20

PROTOCOLO TCP

20.1 INTRODUCCIÓN

En la figura 20.1 se presenta el ambiente TCP/IP y el ambiente ISO.

En el caso del primero, el protocolo IP siempre está presente en la capa de red, sin tener en cuenta los tipos de redes de datos del nivel inferior, sean una red LAN o red WAN. Las unidades de datos asociados con los protocolos de la capa de transporte se transfieren a través de las capas de redes subyacentes en los datagramas IP. Así, existe un acoplamiento muy cercano entre los protocolos de la capa de transporte y el protocolo IP.

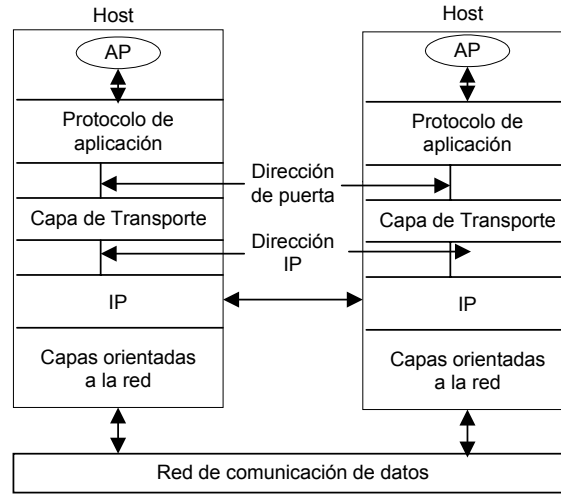

Figura 20.1 Posición de la capa de transporte en ambiente TCP/IP

En contraste, en el caso del ambiente OSI, los protocolos orientados a la red normalmente reflejan el tipo de red subyacente que está siendo usada. Esto significa que la interface de la capa de red podría orientarse a la conexión, si la capa subyacente está basada en el protocolo X.25, o ser no orientada a la conexión si la capa subyacente usa el protocolo ISO-IP. Más aún, en el caso del ambiente TCP/IP, el protocolo de aplicación accede directamente a la capa de transporte. En el caso de la suite OSI, los protocolos de aplicación acceden a la capa de transporte a través de las capas intermedias de sesión y presentación.

Pese a estas diferencias, ambas capas de transporte realizan el mismo conjunto de funciones. De este modo, hay un protocolo no orientado a la conexión (*best-try*) asociado con ambas arquitecturas, así como un protocolo confiable u orientado a la conexión. Obviamente no todas las aplicaciones requieren un servicio confiable. Por ejemplo, al transferir archivos de imágenes digitales, los errores casuales son relativamente menos importantes que la velocidad de transferencia.

Un requerimiento similar se aplica a los mensajes de voz digitalizada. Es importante tomar en cuenta que la tasa de errores puede ser baja en redes de área local, por lo cual los protocolos no orientados a la conexión deberían emplearse sólo donde los errores ocasionales son aceptables. Para otras aplicaciones se utilizan los protocolos orientados a la conexión. A continuación, se describen los dos protocolos asociados con ambos ambientes: el UDP y el TCP.

20.2 PROTOCOLO DATAGRAMA DE USUARIO - UDP

El protocolo de datagrama de usuario (*User datagram Protocol* – UDP) opera sobre el IP, provee un servicio no orientado a la conexión, con la transferencia de mensajes direccionados individuamente. Este modo de operación minimiza la cantidad de sobrecabeceras asociadas con cada transferencia de mensaje, debido a que no se establece previamente ninguna conexión de red antes de enviar este paquete. Además el IP no realiza control de errores.

Por otro lado, el protocolo TCP/IP que transporta al proceso de la aplicación de usuario se conoce como protocolo de control de transmisión (*Transmission Control Protocol* – TCP). La figura 20.2 muestra la posición de ambos protocolos en relación con otros protocolos en conjunto, con sus unidades de datos y las direcciones de intercapas. En esta sección describimos al UDP y en la siguiente, al TCP.

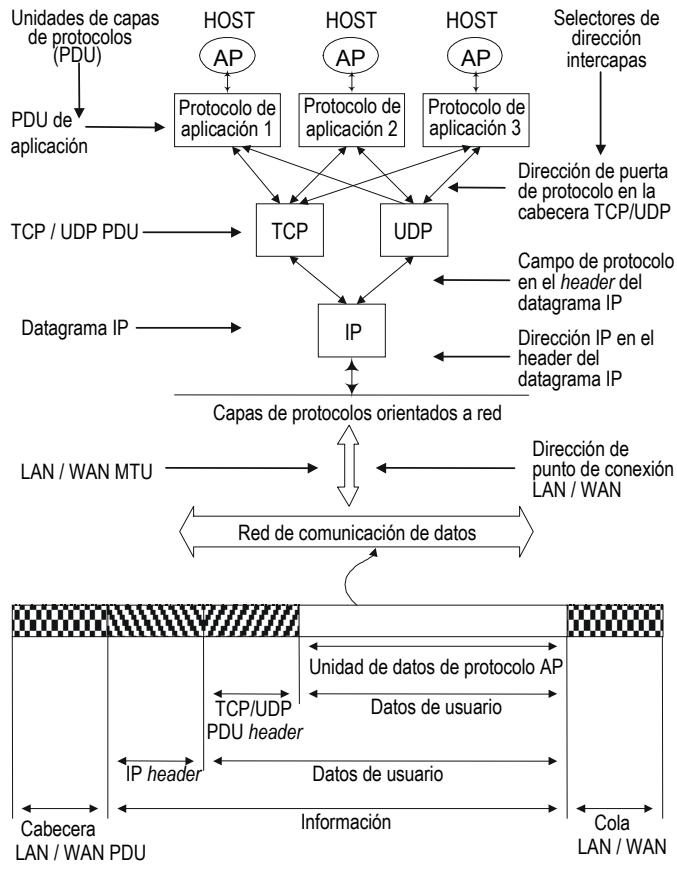

Figura 20.2 Protocolo TCP/IP con las unidades de datos asociados y los selectores de direcciones intercapas

Esta estructura de la trama de información se trasmite o se recibe por un *host*. En la interface de red, ésta comprende la cabecera de la trama LAN/WAN, el campo de datos de información de usuario y la cola asociada (marcados de fin de trama con el control de errores). Como se recuerda, la dirección IP comprende a las direcciones *netid* y *hostid*, las cuales se usan para enrutar los datagramas a un *host* específico. También en la cabecera del datagrama hay un campo de tipo de protocolo que indica el protocolo que está asociado dentro del computador, al cual el campo de datos de usuario en el datagrama deberá ser entregado. Éste podría ser un protocolo asociado con el IP, tal como el protocolo de control de errores (*Internet Control Message Protocol* -ICMP), o uno de los dos protocolos de transporte UDP o TCP. En la figura 20.2 presentamos las aplicaciones del UDP y el TCP.

En notación decimal, la dirección de un protocolo de aplicación tiene la siguiente forma:

128.3.2.3,53

La primera parte viene a ser: dirección IP-*netid* = 128.3 y dirección *hostid* = 2.3. La parte que sigue a la coma, es la dirección de puerta (53) del protocolo de aplicación. En la terminología ISO esta dirección compuesta se conoce como una **dirección completamente calificada** o ***socket.***

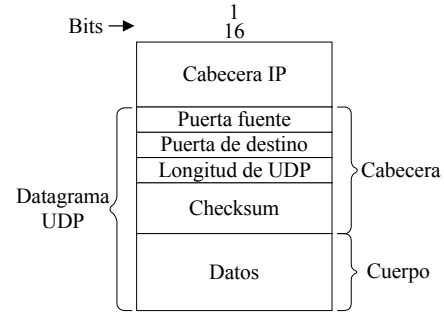

Figura 20.3 Formato del datagrama UDP

20.2.1 ESTRUCTURA DEL FORMATO DEL DATAGRAMA UDP

Como ya se vio, el UDP es un protocolo no orientado a la conexión con una sola PDU, conocida también como datagrama, que se transmite en el campo de datos del datagrama IP. La figura 20.3 muestra el formato de la cabecera del datagrama de usuario. A continuación, describiremos sus campos.

- **Puerta de origen** o **fuente** (*source port*)**:** Dirección de puerta de aplicación origen. Tiene 16 bits.
- **Puerta de destino** (*destination port*)**:** Dirección de puerta del protocolo de aplicación a la cual se desea llegar. Tiene 16 bits. En un datagrama UDP, la puerta origen es opcional y se incluye sólo si se espera una respuesta. Si ésta no se requiere, se le pone 0.
- **Longitud:** Número total de octetos en el datagrama completo UDP; incluye su cabecera.
- ***Checksum*:** Es el complemento de dígitos **1** de 16 bits, de la suma de complemento de dígitos 1 de una seudocabecera de información conformada por los campos de la dirección de fuente, dirección de destino, el tipo de protocolo (en este caso es igual a 17 para el UDP), la longitud del UDP y un relleno de dígitos **0** para completar un múltiplo par de octetos. En la figura 20.4 mostramos esta seudocabecera que se emplea para el cálculo del *checksum*.

El UDP es muy útil en aplicaciones orientadas al comando/respuesta y donde éstos pueden enviarse en un solo datagrama.

0 7	8 15	16 23	24 31
Dirección de fuente			
Dirección de destino			
Ceros	Protocolo	Dirección del UDP	

Figura 20.4 Campos de seudocabecera UDP usada para calcular el *checksum*

20.2.2 CARACTERÍSTICAS DEL UDP

- Identifica a los procesos aplicativos utilizando sus números de puertas.
- Es un protocolo no orientado a la conexión, por ende no tiene la sobrecabecera de las fases de establecimiento, mantenimiento y liberación de una conexión.
- No tiene secuenciamiento de los datos, por lo cual no puede garantizar una entrega confiable.
- Es eficiente en las aplicaciones *broadcast* y *multicast*.
- Es más rápido, simple y eficiente que el TCP, sin embargo no es tan robusto como éste.

20.2.3 APLICACIONES QUE CORREN SOBRE EL UDP

Pese a ser un protocolo no tan confiable como el TCP, el UDP sirve de plataforma para varias aplicaciones prácticas, entre las que se incluyen:

- Protocolo trivial de transferencia de archivos (*Trivial File Transfer Protocol* –TFTP). Port: 69.
- Sistema de dominio de nombres (*Domain Name Sistem* –DNS). Port: 53 (*Domain Name System server – domain*).
- Sistema de archivos en red (*Network File System* –NFS).
- Protocolo simple de administración de red (*Simple Network Management Protocol* –SNMP). *Port*: 161.
- Protocolo de enrutamiento de información (*Routing Information Protocol* –RIP).

20.3 PROTOCOLO DE CONTROL DE TRANSMISIÓN – TCP

El UDP se usa cuando no se necesita corrección de errores o al intercambiar mensajes de pregunta/respuesta entre dos protocolos de aplicación. Sin embargo, la mayoría de aplicaciones distribuidas de sistemas abiertos requiere de un servicio de transporte de mensajes confiable, es decir orientado a la conexión; por ejemplo: la transferencia de un archivo con los registros bancarios de un cliente. Obviamente, en estas aplicaciones la corrección de tan sólo un bit binario es crucial.

En el ambiente TCP/IP, el protocolo de transporte orientado a la conexión se conoce como el protocolo de control de transmisión (*Transmission Control Protocol* – TCP), mientras que el servicio que ofrece a los usuarios a través de los protocolos de aplicación se denomina "servicio confiable de transporte de datos".

20.3.1 SERVICIO DE TRANSPORTE CONFIABLE DE DATOS - TCP

Este servicio es similar a los servicios de usuario asociados con la clase 4 del protocolo de transporte de la ISO. Se proporciona primitivas de servicio para:

- Establecer una conexión lógica con un protocolo par en un computador remoto.
- Intercambiar mensajes sobre esta conexión de un modo *duplex*.

- Habilitar un protocolo de aplicación, a través de las capas de presentación y sesión en el caso de un ambiente ISO.
- Liberar una conexión.

El protocolo TCP realiza las funciones necesarias para transferir los datos asociados con estos intercambios de manera confiable, esto es libre de errores, sin pérdidas o duplicidad y en el mismo orden que fueron entregados. A continuación, desarrollamos las funciones más notables del TCP:

a) Transferencia básica de datos

El término TREN DE DATOS CONFIABLE se usa con el TCP porque trata a todos los datos del usuario asociados con una conexión como una secuencia de mensajes de pregunta/respuesta, por ejemplo, como dos trenes de datos separados, uno en cada dirección, compuestos por cadenas de octetos. La orientación del TCP es enviar una secuencia de octetos y no bloques como otros protocolos.

Para conseguir un servicio confiable, el protocolo TCP transmite todos los datos en unidades conocidas como **segmentos**. Normalmente, el protocolo TCP decide cuándo un nuevo segmento debe transmitirse. En el lado de destino, el protocolo TCP receptor almacena los datos recibidos en una memoria *buffer* asociada con la aplicación y los entrega cuando se llena el *buffer*. De esta manera, un segmento puede consistir en múltiples mensajes de usuario si se intercambian unidades cortas de mensajes, o parte de un solo mensaje grande si se transfiere un archivo extenso. La longitud máxima de cada segmento es una función del protocolo TCP, el cual asegura que el tren de datos asociados en cada dirección sea entregado al otro lado de una manera confiable.

b) Confiabilidad

Uno de los aspectos más importantes del TCP consiste en la entrega confiable de datos, de extremo a extremo. Para proporcionar esta confiabilidad y recobrar los datos que se dañen, se pierdan, se dupliquen o se lleven fuera de secuencia en la capa de red, el TCP emplea la retransmisión positiva de confirmación (*Positive Acknowledgement Retransmisión* -PAR). La figura 20.5 ilustra un ejemplo de recuperación de datos perdidos con temporizador.

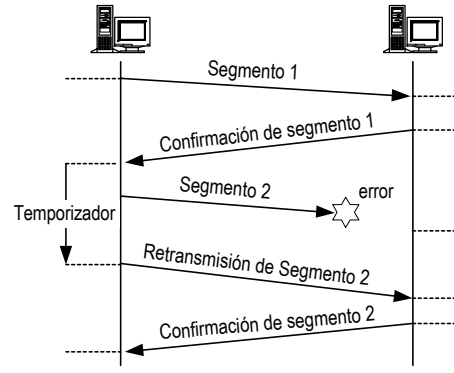

Figura 20.5 Recuperación de datos perdidos con temporizador

c) Control de flujo

Las computadoras que envían y reciben segmentos TCP operan a distintas velocidades de transmisión, debido a sus diferentes procesadores y anchos de banda de la red. Como resultado es probable que un transmisor envíe segmentos más rápido de lo que el receptor pueda asimilar. Para controlar este flujo se usa el mecanismo de ventana a nivel de octetos y el número de secuencia de octeto. La figura 20.11 muestra un ejemplo de este control de la fase de transferencia TCP.

d) Multiplexaje

El TCP permite que una computadora pueda utilizar diversos servicios de comunicaciones simultáneamente sobre una sola interface física. Para ello usa las direcciones de puertas. En la figura 20.6 presentamos diversas aplicaciones corriendo con diferentes puertas.

e) Conexiones

Una conexión está completamente especificada por un par de puntos extremos. Un punto extremo local puede participar en muchas conexiones con varios puntos extremos remotos.

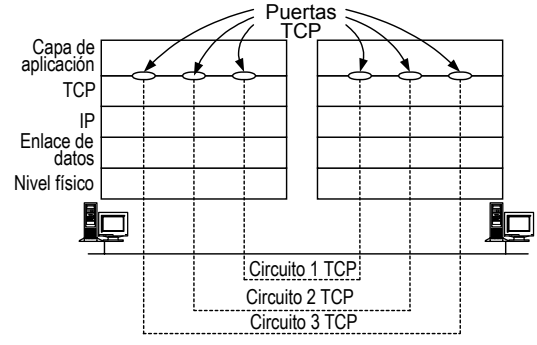

Figura 20.6 Múltiples conexiones entre dos *hosts*

20.3.2 APLICACIONES QUE OPERAN SOBRE TCP

Entre las aplicaciones que corren sobre TCP tenemos las siguientes:

- Protocolo de transferencia de archivos (*File Transfer Protocol* – FTP) – Módulo de transferencia de datos. Port: 20.
- Protocolo de transferencia de archivos (*File Transfer Protocol* – FTP) – Módulo de control. Port: 21.
- Protocolo simple de transmisión de correo (*Simple Mail Transfer Protocol* – SMTP). Port: 25.
- Sistema de dominio de nombres (*Domain Name System* – DNS). Port: 53.
- Telnet. Port: 23.
- Servidor de Protocolo *Bootstrap* (*Bootstrap Protocol Server* – Bootps). Port: 67.
- Cliente de Protocolo *Bootstrap* (*Bootstrap Protocol Client* – Bootps). Port: 68.
- Protocolo *Gopher*. Port: 70.
- Protocolo *Finger*. Port: 79.
- *World Wide Web* HTTP. Port: 80.
- *Post Office Protocol v.3* (POP3). Port: 110.
- *Internet Relay Chat Protocol* (IRC). Port: 194.

20.3.3 PRIMITIVAS DE SERVICIOS DE USUARIO DEL TCP

Una lista de las primitivas de servicios de usuario asociados con el TCP/IP, junto con sus parámetros se muestran en la tabla 20.1. La mayoría de las aplicaciones distribuidas de los sistemas abiertos están basadas en el **modelo cliente/servidor**. Esto se entiende mejor considerando a un programa de aplicación (proceso) accesando y usando un sistema remoto de archivos (proceso de aplicación). El sistema de archivo es de esta manera el **servidor**, debido a que sólo responde a las peticiones de los servicios de archivo y el programa de aplicación es el **cliente** debido a que este siempre inicia las peticiones.

PRIMITIVA	TIPO	CLIENTE/ SERVIDOR	PARÁMETROS
Apertura pasiva no específica *UNSPECIFIED PASSIVE_OPEN*	Petición	S	Puerta de origen, ***temporizador, acción en caso de expiración de temporizador, precedencia, rango de seguridad***.
Apertura completa pasiva *FULL_PASSIVE_OPEN*	Petición	S	Puerta de origen, puerta de destino, dirección de destino, ***temporizador, acción en caso de expiración de temporizador, precedencia, rango de seguridad***.
Apertura activa *ACTIVE_OPEN*	Petición	C	Puerta de origen, puerta de destino, dirección de destino, ***temporizador, acción en caso de expiración de temporizador, precedencia, rango de seguridad***.
Apertura activa con datos incluidos *ACTIVE_OPEN_WITH DATA*	Petición	C	Puerta de origen, puerta de destino, dirección de destino, ***temporizador, acción en caso de expiración de temporizador, precedencia, rango de seguridad***.
Identificativo de la apertura *OPEN_ID*	Respuesta (local)	C	Nombre de conexión local, puerta origen, puerta destino, dirección destino.
Apertura exitosa *OPEN_SUCCESS*	Confirmación	C	Nombre de conexión local.
Apertura fallada *OPEN_FAILURE*	Confirmación	C	Nombre de conexión local.
Transmitir *SEND*	Petición	C/S	Nombre de conexión local, datos, longitud de datos, flag de push, flag de urgente, ***temporizador, acción en caso de expiración de temporizador***.
Entregar *DELIVER*	Indicación	C/S	Nombre de conexión local, datos, longitud de datos, flag de urgente.
Asignar *ALLOCATE*	Petición	C/S	Nombre de conexión local.
Cerrar *CLOSE*	Petición	C/S	Nombre de conexión local.
Cerrando *CLOSING*	Indicación	C/S	Nombre de conexión local, código de motivo.
Terminar *TERMINATE*	Confirmación	C/S	Nombre de conexión local.
Abortar *ABORT*	Petición	C/S	Nombre de conexión local.
Estado *STATUS*	Petición	C/S	Nombre de conexión local.
Respuesta de estado *STATUS_RESPONSE*			Nombre de conexión local, puerta origen, dirección de origen, puerta de destino, ventana de recepción, ventana de transmisión, esperando ACK, esperando recepción, urgente, ***precedencia, rango de seguridad, temporizador***.
ERROR	Indicación	C/S	Nombre de conexión local, código de motivo.

Tabla 20.1 Primitivas TCP de servicio de los usuarios y sus parámetros

Nótese también que un proceso de un solo servidor puede normalmente tener la capacidad de soportar accesos de una comunidad de clientes distribuidos de una manera concurrente. Muchas de las primitivas de servicio de usuario reflejan este modo de interacción. Aquellas que se relacionan a cada tipo de usuario –Cliente/Servidor– se indican en la tabla 20.1.

El parámetro de **dirección de destino** es la dirección IP del computador de destino y las **puertas de origen** y **de destino** son las direcciones de puerta de las aplicaciones de origen y de destino, respectivamente.

El parámetro del **temporizador** permite al protocolo de aplicación de usuario especificar el intervalo de tiempo máximo que el protocolo origen TCP deberá esperar por una confirmación al segmento que ha enviado. Como se recordará, debido a que el IP provee un servicio no orientado a la conexión, un segmento podría ser descartado durante la transferencia. Por eso, el valor del temporizador es mayor que el doble del tiempo de vida de cada datagrama IP. El parámetro de **acción frente a la expiración de un temporizador** especifica la acción que se ejecutará ante este hecho. La acción normal es cerrar la conexión.

El parámetro de **precedencia** es una colección de bits que permiten al usuario especificar el tipo de contenido del campo de tipo de servicio. Este parámetro está en la cabecera del datagrama IP. Su contenido ya lo explicamos al tratar sobre el protocolo IP. Como se recordará, éste cumple la misma función que el parámetro de calidad de servicio (*Quality Of Service* - QOS), asociado con las primitivas ISO correspondientes. Nótese que el protocolo IP usa este parámetro, mas no el TCP. Éste es entonces un ejemplo de parámetro atravesado (*pass-through*), es decir que este parámetro se transfiere de capa de protocolo a otra capa de protocolo sin modificación.

El parámetro del **rango de seguridad** permite a la aplicación del servidor especificar el nivel de seguridad que será aplicado a los usuarios. Las banderas o *flags* de ***push*** y **urgente** habilitan al usuario para indicar al TCP cómo debería éste tratar los datos:

- **Push** significa que los datos deben transmitirse inmediatamente.
- **Urgent** indica que los datos se deberán transmitir fuera del flujo normal.

Finalmente, el nombre de **conexión local** le asigna la entidad local del protocolo TCP al establecer por primera vez una conexión. Aunque la dirección de puerta habilita al TCP a relacionar a un segmento recibido a un protocolo de aplicación particular, en el caso de un servidor se pueden tener múltiples transacciones –y de aquí, múltiples conexiones lógicas– progresando concurrentemente. El nombre de **conexión local** se usa para permitir que el TCP relacione primitivas a la misma conexión. El parámetro de identificador de punto de conexión final en una suite OSI lleva a cabo la misma función. La mayoría de los parámetros de la primitiva de STATUS están relacionados con la entidad de protocolo TCP y será tratada más adelante.

Muchos de los parámetros se relacionan con una operación de relativo bajo nivel del TCP/IP. Sin embargo, en muchos casos los mismos valores de parámetros pueden usarse con distintas primitivas. Para permitir esto, los parámetros en ***cursiva*** tienen valores por defecto. Si un valor no está explícitamente establecido se asumen los valores por defecto. La figura 20.7 ilustra la interrelación de las diversas primitivas en forma de diagrama de secuencia en el tiempo.

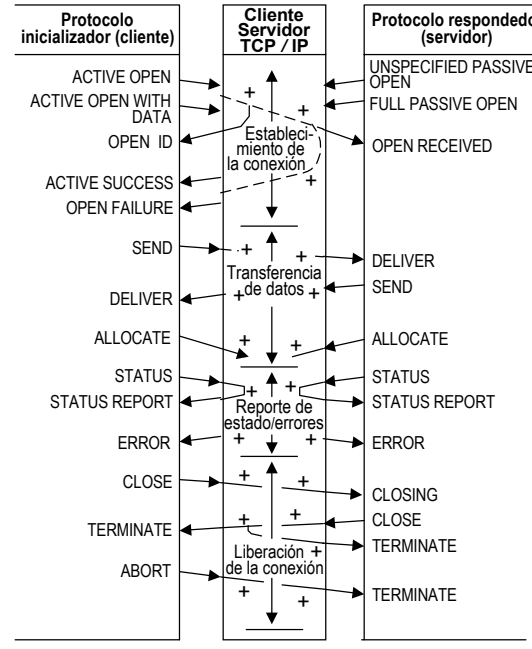

Figura 20.7 Diagrama de secuencia en tiempo de los servicios del usuario TCP

Un servidor indica su disposición de aceptar unas peticiones de conexión con las primitivas UNSPECIFIED PASSIVE OPEN y FULL_PASSIVE–OPEN. El término **pasivo** indica que el servidor está listo para recibir una petición de conexión en vez de (activamente) iniciar una conexión.

La primitiva UNSPECIFIED OPEN indica que está preparado a aceptar una petición de un proceso que cumpla con un nivel de seguridad específico (si está presente). Sin embargo, la otra primitiva, FULL_OPEN, incluye además una puerta de destino y un parámetro de dirección IP, el cual indica una aplicación específica de un proceso del cual se espera una respuesta. Normalmente un servidor está abierto en su primera apertura y queda abierto hasta que la máquina se apague.

En el lado del cliente, un usuario a través de un protocolo de aplicación, puede iniciar el establecimiento de una conexión de dos maneras:

En la primera alternativa, el protocolo del cliente simplemente emite una petición ACTIVE OPEN, especificando unas direcciones de puerta y dirección IP de destino. El TCP local responde devolviendo un OPEN ID, el cual informa el protocolo de aplicación y el identificativo –nombre de la conexión– el cual ha sido asignado a esta conexión (petición). Este intercambio inicia la conexión. Si es exitosa ambos usuarios empiezan a intercambiar mensajes usando las primitivas SEND y DELIVER.

En la segunda, el cliente envía una primitiva de petición ACTIVE OPEN WITH DATA, la cual contiene los datos de mensajes del usuario en un parámetro. El TCP local responde como antes retornando un nombre asignado a la conexión. Sin embargo, este inicia la conexión concurrentemente con la transferencia del mensaje recibido. Este modo es muy útil en el intercambio de mensajes cortos y únicos, debido a que las sobrecabeceras asociadas con la llamada se reducen.

Los datos se transfieren en cada dirección usando las primitivas de SENDER / DELIVER junto con los *flags* de *push* y *urgent*, como sea requerido.

La primitiva ALLOCATE la emplea un usuario para incrementar la cantidad de *buffer* de almacenamiento para su conexión en el lado local.

Por otro lado, el usuario en la fase de transferencia de datos puede solicitar el estado de la conexión utilizando la primitiva STATUS. El TCP local responde con la primitiva STATUS RESPONSE. Si se presenta una condición de error –por ejemplo cuando dos entidades TCP se desincronizan– se informa al usuario con la primitiva de ERROR.

Finalmente la conexión puede liberarse de dos maneras. Debido a que el flujo de mensaje de datos en cada dirección se controla por separado, es usual liberar la conexión en cada dirección separadamente. Esto se conoce como **desconexión armoniosa** (*graceful disconnection*) y requiere que ambos usuarios emitan una primitiva de petición de liberación separada de CLOSE después de que todos los datos han sido remitidos al TCP local. Un modo alternativo de usuario consiste en emitir una primitiva de ABORT que obligue a que ambos lados descarten todos los datos pendientes y terminen la conexión.

20.4 OPERACIÓN DE PROTOCOLO

El protocolo TCP incorpora muchas de las características del protocolo HDLC explicado anteriormente. Soporta transmisión *duplex* de mensajes e incorpora un procedimiento de control de ventana deslizante.

Todas las unidades de datos que utiliza el protocolo TCP/IP para establecer una conexión, transferir los datos y liberar una conexión tienen un formato normalizado conocido como **segmento**. Estos segmentos se transfieren entre entidades TCP en el campo de datos de usuario de los datagramas IP. Su formato se muestra en la figura 20.8.

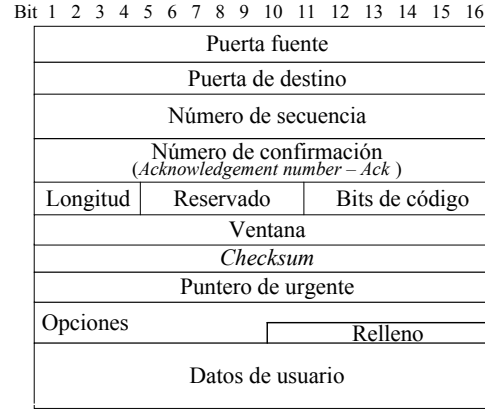

Figura 20.8 Formato del segmento (PDU) del protocolo TCP

20.4.1 DESCRIPCIÓN DE LOS CAMPOS DEL SEGMENTO

Seguidamente describimos los campos del formato del segmento.

- **Dirección de puerta de origen** y la **dirección de puerta de destino**: Éstos tienen el mismo significado que en el datagrama UDP, excepto que en el TCP estas direcciones indican los puntos extremos de las conexiones lógicas entre dos protocolos de aplicación.
- **Número de secuencia** y **número de confirmación**: Conceptualmente, se añade un número de secuencia a cada octeto que ha de transmitirse. El número de secuencia del primer octeto que transmite un segmento es el número de secuencia de segmento. Cuando un receptor envía un segmento, se incluye un número de confirmación, que es el número de secuencia del próximo octeto que se está esperando. La operación de estos números se aprecia en la fase de transferencia que veremos más adelante. Entonces, los números de secuencia y de confirmación están relacionados a la posición de un octeto en el tren completo del mensaje, en vez de la posición de un bloque de mensaje en la secuencia. Así, el número de secuencia indica la posición en el primer octeto del campo de datos del segmento relativo al inicio del mensaje completo. La presencia del campo de **opciones** en la cabecera del segmento significa que el segmento puede (en teoría) ser de longitud variable.
- **Longitud del encabezamiento**: Número de palabras de 32 bits que tiene el encabezamiento.
- **Reservado**: Son 6 bits que se reservan para un futuro uso.
- **Campo de códigos**: Este campo, cuya descripción se presenta en la tabla 20.2, tiene 6 bits. Todos los segmentos (y de aquí los tipos de PDU) tienen el mismo formato de cabecera y la validez de campo seleccionados en la cabecera del segmento está indicada por la colocación de bits en el campo de códigos (*code fields*). Si el bit se pone igual a 1, el campo correspondiente es válido. Nótese que pueden ponerse varios bits de código en un solo segmento.

POSIC. DE BIT	NOMBRE – FUNCIÓN
11	URG = Campo del puntero urgente es válido
12	ACK = Campo de confirmación (*acknowledgement*) válido
13	PSH = Transmitir de inmediato al recibir este segmento
14	RST = Usado para reiniciar un circuito virtual debido a un error no recuperable
15	SYN = Sincronizar el número de secuencia al número indicado en el campo de número de secuencia
16	FIN = Sirve para indicar la finalización de una comunicación

Tabla 20.2 Definiciones de los bits de código de protocolo TCP

A continuación, pasamos a describir cada bit de código.

- **Campo de ventana** (***window field***): El receptor utiliza este campo para implementar el control de flujo. El TCP receptor reporta un ventana hacia el TCP transmisor. Dicha ventana informa el número de octetos –empezando por el número de confirmación– que el TCP receptor está actualmente preparado para recibir. Esto está determinado por la cantidad de memoria de *buffer* que la puerta de origen tenga asignada para esta conexión.
- ***Checksum***: Es el complemento de la suma de todas las palabras de 16 bits en el segmento, empleando una aritmética de complemento de 1. Este cálculo se realiza sobre una seudocabecera conformada por las direcciones IP de fuente y destino, el identificativo de protocolo (*ID protocol* = 6 para el TCP) y la longitud del TCP. La figura 20.9 ilustra esta seudocabecera.
- **Campo de datos**: El número máximo de octetos por defecto en el campo de datos de un segmento es de 536 octetos. Este valor se elige asumiendo que las redes WAN están presentes en la ruta, en gene-

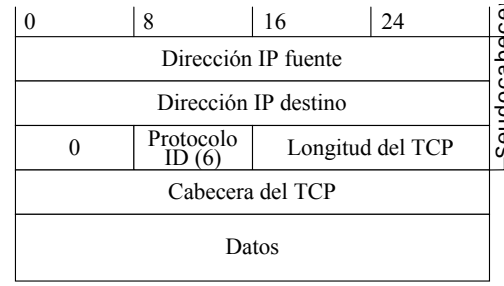

Figura 20.9 Campos seleccionados para confeccionar el control de errores

ral y que tienen una menor probabilidad de errores. Cuando dos protocolos de aplicación que se están comunicando corren en computadores, ambos conectados a una red con una tasa de errores baja (por ejemplo, una red LAN) se puede usar un segmento de mayor tamaño. Aquí el campo de **opciones** se usa para permitir al TCP transmisor indicar al receptor el número máximo de octetos en el campo de datos de un segmento que se va a transmitir.

20.4.2 ESTABLECIMIENTO DE CONEXIÓN

Aunque en la interacción cliente-servidor, el cliente siempre inicia colocando la llamada, en aplicaciones más generales, no basadas en el modelo cliente servidor, es posible para ambas partes intentar colocar una conexión al mismo tiempo. Para esto, se establece la conexión usando un intercambio de mensajes en tres sentidos. Esto se conoce como procedimiento *three-way-handshake*.

El flujo de datos en cada dirección de una conexión se controla independientemente para evitar toda ambigüedad en la secuencia inicial de números de ambos lados de la conexión; es decir cada lado informa al otro el número inicial de secuencia que va a usar. Este hecho es confirmado como parte del procedimiento de *handshake*. En la figura 20.10 mostramos dos ejemplos de intercambios de segmentos.

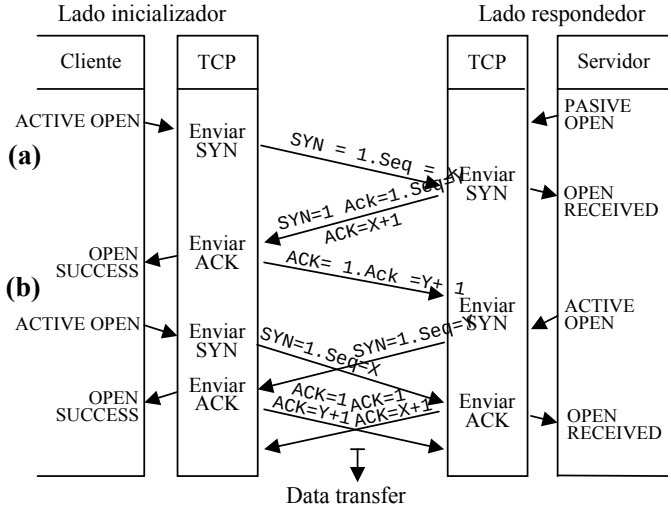

Figura 20.10 Establecimiento de conexión del TCP
(a) Cliente-servidor (b) Posibilidad de colisión

Una conexión se establece por el lado inicial enviando un segmento con el *flag* del bit de SYN colocado igual a 1, y el número de secuencia inicial propuesto en el campo del número de secuencia (SEQ = x).

En el lado de recepción, el lado respondedor primero toma una nota del número de secuencia de la dirección entrante. Entonces, retorna un segmento con los *flags* de SYN y ACK puestos con el número de campo de secuencia colocado a su propio valor asignado para la dirección inversa (SEQ = Y) y el campo de confirmación con X + 1 (ACK = X + 1) para confirmar que se ha tomado en cuenta el valor inicial para esta dirección entrante.

Habiendo recibido esta información, el lado inicial toma nota del valor de Y, y retorna un segmento con el *flag* de ACK = 1 y el campo de confirmación ACK = Y + 1. En caso que ambos lados envíen el segmento SYN al mismo tiempo –como se aprecia en la figura 20.10 (b)– cada lado simplemente retorna un segmento de confirmación ACK con el número de secuencia apropiado. Así se habilitan ambos lados de la conexión y pueden empezar a enviar datos inmediatamente.

20.4.3 TRANSFERENCIA DE DATOS

Como se indicó, los procedimientos de control de errores y de flujos asociados con la fase de transferencia de datos se basan en la ventana corrediza. Ya que este procedimiento se ha desarrollado en detalle al tratar el protocolo HDLC no lo volveremos a repetir. Para ilustrar el uso de un *flag* o bandera de PUSH, un ejemplo de un típico intercambio de segmentos relacionados a un intercambio de mensajes petición-respuesta se muestra en la figura 20.11.

En este ejemplo se asume que el cliente emite un corto mensaje de N octetos y para forzar la entrega, coloca el parámetro de PUSH. El TCP transmisor envía estos octetos directamente en un segmento con los *flags* SYN y PUSH puestos igual a 1 y el número de secuencia en su valor actual, asumiendo ser X.

En el lado de recepción del segmento, el lado receptor detecta el *flag* de PUSH y libera a éste directamente y retorna un segmento como el *flag* de ACK = 1 y el número de confirmación = X + N + 1, indicando que es el próximo octeto que está esperando recibir, con un valor de ventana igual a N para retornar la ventana a su programación o valor original.

Entonces asumimos que el servidor devuelve un mensaje de respuesta cuyo tamaño equivale a tres veces el número máximo W de segmentos. El TCP local debe entonces transmitir el mensaje en tres segmentos.

Para ilustrar el mecanismo de ventana corrediza, se asume que el tamaño de ventana es equivalente a 2W, es decir suficiente como para que sean enviados como máximo dos segmentos sin esperar una confirmación. Entonces, después de enviar dos segmentos, el servidor debe esperar hasta recibir una confirmación para ampliar su campo de ventana.

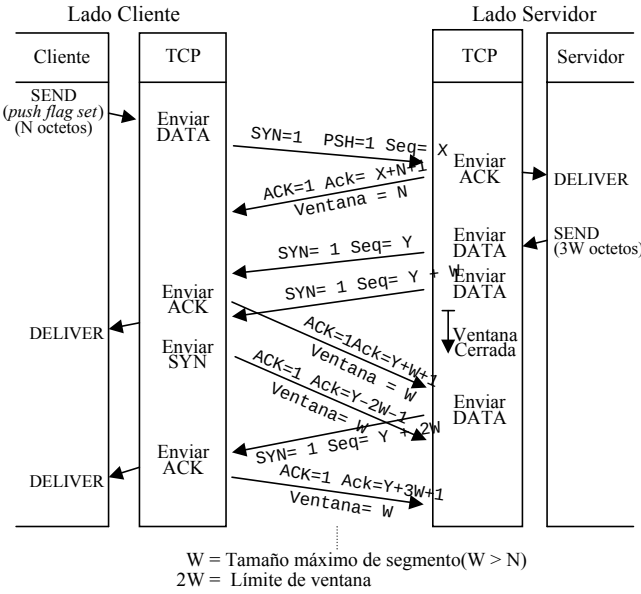

Figura 20.11 Ejemplo de transferencia de datos en el TCP

Una vez recibido cada segmento, el cliente TCP retorna un segmento con el *flag* de ACK puesto igual a 1. Al recibir el segundo segmento, éste entrega a la aplicación del cliente el contenido de su *buffer*, que contiene dos segmentos. En el servidor, una vez recibido el primer segmento de ACK, el servidor TCP incrementa su valor de crédito y procede a enviar el tercer segmento.

En el ejemplo se asume que este tercer segmento se entrega a la aplicación del cliente por el TCP receptor directamente. Aunque no se muestra, si se recibe un segmento con una secuencia o campo de confirmación incorrectos, entonces se transmite un segmento con los *flags* de RESET, SYN y ACK puestos en 1 y los valores esperados en los campos apropiados.

20.4.4 TERMINACIÓN DE LA CONEXIÓN

Las acciones en respuesta de cada entidad TCP a los dos métodos alternativos de terminación de conexión se dan en la figura 20.9. La primera es la desconexión armoniosa y la segunda la secuencia de aborto.

En el primer ejemplo se asume que se desea terminar la conexión. De aquí, habiendo recibido la primitiva de CLOSE, este envía un segmento con el *flag* de SYN= 1. Al recibir este segmento, el servidor emite una primitiva de CLOSING al protocolo del servidor y retorna un segmento de ACK al cliente confirmando la recepción del SYN = 1.

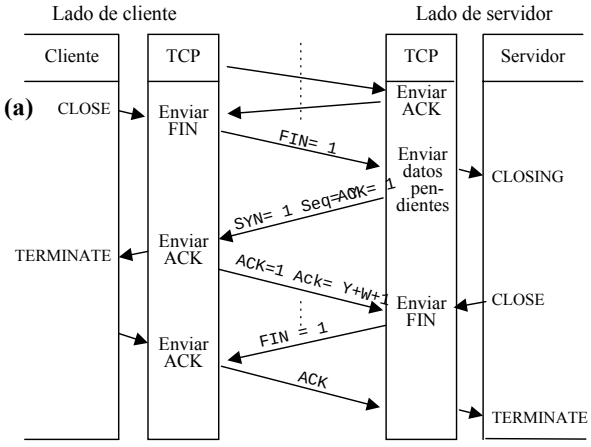

Figura 20.12 Terminación armoniosa de una conexión TCP

Se asume que el protocolo del servidor también terminó de transmitir sus datos por lo cual emite la primitiva de CLOSE en respuesta a la primitiva de CLOSING. Sin embargo, en el ejemplo se indica que el servidor TCP aún tiene datos pendientes para enviar, por lo cual este los transmite en un segmento junto con los *flags* de SYN y FIN colocados en 1.

Al recibir este segmento el cliente TCP emite una primitiva TERMINATE y retorna una confirmación ACK para confirmar los datos recibidos recientemente. Cuando el servidor TCP recibe el ACK transmite su propio TERMINATE al protocolo servidor. Alternativamente, si no se recibe un ACK dentro de un tiempo igual al doble del periodo del tiempo de vida, se asume que el ACK ha estado corrompido y envía una primitiva TERMINATE al servidor.

En el caso de una secuencia de aborto, el lado del cliente termina inmediatamente ambos lados de la conexión y envía un segmento con el *flag* RST = 1. Al recibir este segmento el servidor termina ambos lados de la conexión abruptamente y emite una primitiva TERMINATE con el código del motivo por el cual ha abortado la conexión.