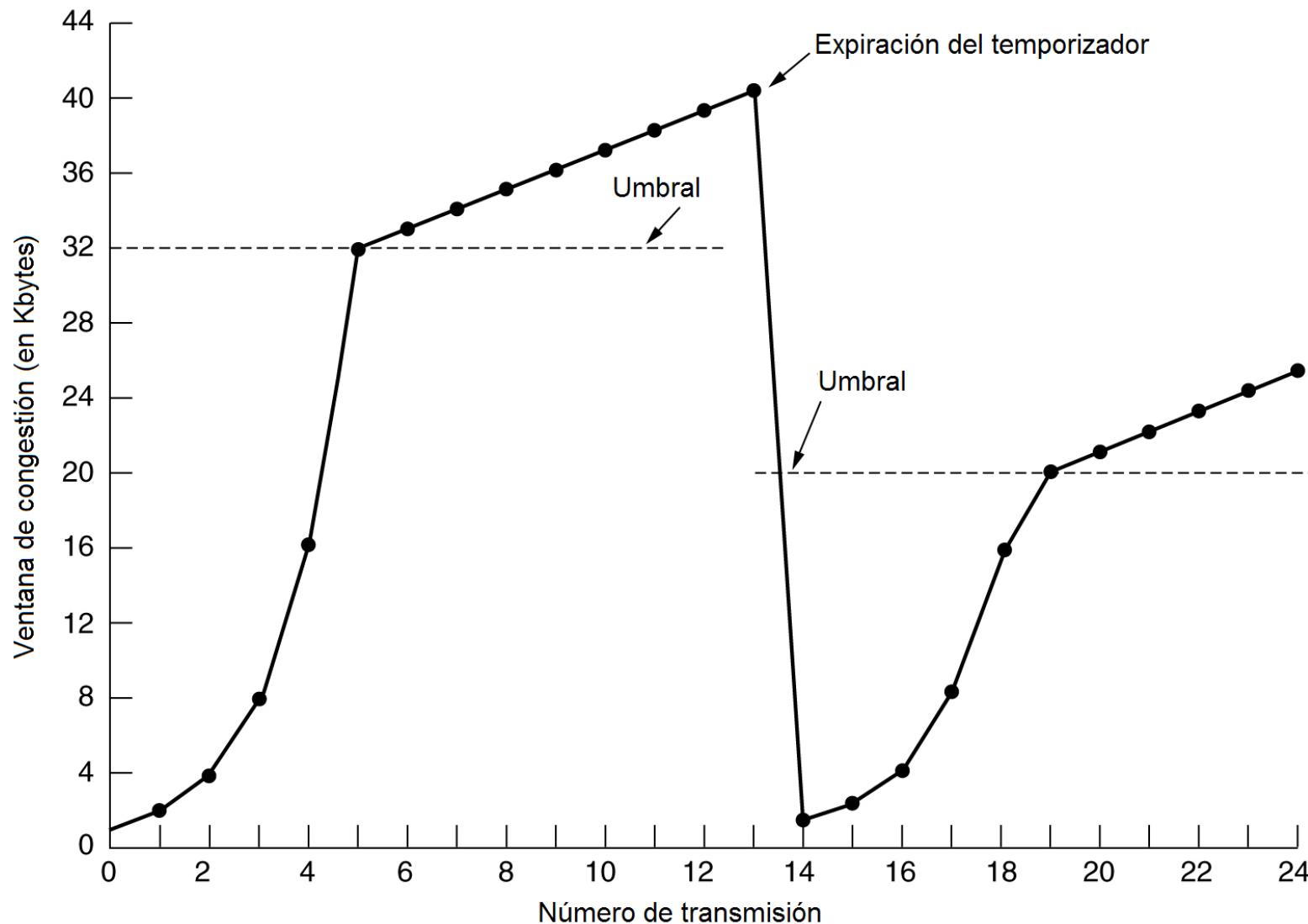
(b)

- (a) Red rápida alimentando un receptor de baja capacidad.
- (b) Red lenta alimentando un receptor de alta capacidad.

- La solución de Internet es aceptar que existen 2 problemas potenciales: la **capacidad de la red y la capacidad del receptor**
- Para evitar este problema en Internet se han propuesto diversas soluciones. Una de ellas consiste en la implementación de los procedimientos decremento multiplicativo e inicio lento a nivel de TCP. Para comprender estos conceptos definimos tres parámetros:
 1. Ventana permitida: correspondiente a la ventana de emisión o cantidad de datos que el emisor puede transmitir de forma consecutiva sin esperar confirmación.
 2. Ventana de recepción: correspondiente a la cantidad de datos que el receptor autoriza a enviar de forma consecutiva al emisor.
 3. Ventana de congestión, corresponde a la máxima cantidad de datos que se autoriza a transmitir en una situación de congestión.

- La relación existente es: ventana permitida = mínimo(ventana de recepción, ventana de congestión).
- Decremento multiplicativo:
 - Reducir la ventana de congestión a la mitad de su valor anterior cada vez que expire el temporizador asociado a la retransmisión de un segmento (notificación implícita de congestión)
- Inicio lento (Primera fase)
 - Inicialmente la ventana de congestión tiene el tamaño máximo de segmento (MSS).
 - Por cada segmento enviado con éxito la ventana se amplía en un MSS.
 - Esto supone un crecimiento exponencial (en potencia de 2).
 - Si la ventana de congestión supera a la de control del flujo, ésta será la que suponga una restricción.

- Inicio lento (Segunda fase)
 - Cuando se pierde un segmento, se sigue un esquema de prevención de la congestión.
 - La ventana de congestión vuelve a su valor inicial.
 - Se fija un ‘umbral de peligro’ en un valor igual a la mitad de la ventana que había cuando se produjo la pérdida.
 - La ventana de congestión crece como antes hasta el umbral del peligro; pero a partir de ahí crece en un solo segmento cada vez.



Ejemplo del algoritmo de congestión en Internet



Tema 5: La Capa de Transporte

1. Introducción
2. Elementos de los protocolos de transporte
3. Un protocolo de transporte sencillo
4. El protocolo de transporte UDP
5. El protocolo de transporte TCP
6. Aspectos de desempeño
7. Bibliografía



6. Aspectos de Desempeño

6.1 Introducción

6.2 Problemas de desempeño

6.3 Medición del desempeño

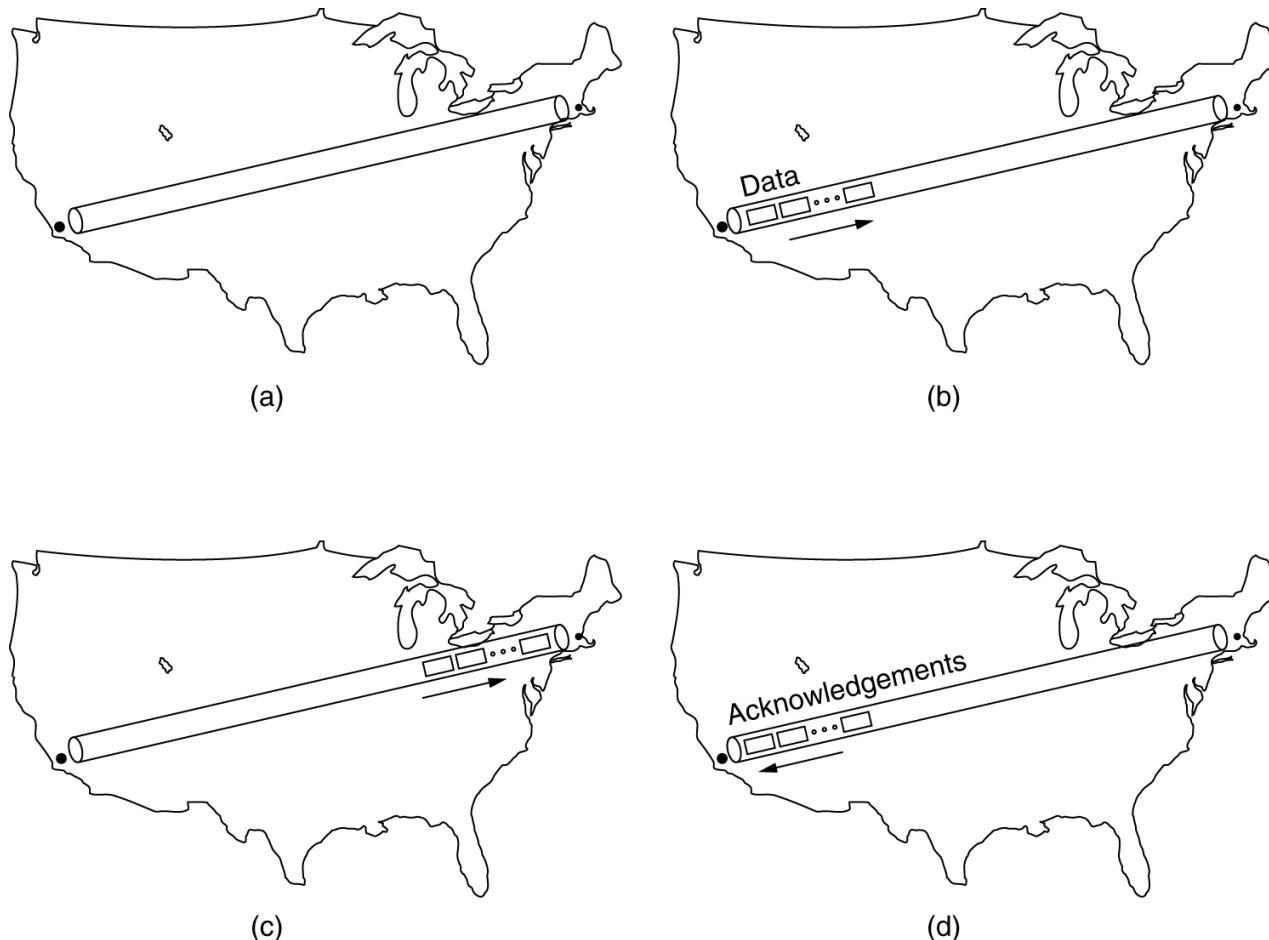
6.4 Diseño para mejorar el desempeño

6.5 Procesamiento rápido de las TPDUs

- Cuando hay cientos o miles de computadoras conectadas entre sí, son comunes las interacciones complejas con consecuencias imprevisibles.
- Teoría y práctica muchas veces no va de la mano y es la experiencia de trabajo de un administrador de redes lo que hace posible que el desempeño de una red sea el correcto.
- Como la capa de red está ocupada en cuestiones de enrutamiento y control de congestión, los puntos a analizar orientados al sistema están en la capa de transporte.
- Veremos algunos aspectos del desempeño de redes y cómo mejorarlo.

- Nos centramos en cuatro aspectos del desempeño de las redes:
 1. Problemas del desempeño
 2. Medición del desempeño de una red
 3. Diseño de sistemas con mejor desempeño
 4. Procesamiento rápido de las TPDUs.

- El desempeño se degrada cuando **no hay un equilibrio estructural de los recursos**. Líneas muy rápidas y computadoras lentas o viceversa.
- **Sobrecarga síncrona**: cuando hay un fallo del suministro, y todas las máquinas en una secuencia de arranque común acuden al servidor DHCP para conocer su identidad. Si todas las máquinas hacen esto al unísono el servidor se vendría abajo.
- Elección del tiempo correcto de expiración de los **temporizadores**: si se asigna un **valor muy bajo** ocurrirán retransmisiones innecesarias, congestionando los medios, **si es muy alto** ocurrirán retardos innecesarios tras la pérdida de una TPDU.
- Elección de un tamaño de **ventana de recepción adecuado** en función de la velocidad de la red.



Estado de transmisión de un megabit de San Diego a Boston

(a) En $t = 0$, (b) Tras 500 μ sec, (c) Tras 20 msec, (d) Tras 40 msec.

Una cantidad importante es $P = A \cdot B \cdot \text{retardo}$. Para lograr un buen desempeño la ventana del receptor debe tener cuando menos el tamaño P o ser un poco más grande

Cuando una red tiene un pobre desempeño los usuarios se quejan, exigiendo mejoras, le corresponde a los operadores de red, determinar exactamente lo que ocurre. El estudio siguiente se basa en el [trabajo de Mogul \(1993\)](#).

El ciclo usado para mejorar el desempeño de las redes:

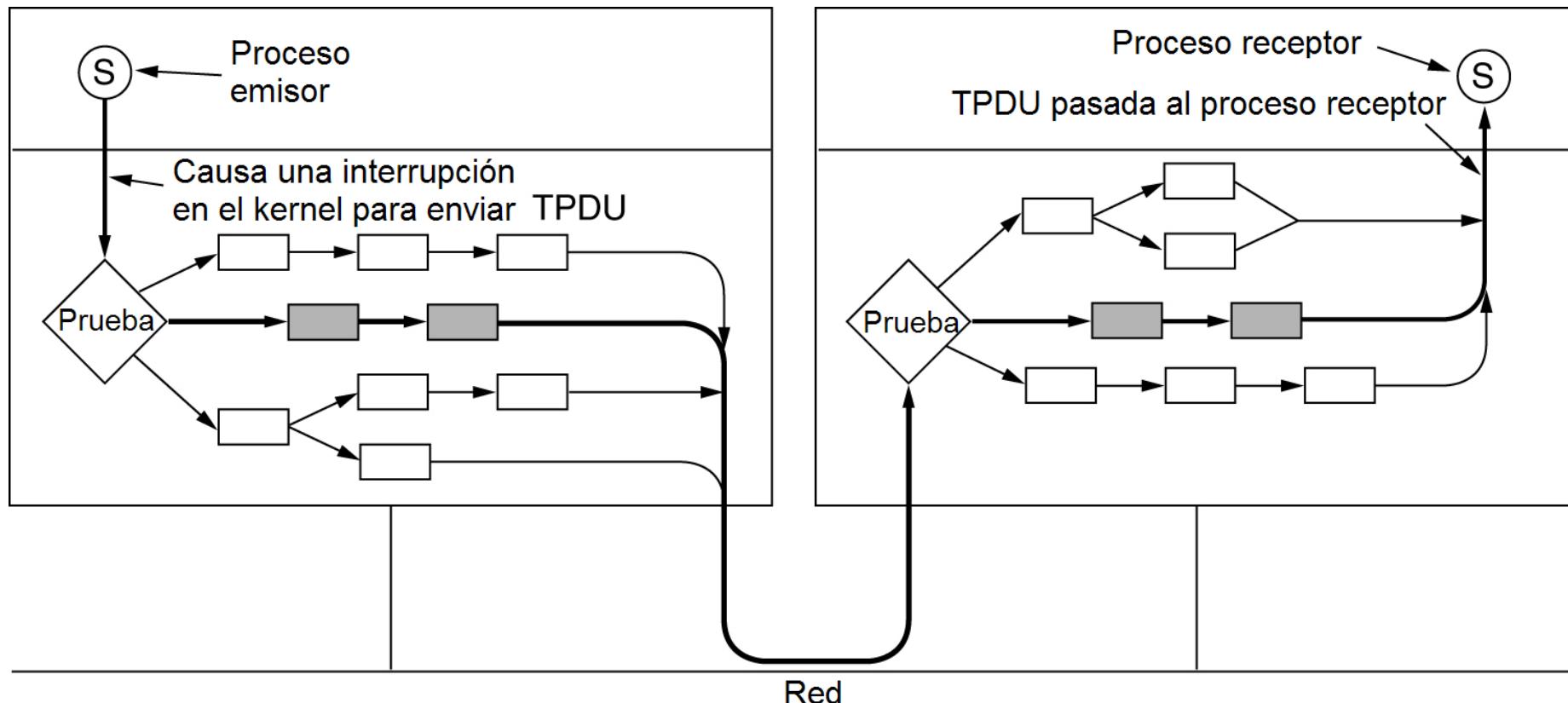
1. Medir los parámetros pertinentes y el desempeño de la red
2. Estudiar los parámetros para entender lo que ocurre
3. Cambiar un parámetro

Estos pasos se repiten hasta que el desempeño sea lo bastante bueno o que quede claro que se han agotado todas las mejoras posibles.

- Aspectos que deben tenerse en cuenta, cuando se procede a la medición del desempeño y al establecimiento de los parámetros:
 1. Asegurarse que el tamaño de la muestra es lo bastante grande.
 2. Asegurarse de que las muestras son representativas.
 3. Tener cuidado al usar relojes de intervalos grandes.
 4. Asegurarse de que no ocurre nada inesperado durante las pruebas.
 5. Tener en cuenta la caché en las mediciones.
 6. Tener cuidado con la extrapolación de los resultados.

- Para mejorar el desempeño, es importante un buen diseño de la red. Una red mal diseñada puede mejorarse sólo hasta un límite.
- Mogul fue el primero en postular una serie de reglas para el buen diseño de una red, producto del conocimiento común de diseñadores durante generaciones:
 1. La velocidad de la CPU es más importante que la velocidad de la red.
 2. Reducir el número de paquetes para reducir la sobrecarga de software (procesamiento de encabezados, procesamiento de bytes, suma verificación).
 3. Reducir al mínimo las commutaciones de contexto: del usuario al kernel y viceversa, el procedimiento de biblioteca debe guardar la mayor cantidad de datos antes de enviar.

4. Reducir al mínimo las copias: paquete se copia de búfer de tarjeta de red a un búfer del kernel, de ahí a la capa de red, luego búfer de la capa de transporte y por último en el proceso de aplicación.
5. Es posible comprar más ancho de banda, pero no un retardo menor.
6. Evitar la congestión es mejor que recuperarse de ella.
7. Evitar expiraciones del temporizador.



La moraleja de los anterior es que el obstáculo principal en las redes rápidas es el software de los protocolos. La trayectoria rápida del emisor al receptor se indica con una línea gruesa. Los pasos de procesamiento de esta trayectoria se muestran sombreados. **Los encabezados de TPDU consecutivas son casi iguales.** Para aprovechar esto al principio de la trayectoria, se copia el encabezado en un búfer de trabajo palabra por palabra y sólo sobreescrivo los campos que varían. Después se pasa a la capa de red un apuntador al encabezado, y otro a los datos y la capa de red puede hacer lo mismo antes de pasarlo a la capa de enlace de datos.

Puerto de origen	Puerto de destino
Número de secuencia	
Número de confirmación de recepción	
Long	Sin usar
Suma de verificación	Apuntador urgente

(a)

VER.	IHL	TOS	Longitud total
Identification			Desplazamiento de encabezado
TTL	Protocol	Suma de verificación del encabezado	
Dirección de origen			
Dirección de destino			

(b)

(a) Encabezado TCP. (b) Encabezado IP. En ambos casos, los campos sombreados se toman sin cambios del prototipo.



Tema 5: La Capa de Transporte

1. Introducción
2. Elementos de los protocolos de transporte
3. Un protocolo de transporte sencillo
4. El protocolo de transporte UDP
5. El protocolo de transporte TCP
6. Aspectos de desempeño
7. Bibliografía

- D.E. Comer. Internetworking with TCP/IP. Volumen I: Principles, Protocols and Architecture, 3^a Edicion, Prentice Hall, 1995.
- A. S. Tanenbaum. Redes de Computadoras, 5^a Edición. Prentice-Hall, 2012.
- W.R. Stevens. TCP/IP Illustrated, Vol.1 The Protocols, Ed. Addison Wesley, 2000.
- W. R. Stallings. Comunicaciones y Redes de Computadoras, 7^a Edición. Prentice-Hall, 2004.
- L.L. Peterson. Computer Networks. A System Approach, Morgan Kaufmann, 1996.