- Capa de Transporte
  - Servicios que se proporcionan a las capas superiores
    - Primitias del servicio de transporte
    - Sockets de Berkeley
      - Resumen del funcionamiento:
    - Liberacion de una conexion
      - Liberación Asimétrica
      - Liberación Simétrica
      - Solución Propuesta: Acuerdo de Tres Vías
    - Control de errores y almacenamiento en bufer
      - 1. Control de errores
      - 2. Control de flujo
      - Mecanismos de control de flujo y errores en el transporte:
      - Diferencia entre las capas de enlace y transporte:
      - Manejo de los búferes:
      - Resumen final
    - Multiplexion
      - Ejemplo:
      - Cómo se logra la multiplexión:
  - Control de Congestion
  - UDP
    - Encabezado de UDP
    - Llamada a Procedimiento Remoto (RPC)
    - RTP (Protocolo de Transporte en Tiempo Real)
    - RTCP (Protocolo de Control de Transporte en Tiempo Real)
  - TCP
    - 🔷 Características Claves de TCP:
  - 📌 Modelo de Servicio TCP
    - 🔷 Sockets y Puertos
      - ** Límites del Tamaño del Segmento**
      - 4. Control de Flujo con Ventana Deslizante
      - 5. Problemas y Optimización en TCP
    - Encabezado TCP
    - Establecimiento de una Conexión TCP
    - Pasos del Proceso de Conexión
    - Caso Especial: Conexión Simultánea

Capa de Transporte

La capa de red provee una entrega de paquete punto a punto mediante el uso de datagramas o circuitos virtuales. La capa de trasnporte se basa en la capa de red para proveer transporte de datos de un proveso en una maquina de origen a un proceso en una maquina de destino, con un nivel deseado de confiabilidad que es independiente de las redes fisicas que se utilizan en la actualidad. Ofrece las abstracciones que necesitan las aplicaciones para usar la red. Sin esta capa, todo el concepto de protocolos por capas tendría muy poco sentido.

Servicios que se proporcionan a las capas superiores

La capa de transporte se encarga de proporcionar un servicio de transmision eficiente , confiable y economico. Existen dos tipos de servicio en la capa de transporte: orientado a conexion (como TCP) y sin conexion (como UDP).

La `capa de transporte` es esencial porque proporciona un servicio más confiable que la capa de red, la cual puede estar fuera del control de los usuarios (los enrutadores son operados por proveedores de red). Esta capa también aísla las

aplicaciones de las imperfecciones y variabilidad de las redes, lo que permite que los programas funcionen en diversas redes sin preocuparse por las diferencias en las interfaces de red o el nivel de fiabilidad.

Finalmente, se resalta la distinción entre las capas inferiores (1-4), que son proveedoras del servicio de transporte, y las capas superiores (por encima de la capa 4), que son las que usan este servicio. La capa de transporte actúa como **límite principal** entre el proveedor de servicio (la red) y el usuario del servicio (las aplicaciones).

Primitias del servicio de transporte

`Primitivas del servicio de transporte` son operaciones que permiten a los programas de aplicacion acceder al servicio de transporte de la capa correspondiente. Estas primitivas son esenciales para establecer, usar y liberar conexiones de manera confiable entre procesos, ocultando las imperfecciones de la red.

Se introducen las **primitivas básicas de un servicio de transporte orientado a conexión** , que incluyen:

1. **LISTEN** : El servidor espera a que un cliente se conecte.
2. **CONNECT** : El cliente intenta establecer una conexión con el servidor.
3. **SEND** : Envía datos a través de la conexión.
4. **RECEIVE** : Recibe datos de la conexión.
5. **DISCONNECT** : Libera la conexión cuando ya no es necesaria

El **servicio de transporte** es una capa crítica que ofrece a las aplicaciones un medio para intercambiar datos de manera confiable, aunque la red subyacente (capa de red) pueda ser inestable o no confiable. A diferencia del servicio de red, que a menudo está expuesto a la pérdida de paquetes o fallos en los enrutadores, la capa de transporte asegura que los datos lleguen correctamente al destino, gestionando retransmisiones y reconexiones si es necesario.

Así, los segmentos (intercambiados por la capa de transporte) están contenidos en paquetes (intercambiados por la capa de red). A su vez, estos paquetes están contenidos en tramas (intercambiadas por la capa de enlace de datos). Cuando llega una trama, la capa de enlace de datos procesa el encabezado de la trama y, si la dirección de destino coincide para la entrega local, pasa el contenido del campo de carga útil de la trama a la entidad de red. Esta última procesa de manera similar el

encabezado del paquete y después pasa el contenido de la carga útil del paquete a la entidad de transporte

Sockets de Berkeley

Otro conjunto de primitivas de trasnporte : las primitivas de socket que se utilizan para TCP.

un `socket` es un punto final de comunicacion en una red. La API de sockets proporciona un conjunto de primitivas para crear, gestionar y cerrar conexiones de red.

Las principales primitivas de socket para TCP son:

- **SOCKET** : Crea un nuevo socket.
- **BIND** : Asocia una dirección (IP y puerto) al socket.
- **LISTEN** : Marca el socket como listo para aceptar conexiones.
- **ACCEPT** : Acepta conexiones entrantes.
- **CONNECT** : Inicia una conexión activa con un servidor.
- **SEND** / **RECEIVE** : Envía y recibe datos.
- **CLOSE** : Cierra la conexión.

> 💡 **NOTA**: cualquier numero entre 1024 y 65535 funcionara como puerto; los puertos por debajo de 1023 estan reservados para los usuarios privilegiados
>
> Pro ejemplo:
>
> - **80** : HTTP (navegación web)
> - **443** : HTTPS (navegación web segura)
> - **21** : FTP (protocolo de transferencia de archivos)
> - **25** : SMTP (envío de correo)
>
> `Puerto` se usa para identificar un servicio o aplicacion especfica en una maquina dentro de una red. Los puertos permiten que diferentes aplicaciones en una misma maquina se comuniquen con otras maquina a traves de una reed, sin que haya conflicto entre ellas.

> 💡 **Siguiente es un posible escenario para una conexion de transporte:**
>
> 1. Un proceso servidor de correo se enlaza con el TSAP(ptunto termjnal que identifica un proceso de app en la capa de transporte) 1522 en el host 2 para esperar una llamada entrante. La manera en que un proceso se enlaza

> con un TSAP está fuera del modelo de red y depende por completo del sistema operativo local. Por ejemplo, se podría usar una llamada como nuestra LISTEN.
>
> - Un **proceso de aplicación en un cliente** se enlaza con otro TSAP (puerto) y solicita una conexión a un **servidor de correo** especificando la dirección de destino (TSAP).
> - Después de establecer la conexión, los datos (como el mensaje de correo) se transmiten entre ambos procesos.

El desafio es como un proceso de app en un host (por ejemplo,un cliente) sabe la direccion TSAP de un servicio :

- Para ello se establecen direcciones TSAP estables, donde los servidores conocidosse asignan a puertos fijos
- Para servicios desconocidos se utiliza un asignador de puertos. El asignador de puertos actúa como un **"directorio"** que devuelve la dirección correcta del TSAP asociado a un servicio solicitado.
- En lugar de tener cada servicio escuchando en su propio TSAP durante todo el día, se utiliza un servidor especial como **inetd** (en sistemas UNIX).

Resumen del funcionamiento:

1. El **cliente** envía una solicitud al asignador de puertos (portmapper) para obtener la dirección TSAP del servicio que busca.
2. El asignador de puertos devuelve la dirección TSAP del servicio solicitado.
3. El cliente establece una conexión con el servidor especificado a través del TSAP proporcionado.
4. El servidor gestiona la conexión para procesar la solicitud del cliente.

Liberacion de una conexion

el proceso de liberación de una conexión en redes de comunicaciones, específicamente en protocolos de transporte, como TCP, y plantea dos formas principales de liberación: **asimétrica** y **simétrica** .

Liberación Asimétrica

- En este enfoque, una de las partes (el host 2, por ejemplo) puede interrumpir la conexión en cualquier momento, lo que podría llevar a una desconexión abrupta.

Liberación Simétrica

- La liberación simétrica trata la conexión como dos canales unidireccionales, donde cada parte debe liberar su conexión de forma independiente. Esto significa que ambos hosts deben estar de acuerdo en que la conexión se ha completado antes de liberarse completamente.

Sin embargo, al implementar protocolos simétricos, existe un problema conocido como el **"Problema de los Dos Ejércitos"** : En la liberación de una conexión, ambos lados deben estar de acuerdo en que la conexión debe terminarse. Si una parte no está segura de si la otra ha recibido la solicitud de desconexión (por ejemplo, un mensaje `DISCONNECT`), nunca se desconectará, creando un "deadlock" o impidiendo que se libere la conexión.

Solución Propuesta: Acuerdo de Tres Vías

- **Host 1** envía una solicitud de desconexión (`DISCONNECT REQUEST`).
- **Host 2** responde con su propio mensaje de solicitud de desconexión, y ambos inician un temporizador por si alguno de los mensajes se pierde.
- Cuando **Host 1** recibe el mensaje de **Host 2** , envía un mensaje de confirmación (`ACK`), y la conexión se libera.
- Finalmente, **Host 2** recibe el `ACK` y también libera la conexión.

Liberar una conexión correctamente y sin pérdida de datos es más complejo de lo que parece. Aunque protocolos como TCP implementan un cierre simétrico, en ocasiones (como en servidores web) se utiliza una **desconexión asimétrica** para hacerla más rápida, confiando en que el cliente detectará la desconexión y liberará su estado de conexión cuando sea necesario.

En resumen, la **liberación simétrica** es más segura y confiable para evitar la pérdida de datos, pero requiere una coordinación cuidadosa entre las partes para asegurar que ambas estén de acuerdo en cuándo liberar la conexión.

Control de errores y almacenamiento en bufer

El control de errores y el control de flujo en los protocolos de transporte:

1. Control de errores

- **Objetivo:** Garantizar que los datos se entreguen correctamente, sin errores.

- **Método:** Se utiliza un **código de detección de errores** (por ejemplo, CRC o suma de verificación) en las tramas para verificar que los datos se hayan recibido correctamente.

2. Control de flujo

- **Objetivo:** Evitar que un transmisor rápido sobrecargue a un receptor lento.
- **Método:**
  - El **emisor** no puede enviar más datos de los que el **receptor** puede manejar en un momento dado.
  - En los protocolos de transporte, el control de flujo se maneja mediante **ventanas deslizantes** , donde el tamaño de la ventana limita la cantidad de datos que pueden estar pendientes de confirmación.

Mecanismos de control de flujo y errores en el transporte:

1. **Detección de errores:** Cada trama tiene un código de verificación (CRC o suma de verificación) para comprobar la integridad de los datos durante la transmisión.
2. **Retransmisión (ARQ):** Si el receptor no recibe correctamente una trama, el emisor retransmite la trama hasta recibir una confirmación de recepción exitosa.
3. **Ventanas deslizantes:**
   - Hay un límite en el número de tramas (o segmentos) pendientes de confirmación que el emisor puede enviar.
   - Protocolos como **parada y espera** (donde solo se permite un paquete pendiente a la vez) o **ventanas deslizantes** más grandes (que permiten transmitir varios segmentos a la vez) se usan dependiendo de la velocidad y características del enlace.
   - Un **tamaño de ventana mayor** mejora el rendimiento en enlaces rápidos y de mayor latencia.

Diferencia entre las capas de enlace y transporte:

- **Capa de enlace:** Las sumas de verificación funcionan solo dentro de un enlace, pero no protegen los datos a través de múltiples enlaces (como dentro de un enrutador).
- **Capa de transporte:** La suma de verificación de la capa de transporte asegura la integridad de los datos a través de toda la red (punto a punto).

Manejo de los búferes:

- **Búferes en el emisor:** Se utilizan para almacenar los segmentos transmitidos que aún no han recibido confirmación de recepción. Estos segmentos pueden perderse y necesitan retransmitirse.
- **Búferes en el receptor:** El receptor puede usar un único conjunto de búferes compartido o asignar búferes por cada conexión. Si no hay espacio suficiente en los búferes, el receptor puede desechar segmentos.
- **Asignación dinámica de búferes:**
  - La asignación de búferes puede variar dependiendo de las necesidades del tráfico y la capacidad de almacenamiento en el receptor.
  - El **emisor** puede solicitar más búferes si es necesario, y el **receptor** puede asignar estos búferes según su disponibilidad.
  - TCP utiliza una técnica de ventana dinámica para ajustar la cantidad de búferes que puede usar un emisor en función de la capacidad de recepción y el tráfico.

Resumen final

- Los protocolos de transporte como **TCP** utilizan **ventanas deslizantes** para controlar el flujo de datos y evitar la congestión.
- El control de **errores** y **flujo** se asegura mediante técnicas como la detección de errores, la retransmisión automática (ARQ), y el ajuste dinámico de la ventana deslizante.
- El uso de **búferes** tanto en el emisor como en el receptor es fundamental para manejar los datos de manera eficiente y evitar la pérdida de información.
- En redes de alta latencia o baja capacidad, el tamaño de la ventana debe ajustarse para maximizar el rendimiento sin sobrecargar la red o los dispositivos de almacenamiento en los hosts.

Multiplexion

La **multiplexión** es un proceso mediante el cual se permite que múltiples flujos de datos o conversaciones compartan un mismo canal de comunicación, como un enlace físico o una dirección de red, para mejorar la eficiencia y aprovechar mejor los recursos disponibles. En el contexto de las redes, la multiplexión se puede aplicar de diferentes formas dependiendo de la capa del modelo OSI que esté involucrada.

La multiplexión es la solución que permite que **múltiples aplicaciones** que corren en un mismo host **compartan una sola dirección IP** sin interferir entre ellas. Para esto, la multiplexión se apoya en un concepto clave: **puertos de red** .

Ejemplo:

- Supón que tienes las siguientes aplicaciones en tu host (servidor):
  - **Aplicación A** (por ejemplo, un servidor web)
  - **Aplicación B** (por ejemplo, un servidor FTP)
  - **Aplicación C** (por ejemplo, un servidor de correo)

Todas estas aplicaciones se ejecutan en el mismo host con la dirección IP **192.168.1.1** , pero necesitan usar puertos diferentes para que sus datos no se mezclen. Los **puertos** funcionan como "canales" para separar las conexiones de las distintas aplicaciones.

- **Aplicación A (servidor web)** usa el puerto **80** (HTTP).
- **Aplicación B (servidor FTP)** usa el puerto **21** (FTP).
- **Aplicación C (servidor de correo)** usa el puerto **25** (SMTP).

Cómo se logra la multiplexión:

Cuando un paquete llega al host **192.168.1.1** , el sistema operativo del host revisa la **dirección IP** (que es la misma para todas las aplicaciones) y el **número de puerto** que lleva el paquete. El número de puerto permite que el sistema operativo sepa a qué aplicación entregar el paquete.

- Si el paquete tiene como destino el **puerto 80** , se enviará al proceso que gestiona la **Aplicación A (servidor web)** .
- Si el paquete tiene como destino el **puerto 21** , se enviará al proceso que gestiona la **Aplicación B (servidor FTP)** .
- Si el paquete tiene como destino el **puerto 25** , se enviará al proceso que gestiona la **Aplicación C (servidor de correo)** .

Control de Congestion

UDP

Internet tiene dos protocolos principales en la capa de transporte: uno sin conexion (`UDP`) y otro orientado a conexion(`TCP`).

`UDP` (*Use Datagrama Protocol*) no hace mas que enviar paquetes entre aplicaciones y deja que las aplicaciones construyan sus propios protocolos en la parte superior segun

sea necesario.

- El protocolo se describe en el RFC 768
- no establece una conexión antes de enviar datos y no realiza mecanismos de control de flujo, control de congestión ni retransmisiones, todo esto le corresponde a los procesos de usuario.
- Lo que sí realiza es proporcionar una interfaz para el protocolo IP con la característica agregada de demultiplexar varios procesos mediante el uso de los puertos y la detección de errores extremo a extremo opcional.
- Un área en la que UDP es especialmente útil es en las situaciones cliente-servidor. Con frecuencia, el cliente envía una solicitud corta al servidor y espera una respuesta corta. Si se pierde la solicitud o la respuesta, el cliente simplemente puede esperar a que expire su temporizador e intentar de nuevo. El código no sólo es simple, sino que se requieren menos mensajes (uno en cada dirección) en comparación con un protocolo que requiere una configuración inicial, como TCP.
- Una aplicación que utiliza de esta manera a UDP es DNS (el Sistema de Nombres de Dominio): En resumen, un programa que necesita buscar la dirección IP de algún host, por ejemplo, www.cs.berkeley.edu, puede enviar al servidor DNS un paquete UDP que contenga el nombre de dicho host. El servidor responde con un paquete UDP que contiene la dirección IP del host. No se necesita configuración por adelantado ni tampoco una liberación posterior. Sólo dos mensajes que viajan a través de la red.

Encabezado de UDP

El segmento de UDP consta de un encabezado de 8 bytes seguido de la carga util de datos. El encabezado incluye:

- **Puerto de origen y destino** : Identifican los procesos en los extremos de la comunicación (similar a apartados postales para las aplicaciones).
- **Longitud** : Especifica la longitud total del datagrama UDP, incluido el encabezado y los datos. La longitud mínima es de 8 bytes y la máxima es de 65,515 bytes.
- **Suma de verificación** : Proporciona una verificación opcional de la integridad de los datos, asegurando que no haya errores durante la transmisión. Este cálculo involucra el encabezado UDP, los datos y un **pseudoencabezado IP** que incluye las direcciones IP de origen y destino.El algoritmo de suma de verificacion consiste simplemente en sumar todas las palabras de 16 bits en complemento a uno y sacar el complemente a uno de la suma. Como consecuencia , cuando el

receptor realiza el calculo de todo el segmento (incluyendo del campo de al suma) , el resultado debe ser 0.

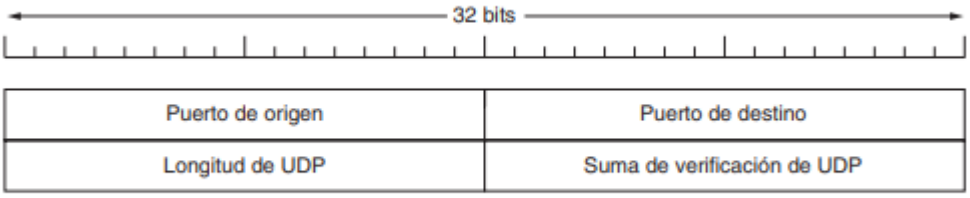

Figura 6-27. El encabezado UDP.

Llamada a Procedimiento Remoto (RPC)

Las **Llamadas a Procedimiento Remoto (RPC)** permiten que un programa en una máquina (cliente) invoque un procedimiento en otra máquina (servidor), ocultando la complejidad de la comunicación de red. Conceptualmente, esto es similar a llamar a una función local, pero el procedimiento se ejecuta en una máquina remota.

RTP (Protocolo de Transporte en Tiempo Real)

- Se describe en el RFC 3550
- Protocolo de capa de `Aplicacion` que usa el protocolo UDP como protocolo de transporte
- Se utiliza para aplicaciones multimedias
- La funcion basica es multiplexar varios flujos de datos de timepo real en un flujo de paquetes UDP
- No hay garantías especiales acerca de la entrega, así que los paquetes se pueden perder, retrasar, corromper, etcétera
- **Numeración de paquetes:** Cada paquete lleva un número de secuencia, lo que permite detectar pérdidas y tomar medidas, como interpolar audio o descartar una imagen de video.

> 💡 **Ejemplo de uso:**
>
> - Un video en vivo se transmite con RTP sobre UDP.
> - RTP numera los paquetes y agrega marcas de tiempo.
> - UDP los envía sin garantía de entrega, pero RTP permite que el receptor reproduzca el video sin interrupciones perceptibles.

RTCP (Protocolo de Control de Transporte en Tiempo Real)

Complementa a RTP proporcionando retroalimentación sobre la calidad de la transmisión.

- **Monitorea la calidad de la red** , midiendo el retardo, la variación del retardo (**jitter**) y la congestión.
- **Ajusta la tasa de transmisión** , permitiendo que el emisor cambie el formato de codificación según el ancho de banda disponible.
- **Sincroniza múltiples flujos** , por ejemplo, en una transmisión de video con audio en varios idiomas.

RTP es el estándar clave para la transmisión de medios en tiempo real, permitiendo la entrega eficiente de audio y video sin garantizar la entrega de los paquetes. RTCP ayuda a controlar la calidad de la transmisión y ajustar los parámetros según las condiciones de la red. Para evitar problemas como el jitter, se utiliza almacenamiento en búfer para garantizar una reproducción fluida.

TCP

UDP es un protocolo simple y tiene algunos usos muy importantes, como las interacciones cliente-servidor y multimedia, pero para la mayoría de las aplicaciones de Internet se necesita una entrega en secuencia confiable. UDP no puede proporcionar esto, por lo que se requiere otro protocolo. Se llama TCP y es el más utilizado en Internet.

Hace casi todo . Realiza las conexiones y agrega confiabilidad mediante las retransmisiones, junto con el control de flujo y el control de congestión, todo en beneficio de las aplicaciones que lo utilizan.

> 🔷 **Características Claves de TCP:**
>
> ✅ **Fiabilidad:** TCP divide los datos en segmentos y garantiza que lleguen completos y en orden. ✅ **Control de congestión:** Ajusta dinámicamente la velocidad de envío para evitar saturar la red. ✅ **Reensamblado de datos:** Si los paquetes llegan desordenados, TCP los reorganiza antes de entregarlos a la aplicación. ✅ **Detección de errores:** Usa mecanismos como el **checksum** para verificar que los datos no estén corruptos. ✅ **Conexión orientada:** TCP requiere establecer una conexión antes de enviar datos, mediante un proceso llamado **Three-Way Handshake** .

📌 Modelo de Servicio TCP

Para que dos dispositivos intercambien datos con TCP, deben **crear sockets** .
Un **socket** es una combinación de una **dirección IP** y un **número de puerto** .

🔷 Sockets y Puertos

- Cada aplicación en un dispositivo usa un puerto específico para comunicarse.
- Existen **puertos bien conocidos (1-1023)** para servicios estándar:
  - **HTTP (80), HTTPS (443), FTP (21), SSH (22), SMTP (25)** , etc.
- Otros puertos (1024-49151) pueden ser registrados para aplicaciones específicas.

TCP asigna a **cada byte** de una conexión un número de secuencia único de **32 bits** . Esto permite un control preciso del flujo de datos y garantiza que los paquetes lleguen en el orden correcto.. UDP en cambios , solo se almacena en datagramas justamente por carecer de control de

Un **segmento TCP** es la unidad de datos que TCP intercambia entre el emisor y el receptor.

- **Encabezado de 20 bytes** (puede incluir opciones adicionales).
- **Datos** : Cero o más bytes, según el tamaño del segmento.
- TCP **decide** cuántos datos incluir en cada segmento, basándose en la eficiencia y los límites de la red.

** Límites del Tamaño del Segmento**

- **Máximo teórico** : 65,515 bytes (limitado por IP).
- **MTU (Maximum Transfer Unit)** : La **MTU** es el tamaño máximo de datos que un paquete puede transportar en una única transmisión sin fragmentarse, medido en **bytes** . Generalmente **1500 bytes** en redes Ethernet. En Wi-Fi (802.11) son 2304 bytes y en IPv6 son 1280 bytes.
- **Descubrimiento de MTU** : TCP ajusta dinámicamente el tamaño de los segmentos para evitar **fragmentación** , usando mensajes ICMP (RFC 1191).

4. Control de Flujo con Ventana Deslizante

TCP usa un **protocolo de ventana deslizante** con **tamaño dinámico** , lo que significa que:

1. **El emisor envía un segmento y activa un temporizador** .
2. **El receptor envía una confirmación de recepción (ACK)** , indicando qué datos ha recibido y cuánta capacidad queda en la ventana.
3. **Si no llega la confirmación a tiempo, TCP retransmite el segmento** .

5. Problemas y Optimización en TCP

- **Retransmisiones y Orden de Segmentos** :
- Los segmentos pueden **llegar fuera de orden** (Ej.: el receptor recibe bytes 3,072-4,095 antes de 2,048-3,071).
- TCP debe **manejar estas situaciones sin perder datos** .
- **Temporización y Retransmisiones** :
- Si un segmento se **retrasa demasiado** , el emisor lo retransmite.
- Las retransmisiones pueden incluir **rangos de bytes diferentes** a los originales, lo que requiere una gestión cuidadosa.
- **Optimización del Rendimiento** :
- TCP ha sido mejorado con varios algoritmos para manejar problemas de red, minimizar retransmisiones y mejorar la eficiencia del tráfico.

Encabezado TCP

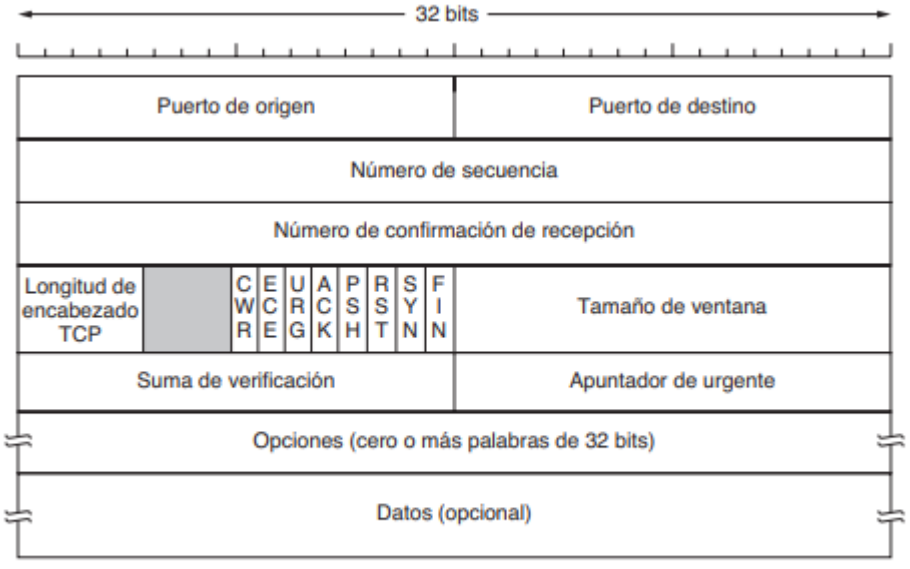

Figura 6-36. El encabezado TCP.

- EL encabezado tiene un tamaño fijo de **20 bytes** y puede incluir opciones adicionales.

- Después de las opciones, si las hay, pueden continuar hasta 65 535 − 20 − 20 = 65 495 bytes de datos, donde los primeros 20 se refieren al encabezado IP y los segundos al encabezado TCP.
- **Puertos de origen y destino** : Identifican los extremos de la conexión en cada host.Un puerto TCP más la dirección IP de su host forman un punto terminal único de 48 bits. Los puntos terminales de origen y de destino enconjunto identifican la conexión. Este identificador de conexión se denomina 5-tupla, ya que consiste en cinco piezas de información: el protocolo (TCP), IP de origen y puerto de origen, IP de destino y puerto de destino. EJ ((TCP, 192.168.1.100, 54321, 203.0.113.10, 80)) 16 bits y 16 biits
- **Número de secuencia** : Indica el orden de los bytes transmitidos. (32 bits)
- **Número de confirmación (ACK)** : Indica el siguiente byte que el receptor espera recibir. (32 bits)
- **Longitud del encabezado** : Especifica el tamaño del encabezado TCP (4 bits)
- **Banderas de control** : Bits individuales con funciones específicas, como:*
  - **URG** : Indica datos urgentes.
  - **ACK** : Valida el número de confirmación.
  - **PSH** : Solicita entrega inmediata de los datos
  - **RST** : Restablece la conexión.
  - **SYN** : Inicia la conexión.
  - **FIN** : Finaliza la conexión
- **Tamaño de ventana** : Controla el flujo de datos permitidos en un momento dado.
- **Suma de verificación** : Garantiza la integridad del segmento.
- **Opciones** : Extienden la funcionalidad de TCP, por ejemplo:
  - **MSS (Maximum Segment Size)** : Establece el tamaño máximo del segmento
  - **Escalado de ventana** : Permite ventanas más grandes en redes rápidas.
  - **SACK (Selective Acknowledgment)** : Mejora la eficiencia de retransmisiones.
  - **Estampas de tiempo** : Ayudan a calcular el tiempo de ida y vuelta.

Establecimiento de una Conexión TCP

El establecimiento de una conexión TCP se realiza mediante un proceso llamado **"acuerdo de tres vías" (three-way handshake)** , el cual garantiza una conexión confiable entre dos dispositivos antes de transferir datos.

> **Pasos del Proceso de Conexión**

1. **El servidor espera conexiones**
    - Se ejecutan las primitivas `LISTEN` y `ACCEPT`, indicando que el servidor está a la espera de una conexión en un puerto específico.
2. **El cliente inicia la conexión**
    - Ejecuta la primitiva `CONNECT`, enviando un **segmento TCP con el bit SYN activado** para solicitar la conexión.
3. **El servidor responde**
    - Si el servidor está escuchando en el puerto solicitado, responde con un **segmento SYN-ACK** , confirmando la recepción de la solicitud.
    - Si el servidor no está escuchando en el puerto, envía un segmento con el **bit RST activado** , rechazando la conexión.
4. **El cliente confirma la conexión**
    - Responde con un **segmento ACK** , finalizando el proceso de establecimiento de conexión.

📌 **Ejemplo de secuencia normal:**

- Cliente → Servidor: `SYN (SEQ = x)`
- Servidor → Cliente: `SYN-ACK (SEQ = y, ACK = x+1)`
- Cliente → Servidor: `ACK (SEQ = x+1, ACK = y+1)` ✅ **Conexión establecida, lista para la transferencia de datos.**

Caso Especial: Conexión Simultánea

Si **dos hosts intentan conectarse entre sí al mismo tiempo** , se genera un cruce de SYN y se evita la duplicación de conexiones al reconocer que ambas pertenecen a la misma **5-tupla** (protocolo, IPs y puertos).

📌 **Ejemplo de conexión simultánea:**

- Host 1 → Host 2: `SYN (SEQ = x)`
- Host 2 → Host 1: `SYN (SEQ = y, ACK = x+1)`
- Host 1 → Host 2: `ACK (SEQ = x+1, ACK = y+1)` ✅ **Se establece una única conexión.**

Seguridad: Ataque de Inundación SYN y Defensa con SYN Cookies

- Un atacante puede saturar un servidor enviando **múltiples paquetes SYN sin completar la conexión** , lo que consume recursos y puede dejar al servidor inoperativo.

- **Defensa: SYN Cookies**
  - En lugar de almacenar los números de secuencia, el servidor los genera con una función criptográfica y los verifica al recibir la respuesta.
  - Así, el servidor no necesita recordar conexiones incompletas, mitigando el ataque.

Liberación de una conexión TCP

En TCP, la liberación de una conexión se realiza de manera independiente en cada dirección, ya que TCP es **full-dúplex** (permite comunicación en ambos sentidos simultáneamente). Para cerrar la conexión, cualquiera de los extremos envía un **segmento TCP con el bit FIN activado** , indicando que no tiene más datos por transmitir.

Una vez que el otro extremo **confirma el FIN con un ACK** , la comunicación en esa dirección se **cierra** , pero el otro sentido puede seguir transmitiendo datos hasta que también envíe su propio **FIN** . Cuando ambos sentidos se cierran, la conexión se libera completamente.

> **Pasos en la liberación de una conexión TCP**
>
> Por lo general, se requieren **cuatro segmentos** para completar la liberación:
>
> 1. **Host A** → **Host B** : Envío de `FIN` (no tiene más datos que enviar).
> 2. **Host B** → **Host A** : Confirmación con `ACK`.
> 3. **Host B** → **Host A** : Envío de `FIN` (también termina su comunicación).
> 4. **Host A** → **Host B** : Confirmación con `ACK`.
>
> Para evitar que una conexión quede colgada si un `FIN` o `ACK` se pierde, TCP usa **temporizadores** . Si no recibe respuesta en un tiempo máximo (normalmente dos veces el tiempo de vida del paquete), el emisor del `FIN` asume que la conexión se cerró y la libera.
>
> 🔷 **Ejemplo cotidiano** : Es como una llamada telefónica en la que ambas personas dicen "adiós" y cuelgan el teléfono.

Explicación del Proceso

- **El cliente inicia la conexión** con `CONNECT`, enviando un `SYN`.
- **El servidor responde** con `SYN + ACK` y espera la confirmación del cliente.

- **La conexión se establece** cuando el cliente envía `ACK`, permitiendo el intercambio de datos.
- **Para cerrar la conexión** , uno de los extremos envía un `FIN`, que es reconocido con un `ACK`.
- **Si el otro extremo también cierra** , envía su propio `FIN`, y el cierre se confirma con otro `ACK`.
- **El estado** `TIME WAIT` asegura que todos los paquetes retrasados desaparezcan antes de cerrar completamente.

> 📌 **Conclusión:** TCP gestiona las conexiones mediante un conjunto de reglas bien definidas, asegurando una comunicación confiable y un cierre seguro de las conexiones.

Ventana Deslizante de TCP

La ventana deslizante en **TCP** es un mecanismo de control de flujo que permite que el receptor administre la cantidad de datos que el emisor puede enviar antes de recibir una confirmación. Separa dos aspectos clave:

1. **Confirmación de recepción:** TCP confirma la correcta recepción de los segmentos enviados.
2. **Asignación del búfer en el receptor:** Determina cuánto espacio tiene el receptor para recibir nuevos datos.

Funcionamiento de la Ventana Deslizante

- El receptor tiene un **búfer** de un tamaño determinado, por ejemplo, **4 KB** .
- El emisor envía un segmento de **2 KB** , que el receptor confirma y anuncia una **ventana de 2 KB** (porque aún tiene espacio disponible).
- Si el emisor envía otros **2 KB** , el búfer del receptor se llena y anuncia una **ventana de 0** , lo que detiene el envío de más datos hasta que la aplicación en el receptor procese y libere espacio.
- **Ventana deslizante:**
  - TCP permite que el emisor envíe varios segmentos sin esperar una confirmación por cada uno.
  - La ventana indica cuánto espacio hay disponible en el búfer del receptor.

- **Problemas que pueden afectar el rendimiento:**
    - **Síndrome de ventana tonta:** Cuando el receptor lee de a 1 byte y actualiza la ventana por solo 1 byte, el emisor envía segmentos demasiado pequeños, desperdiciando recursos. **Solución:** El receptor espera hasta tener suficiente espacio antes de actualizar la ventana.
    - **Retrasos en confirmaciones:** TCP puede retrasar las confirmaciones hasta 500 ms para agrupar varias en un solo mensaje, reduciendo el tráfico innecesario.
    - **Algoritmo de Nagle:** Si la aplicación envía datos en pedazos pequeños, TCP espera a agruparlos en un segmento más grande antes de enviarlos. Esto reduce el uso del ancho de banda pero puede causar retrasos en aplicaciones interactivas.
- **Estrategias de TCP para mitigar estos problemas:**
    - Usar **confirmaciones acumulativas** para reducir la cantidad de ACKs enviados.
    - Permitir deshabilitar el algoritmo de Nagle con **TCP_NODELAY** en casos donde la latencia sea prioritaria (por ejemplo, en juegos en línea).
    - Usar **sondas de ventana** cuando la ventana es 0 para evitar bloqueos en la conexión.

Control de congestion en TCP

La congestión ocurre cuando la cantidad de datos que se envía a través de la red es mayor que la capacidad que los enrutadores pueden manejar. Cuando esto sucede, los enrutadores comienzan a descartar paquetes, lo que afecta la eficiencia de la comunicación.

TCP usa una ventana de congestión (en paralelo a la ventana de control de flujo) para limitar la cantidad de datos que pueden estar en tránsito en la red. La cantidad efectiva de datos que se pueden enviar en un momento dado es el valor mínimo entre estas dos ventanas.

Que agregar ?

Limitaciones de la semántica de transporte de TCP:

- TCP no satisface todas las necesidades de las aplicaciones modernas.

- Algunas aplicaciones requieren preservar los límites de los mensajes, manejar múltiples conversaciones relacionadas o controlar mejor las rutas de red.
- Esto ha llevado a la creación de nuevos protocolos como **SCTP (Stream Control Transmission Protocol)** y **SST (Structured Stream Transport)** , que ofrecen mejoras en estos aspectos.
- Sin embargo, cualquier cambio en TCP genera resistencia, ya que es un protocolo probado y ampliamente utilizado.

Redes Tolerantes al Retardo (DTN)

Las **Redes Tolerantes al Retardo (DTN)** ()Delay-Tolerant Networking) surgen como una solución para entornos donde la conectividad es intermitente o los retardos en la comunicación son elevados, como en redes espaciales, submarinas, móviles y sistemas con infraestructura limitada. A diferencia de los protocolos tradicionales como **TCP** , que requieren una conexión estable de extremo a extremo, las DTN usan **conmutación de mensajes** , almacenando datos en nodos intermedios hasta que se establezca un enlace funcional para su reenvío.

Arquitectura DTN

- Introducida por la **IETF en el RFC 4838** .
- Se basa en el concepto de **almacenamiento-transporte-reenvío** en lugar de la conmutación de paquetes tradicional.
- Los datos viajan en **bundles** (paquetes almacenados en nodos DTN hasta que haya conexión disponible) mas grandes que los tradicioanles paquetes pequennos.
- Es útil en entornos donde los dispositivos solo se conectan ocasionalmente (por ejemplo, satélites, sensores, autobuses).
- Permite transferencias eficientes de grandes volúmenes de datos en horarios de bajo tráfico.

Ejemplo de Aplicación en Redes Espaciales

- Un **satélite LEO** captura imágenes de la Tierra.
- Como la conexión con estaciones terrestres es intermitente, los datos se almacenan hasta que haya un contacto disponible.
- Luego, se transfieren en fragmentos a diferentes estaciones terrestres y, finalmente, al destino final.

Protocolo Bundle (RFC 5050)

- Es el protocolo clave en DTN, permitiendo la transmisión de datos como bundles.
- Se sitúa sobre **TCP/IP u otras tecnologías** y puede operar con diversos protocolos de transporte.
- Requiere una **capa de convergencia** para adaptarse a diferentes tipos de redes.
- Se emplea en **redes espaciales, sensores y sistemas con alta latencia** .

Las **DTN** permiten la comunicación en escenarios donde las redes tradicionales fallan, facilitando el transporte de datos en entornos con conexiones intermitentes y retardos elevados.

Resumen

La **capa de transporte** es fundamental para comprender los protocolos en capas, ya que ofrece servicios esenciales, siendo el más importante el flujo de bytes confiable y orientado a conexión entre el emisor y el receptor. Se accede a esta capa a través de **primitivas de servicio** que permiten establecer, usar y liberar conexiones, con los **sockets de Berkeley** como una interfaz común.

Los protocolos de transporte deben manejar conexiones en redes no confiables, enfrentándose a desafíos como paquetes duplicados y retardados. Para establecer conexiones, se utiliza un **acuerdo de tres vías** , y aunque liberar una conexión es más sencillo, aún presenta complicaciones. La capa de transporte debe administrar primitivas de servicio, conexiones, temporizadores, asignación de ancho de banda con control de congestión y ejecutar una **ventana deslizante** para el control de flujo.

El control de congestión se encarga de asignar el ancho de banda de manera equitativa entre flujos competidores, utilizando la ley AIMD (Additive Increase/Multiplicative Decrease) para lograr una asignación justa. Los dos protocolos de transporte principales en Internet son **UDP** y **TCP** . UDP es un protocolo sin conexión que actúa como envoltura para paquetes IP, permitiendo multiplexión y desmultiplexión de procesos. TCP, por otro lado, es el protocolo de transporte principal, que proporciona un flujo de bytes confiable, bidireccional y controlado por congestión, con un encabezado de 20 bytes.

El rendimiento de TCP ha sido optimizado mediante varios algoritmos. Sin embargo, el rendimiento de las redes puede verse afectado por la sobrecarga de procesamiento de protocolos y segmentos, lo que se agrava a mayores velocidades. Por lo tanto, es

esencial diseñar protocolos que minimicen la cantidad de segmentos y el procesamiento requerido.

Finalmente, las **redes tolerantes al retardo (DTN)** ofrecen servicios de entrega en entornos con conectividad ocasional o retardos largos. En estas redes, los nodos intermedios almacenan, transportan y reenvían "bundles" de información, asegurando que se entregue en el momento adecuado, incluso sin una trayectoria funcional permanente entre el emisor y el receptor