ETSIIT
Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Ingeniería Informática + ADE

Universidad de Granada (UGR)

Autor: Ismael Sallami Moreno

Asignatura: Tema 3: Capa de transporte (FR)

Índice

1 Introducción

La capa de transporte es fundamental en el modelo de comunicaciones, proporcionando servicios que garantizan una comunicación efectiva y eficiente entre aplicaciones que se ejecutan en hosts finales. Sus funciones y servicios principales incluyen:

- **Comunicación extremo a extremo (end-to-end):** Garantiza la transmisión de datos entre procesos de aplicación situados en dispositivos distintos a través de una red compleja.
- **Multiplexación y demultiplexación de aplicaciones:** Utiliza puertos para asociar los datos con las aplicaciones correctas.

La capa de transporte incluye dos protocolos fundamentales:

Protocolo de datagrama de usuario (UDP)

- Permite la multiplexación y demultiplexación de aplicaciones[1]
- Proporciona un servicio no orientado a conexión, caracterizado por su simplicidad y ausencia de confiabilidad.

Protocolo de control de transmisión (TCP)

- Permite la multiplexación y demultiplexación de aplicaciones.
- Ofrece un servicio orientado a conexión y confiable, destacándose por:
  - Control de errores y de flujo.
  - Gestión de la conexión (establecimiento y cierre).
  - Control de congestión para optimizar el uso de la red.

Además, TCP ha evolucionado con extensiones que se adaptan a las necesidades de redes modernas, manteniendo la interoperabilidad entre los extremos y mejorando su rendimiento en diversos escenarios.

[1] En el contexto de la capa de transporte, la multiplexación se refiere al proceso mediante el cual múltiples flujos de datos de diferentes aplicaciones o procesos dentro de un host se combinan para ser transmitidos a través de una única conexión de red. Esto se realiza asignando identificadores únicos, como números de puerto, a cada flujo de datos, lo que permite a la capa de transporte diferenciarlos y enviarlos correctamente al destinatario correspondiente.

La demultiplexación es el proceso inverso: cuando los datos llegan al host receptor, la capa de transporte utiliza esos identificadores (puertos) para separar los flujos de datos y entregarlos a las aplicaciones o procesos adecuados. Esto asegura que la información llegue al lugar correcto, incluso si varios procesos están utilizando la red simultáneamente.

1.1. Comunicación Extremo a Extremo

La comunicación extremo a extremo en la capa de transporte asegura que los datos se transfieran directamente entre las aplicaciones en los hosts emisores y receptores. Este modelo se ilustra en el diagrama mostrado en la página 7 del material, que detalla la interacción entre las capas de aplicación, transporte y red:

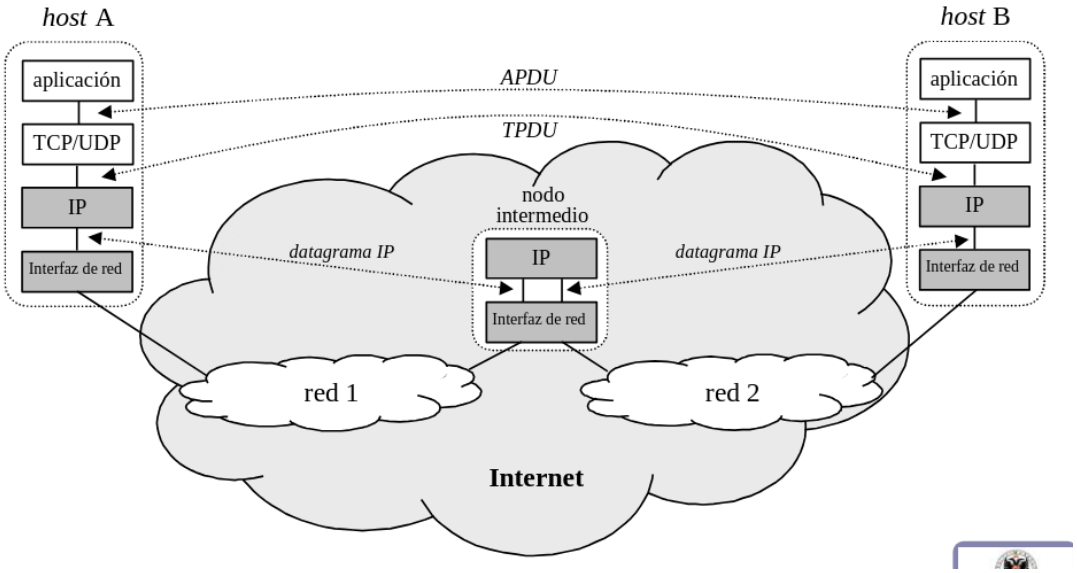

Figura 1: Esquema de comunicación extremo a extremo.

En el esquema, se destacan los siguientes elementos clave:

- **Capas de red y transporte:** Cada host (A y B) utiliza estas capas para encapsular y enviar los datos hacia la aplicación correspondiente en el otro extremo.
- **Datagramas IP y TPDU:** Los datagramas IP son utilizados por la capa de red para transportar las Unidades de Datos del Protocolo de Transporte (TPDU) entre los hosts.
- **APDU (Application Protocol Data Unit):**
  - Las APDU son las unidades de datos utilizadas en la capa de aplicación para la comunicación entre aplicaciones en diferentes sistemas.
  - En el contexto de la capa de transporte, las APDU se encapsulan en segmentos TCP o datagramas UDP, dependiendo del protocolo utilizado.
  - La capa de transporte se encarga de la multiplexación y demultiplexación de estas APDU, asegurando que se entreguen a la aplicación correcta en el host de destino.
  - Las APDU contienen la información específica de la aplicación, como comandos, respuestas y datos de usuario, que se transmiten de manera confiable (TCP) o no confiable (UDP) a través de la red.

- **Intermediarios:** Los nodos intermedios en la red solo manejan el nivel de red, garantizando la entrega de los datagramas IP sin interactuar con las capas superiores.
- **Capas de aplicación:** Las aplicaciones en ambos extremos envían y reciben datos procesados por las capas inferiores, asegurando la comunicación.

Este enfoque modular permite la interoperabilidad y confiabilidad en redes complejas, separando claramente las responsabilidades entre las distintas capas.

2 Protocola de Datagrama del Usuario (UDP)

Las siglas significan User Datagram Protocol, se trata de un protocolo que se basa en la funcionalidad de **best effor**. Se trata de un servicio que lleva a cabo el transporte entre redes, es no orientado a conexión, no handshaking[2], además no hay retardos en el establecimiento y cada TPDU[3] es independiente.

Además, abarca las características de multiplexación y demultiplexación, las cuales ayudan a transportar las TPDU al proceso correcto.

Existen algunos puertos con servicios normalizados, como es el caso del puerto 7 que es un *echo* con función Eco, el 13 que es *daytime* con función fecha, etc. Por otro lado, hay otros puertos que están libres para el uso propio del desarrollador.

Puerto	Aplicación/Servicio	Descripción
7	echo	Eco
13	daytime	Fecha
37	time	Hora
42	nameserver	Servicio de nombres
53	domain	Servicio de nombres de dominio
69	tftp	Transferencia simple de ficheros
123	ntp	Protocolo de tiempo de red

UDP se suele usar para servicios multimedia, debido a que es tolerante a fallos y sensible frente a los retardos que se puedan producir.

[2]El *handshaking* es un proceso de comunicación entre dos dispositivos que establece las reglas para la transmisión de datos. Este proceso asegura que ambos dispositivos estén sincronizados y listos para intercambiar información.

[3]TPDU significa *Transport Protocol Data Unit* (Unidad de Datos del Protocolo de Transporte). Es la unidad de datos que se intercambia entre entidades de la capa de transporte en un modelo de red, como el modelo OSI.

Cada segmento UDP se encapsula en un datagrama IP. Esto significa que el segmento UDP, que contiene los datos de la aplicación y los encabezados UDP, se coloca dentro de la carga útil de un datagrama IP. El datagrama IP, a su vez, añade su propio encabezado, que incluye información necesaria para el enrutamiento y la entrega a través de la red IP. De esta manera, los datos de la aplicación pueden ser transmitidos de manera eficiente a través de la red.

3 Protocolo de Control de Transmisión (TCP)

Características del Transmission Control Protocol (TCP)

El protocolo TCP, definido en diversos RFCs (793, 1122, 1323, 2018, 2581), ofrece un conjunto de características que lo hacen adecuado para la transmisión fiable de datos en redes complejas. Estas características incluyen:

- **Servicio punto a punto:** TCP establece una conexión directa entre dos hosts, permitiendo una comunicación exclusiva entre ellos.
- **Servicio orientado a conexión:** Requiere un estado común entre el emisor y el receptor, logrado a través de un procedimiento de inicialización denominado *handshaking*.
- **Entrega ordenada:** Asegura que las secuencias de bytes generadas por la aplicación lleguen en el orden correcto al receptor, proporcionando un servicio *stream oriented*.
- **Transmisión full-duplex:** Permite la transmisión simultánea de datos en ambas direcciones entre los hosts.
- **Mecanismo de detección y recuperación de errores:** Emplea esquemas ARQ (*Automatic Repeat reQuest*) con confirmaciones positivas acumulativas (ACKs) y temporizadores adaptables (*timeouts*).
- **Control de congestión y flujo:** Utiliza ventanas deslizantes de tamaño máximo adaptable para garantizar un uso eficiente de los recursos de red y evitar saturaciones.
- **Incorporación de confirmaciones (*piggybacking*):** Los ACKs pueden ser integrados en segmentos de datos para optimizar el uso del ancho de banda.

En cuanto a las funcionalidades de TCP encontramos las siguientes: multiplexación y demultiplexación de aplicaciones, control de la conexión (establecimiento y cierre), control de errores y flujo y el control de la gestión.

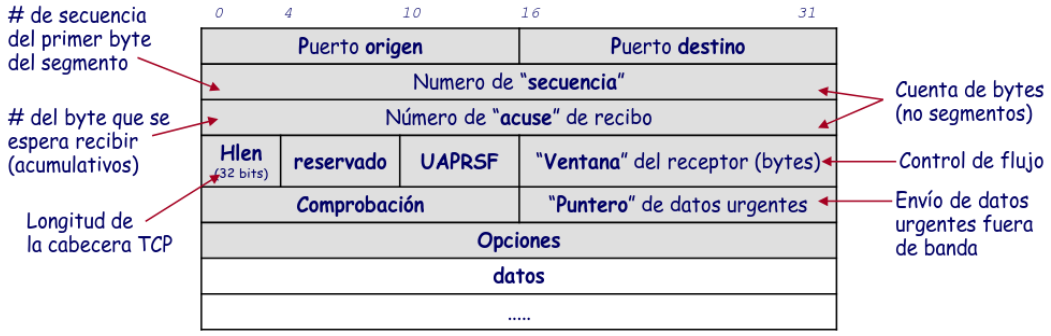

Figura 2: Estructura de un segmento TCP.

3.1. Multliplexación y Demultiplexación

Sirven para transportar las TPDU al proceso correcto, hay una serie de puertos asignados por defecto con servicio:

Puerto	Aplicación/Servicio	Descripción
20	FTP-DATA	Transferencia de ficheros: datos
21	FTP	Transferencia de ficheros: control
22	SSH	Terminal seguro
23	TELNET	Acceso remoto
25	SMTP	Correo electrónico
53	DNS	Servicio de nombres de dominio
80	HTTP	Acceso hipertexto (web)
110	POP3	Descarga de correo

Al igual que en anterior caso, hay otros puertos libres.

La conexión TCP se identifica por le puerto e IP tanto de origen como de destino.

3.2. Control de conexión

TCP ofrece un servicio orientado a conexión. El intercambio de la información tiene 3 fases:

- Establecimiento de la conexión: three-way handshake
- Intercambio de datos
- Cierre de la conexión

Three-way Handshake en TCP

El **three-way handshake** (o apretón de manos en tres pasos) es un proceso fundamental en el protocolo TCP (Transmission Control Protocol) que permite establecer una conexión confiable entre dos dispositivos (por ejemplo, un cliente y un servidor) antes de comenzar a transmitir datos. Este proceso asegura que ambos dispositivos están listos para la comunicación y pueden manejar los datos de manera adecuada.

El **three-way handshake** se lleva a cabo en tres pasos:

1. **SYN (Synchronize)**:
   - El cliente (el dispositivo que inicia la conexión) envía un segmento TCP con la bandera **SYN** activada. Este mensaje indica que el cliente desea establecer una conexión y está listo para sincronizar sus secuencias de números.
   - El mensaje también contiene un número de secuencia inicial que se utilizará para los datos que se transmitirán más adelante.
2. **SYN-ACK (Synchronize-Acknowledge)**:
   - El servidor, al recibir el paquete SYN del cliente, responde enviando un segmento TCP con las banderas **SYN** y **ACK** activadas.
   - El **SYN** indica que el servidor acepta la solicitud de conexión y está listo para comunicarse.
   - El **ACK** (Acknowledge) confirma la recepción del paquete SYN del cliente y le informa que ha recibido el número de secuencia del cliente. Además, el servidor incluye su propio número de secuencia, que se utilizará para las respuestas.
3. **ACK (Acknowledge)**:
   - Finalmente, el cliente envía un paquete de confirmación (ACK) al servidor, indicando que ha recibido correctamente el paquete SYN-ACK.
   - A partir de este momento, la conexión está completamente establecida, y los dos dispositivos pueden comenzar a intercambiar datos.

Este proceso de tres pasos asegura que tanto el cliente como el servidor estén listos para enviar y recibir datos, y que la conexión se realice de manera confiable, manteniendo un control adecuado de las secuencias de los paquetes de datos que se transmitirán.

Resumen:

1. **Cliente** $\rightarrow$ **SYN** (solicita la conexión)
2. **Servidor** $\rightarrow$ **SYN-ACK** (responde aceptando la solicitud)
3. **Cliente** $\rightarrow$ **ACK** (confirma la conexión)

Este método es una de las razones por las que TCP es considerado un protocolo orientado a la conexión y confiable.

3.2.1. Números de Secuencia.

El número de secuencia es un campo de 32 bits que cuenta bytes en módulo 2^{32} (el contador se da la vuelta cuando llega al valor máximo). Este número de secuencia no empieza normalmente en 0, sino en un valor denominado *ISN* (Initial Sequence Number), elegido teóricamente al azar, para evitar confusiones con solicitudes anteriores.

El *ISN* es elegido por el sistema (cliente o servidor). El estándar sugiere utilizar un contador entero incrementado en 1 cada 4 µs aproximadamente. En este caso, el contador se da la vuelta (y el *ISN* reaparece) al cabo de 4 horas y 46 minutos.

El mecanismo de selección de los *ISN* es suficientemente fiable para proteger de coincidencias, pero no es un mecanismo de protección frente a sabotajes. Es relativamente fácil averiguar el *ISN* de una conexión e interceptarla, suplantando a alguno de los dos participantes.

TCP incrementa el número de secuencia de cada segmento según los bytes que tenía el segmento anterior, con una sola excepción: cuando los flags **SYN** o **FIN** están activados, incrementan en 1 el número de secuencia. La presencia del flag **ACK** no incrementa el número de secuencia.

3.2.2. Establecimiento de Conexión TCP con el Three-Way Handshake

A continuación, se describen los pasos implicados en este proceso (Caso normal):

- **Paso 1: SYN** - TCP A envía un segmento con el flag SYN activado y un número de secuencia inicial (ISN) de 100 a TCP B. Este segmento se representa como `SYN seq=100`.
- **Paso 2: SYN-ACK** - TCP B responde con un segmento que tiene el flag SYN activado, un ISN de 300, y un número de acuse de recibo (ACK) de 101 (que es el ISN de TCP A más 1). Este segmento se representa como `SYN-ACK seq=300 ack=101`.
- **Paso 3: ACK** - Finalmente, TCP A envía un segmento con el flag ACK activado, un número de secuencia de 101 (ISN de TCP A más 1), y un número de acuse de recibo de 301 (ISN de TCP B más 1). Este segmento se representa como `ACK seq=101 ack=301`.

Después de estos tres pasos, la conexión TCP se establece y ambos puntos finales pueden comenzar a intercambiar datos de manera fiable. Este proceso es crucial para la comunicación en redes TCP/IP, ya que asegura que ambos puntos finales están preparados para recibir y enviar datos.

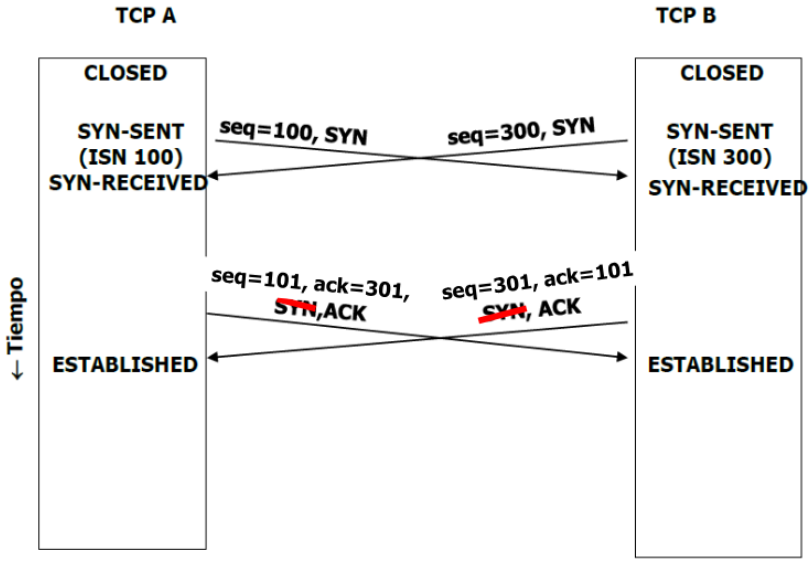

Figura 3: Proceso de Three-Way Handshake en TCP

3.2.3. Establecimiento de la Conexión TCP: Caso con SYN Retrasados y Duplicados

A continuación se describen los eventos representados en el diagrama :

- **TCP A:**
  - **SYN-SENT (ISN 90):** TCP A envía un segmento SYN con un número de secuencia inicial (ISN) de 90.
  - **CLOSED (timeout):** Debido a un tiempo de espera, TCP A cierra la conexión.
  - **SYN-SENT (ISN 100):** TCP A vuelve a enviar un segmento SYN, esta vez con un ISN de 100.
  - **Paquetes SYN duplicados y retrasados:** Varios segmentos SYN con los ISN 90 y 100 se envían, representando paquetes duplicados y retrasados.
  - **Respuesta de TCP B:** TCP B responde con un segmento `SYN-ACK seq=300 ack=91`, y TCP A envía un segmento `RST seq=91`.
  - **Nuevo intento con ISN 100:** TCP A vuelve a enviar un segmento SYN con ISN 100.
  - **SYN-ACK con ISN 400:** TCP B responde con `SYN-ACK seq=400 ack=101`, y TCP A confirma con un `ACK seq=101 ack=401`.
  - **Conexión establecida:** Finalmente, se establece la conexión y ambos puntos finales entran en el estado `ESTABLISHED`.
- **TCP B:**
  - **LISTEN:** TCP B espera la llegada de un segmento SYN.

- **SYN-RECEIVED (ISN 300):** TCP B recibe un segmento SYN y responde, entrando en estado `SYN-RECEIVED`.
- **Recepción de paquetes duplicados:** TCP B recibe varios segmentos SYN con ISN 90 y responde con un `RST seq=91`.
- **Nuevo SYN-RECEIVED (ISN 400):** TCP B recibe un nuevo SYN con ISN 100, responde con `SYN-ACK seq=400 ack=101`.
- **Conexión establecida:** Tras recibir el `ACK seq=101 ack=401` de TCP A, la conexión se establece y ambos puntos finales entran en el estado `ESTABLISHED`.

Esta secuencia demuestra cómo TCP maneja paquetes retrasados y duplicados, asegurando la fiabilidad de la conexión a pesar de las posibles interrupciones y retransmisiones.

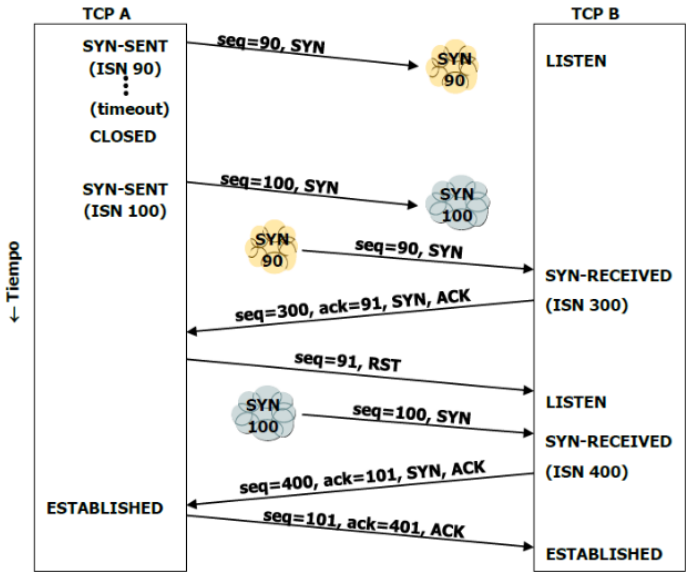

Figura 4: Establecimiento de la Conexión TCP con Paquetes SYN Retrasados y Duplicados

3.2.4. Establecimiento de la Conexión Simultánea en TCP

A continuación, se describe el proceso paso a paso (caso raro):

- **Estado inicial:** Ambos puntos, TCP A y TCP B, comienzan en el estado `CLOSED`.
- **SYN (TCP A):** TCP A envía un segmento SYN con un número de secuencia inicial (ISN) de 100 y pasa al estado `SYN-SENT`.
- **SYN (TCP B):** Simultáneamente, TCP B envía un segmento SYN con un ISN de 300 y también pasa al estado `SYN-SENT`.
- **SYN-ACK (Ambos):** Al recibir el SYN del otro punto final, ambos TCP A y TCP B envían un segmento SYN-ACK. TCP A envía `SYN-ACK seq=101 ack=301`, y TCP B envía `SYN-ACK seq=301 ack=101`.

- **ACK (Ambos):** Finalmente, ambos puntos finales envían un segmento ACK confirmando la recepción de los segmentos SYN-ACK. TCP A envía `ACK seq=101 ack=301` y TCP B envía `ACK seq=301 ack=101`.
- **Estado `ESTABLISHED`:** La conexión se establece y ambos puntos finales entran en el estado `ESTABLISHED`, listos para la transferencia de datos.

Este proceso de establecimiento simultáneo de conexión es una demostración de la capacidad de TCP para manejar conexiones iniciadas por ambos puntos finales al mismo tiempo, garantizando la sincronización y el control adecuados.

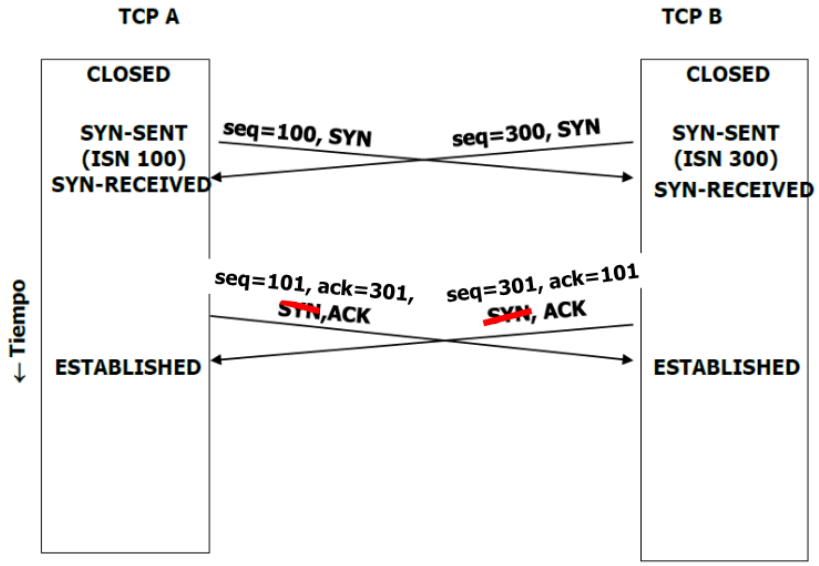

Figura 5: Proceso de Establecimiento de Conexión Simultánea en TCP

3.2.5. Cierre de la Conexión TCP

Se trata de la liberación de recursos y la finalización de la conexión. Para evitar bloqueos por pérdidas, una vez que comienza el proceso de cierre, se usan TimeOuts.

MLS = Maximum Segment Lifetime = 2 minutos

3.3. Control de Errores y Flujo

El control de errores y de flujo son aspectos fundamentales del protocolo TCP para asegurar la entrega fiable y ordenada de los datos entre dos dispositivos. Estos mecanismos son esenciales para optimizar el rendimiento y garantizar que los datos se transmitan correctamente y sin pérdidas.

Mejorar rendimiento → ventana deslizante. Para mejorar el rendimiento y optimizar el uso del ancho de banda, TCP utiliza un mecanismo conocido como *ventana deslizante* (sliding window). Este enfoque permite que el emisor envíe múltiples segmentos de

datos sin esperar confirmaciones para cada uno, lo que mejora la eficiencia. La ventana deslizante controla cuántos datos puede enviar el emisor antes de que se necesite una confirmación. A medida que se reciben los segmentos, la ventana se desliza hacia adelante, permitiendo al emisor continuar enviando más datos..

Control de errores: esquema ARQ con confirmaciones positivas y acumulativas. TCP implementa un esquema de control de errores basado en ARQ (Automatic Repeat reQuest), que utiliza confirmaciones positivas y acumulativas. Esto significa que el receptor envía un mensaje de confirmación (ACK) para indicar que ha recibido correctamente los datos. Si el emisor no recibe esta confirmación en un tiempo determinado, retransmite el segmento de datos. Además, las confirmaciones son acumulativas, es decir, un solo ACK puede confirmar la recepción de múltiples segmentos de datos hasta un número determinado..

Campos involucrados en el control de errores y flujo:

- **Campo de secuencia:** Este campo contiene el número de secuencia del primer byte de datos en el segmento, lo que indica el offset (desplazamiento) en bytes dentro del mensaje. El número de secuencia es esencial para la reordenación de los segmentos en caso de que lleguen fuera de orden..

- **Campo de acuse de recibo (ACK):** Este campo contiene el número de byte esperado en el receptor. Cuando el receptor envía un ACK, está indicando el próximo byte que espera recibir. Este valor es crucial para el control de flujo, ya que ayuda al emisor a saber hasta qué punto los datos han sido recibidos correctamente...

- **Bit A (ACK) del campo de control:** El bit ACK en el campo de control se establece cuando el segmento es una confirmación de la recepción de datos. Si este bit está activado, significa que el receptor ha recibido correctamente los datos hasta el número de secuencia especificado en el campo de acuse de recibo..

- **Campo de comprobación (Checksum):** El campo de comprobación contiene un valor de verificación para asegurar la integridad del segmento de datos. Este valor se calcula aplicando un algoritmo de checksum sobre todo el segmento TCP, que incluye tanto los datos como la cabecera del segmento. Si el valor de checksum calculado en el receptor no coincide con el valor enviado en el segmento, se detecta un error y el segmento se descarta..

- **Uso de pseudo-cabecera TCP:** Para mejorar la fiabilidad del control de errores, TCP utiliza una pseudo-cabecera, que es un bloque de datos adicional utilizado en el cálculo del checksum. Esta pseudo-cabecera incluye la dirección IP de origen y des-

tino, el protocolo (TCP) y la longitud de los datos, lo que ayuda a garantizar que los segmentos TCP se entreguen al destino correcto y no se corrompan durante la transmisión..

En conjunto, estos mecanismos de control de errores y de flujo permiten a TCP proporcionar una transmisión de datos fiable y eficiente, minimizando la posibilidad de errores y optimizando el uso del ancho de banda disponible..

3.3.1. Pérdida de ACK y Timeout Prematuro con ACK Acumulativo

En la imagen se muestran dos diagramas de secuencia que ilustran la comunicación entre dos hosts, Host A y Host B, en una red. La imagen está dividida en dos partes:

1. **Pérdida de ACK**: En esta parte del diagrama, Host A envía un paquete con el número de secuencia 92 y 8 bytes de datos a Host B. Host B recibe el paquete y envía un ACK con el número 100 de vuelta a Host A. Sin embargo, este ACK se pierde en la red (indicado por una X y la palabra pérdida). Debido a la pérdida del ACK, Host A no recibe la confirmación y, después de un tiempo de espera (timeout), retransmite el paquete con el número de secuencia 92 y 8 bytes de datos. Finalmente, Host B recibe el paquete retransmitido y envía nuevamente el ACK con el número 100.

2. **Timeout Prematuro y ACK Acumulativo**: En esta parte del diagrama, Host A envía un paquete con el número de secuencia 92 y 8 bytes de datos a Host B. Luego, Host A envía otro paquete con el número de secuencia 100 y 20 bytes de datos. Host B recibe ambos paquetes y envía un ACK con el número 100 y luego un ACK con el número 120 de vuelta a Host A. Sin embargo, debido a un timeout prematuro, Host A retransmite el paquete con el número de secuencia 92 y 8 bytes de datos antes de recibir el ACK acumulativo con el número 120. Finalmente, Host B recibe el paquete retransmitido y envía nuevamente el ACK con el número 120.

Este diagrama es relevante porque ilustra dos situaciones comunes en la comunicación de redes: la pérdida de ACK y el manejo de timeouts prematuros con ACK acumulativos. Estas situaciones son importantes para entender cómo los protocolos de comunicación, como TCP, manejan la retransmisión de paquetes y la confirmación de recepción para asegurar una comunicación confiable.

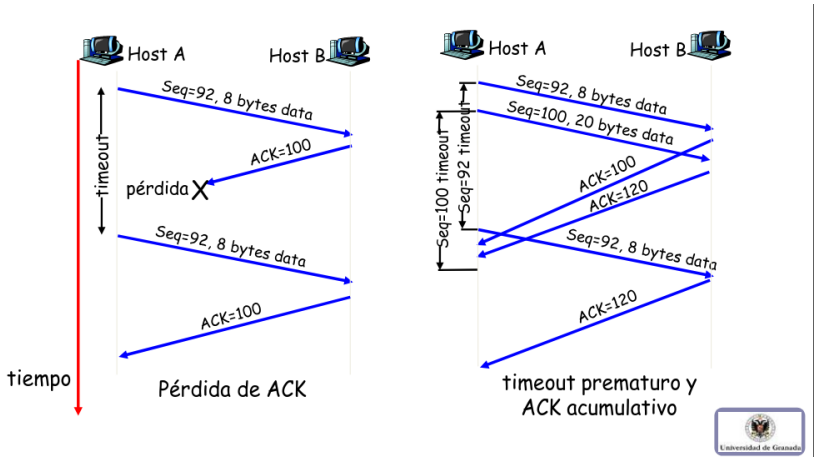

Figura 6: Pérdida de ACK y Timeout Prematuro con ACK Acumulativo

Evento	**Acción del TCP receptor**
Llegada ordenada de segmento, sin discontinuidad, todo lo anterior ya confirmado.	Retrasar ACK. Esperar recibir al siguiente segmento hasta 500 mseg. Si no llega, enviar ACK.
Llegada ordenada de segmento, sin discontinuidad, hay pendiente un ACK retrasado.	Inmediatamente enviar un único ACK acumulativo.
Llegada desordenada de segmento con # de sec. mayor que el esperado, discontinuidad detectada.	Enviar un ACK duplicado, indicando el # de sec. del siguiente byte esperado.
Llegada de un segmento que completa una discontinuidad parcial o totalmente.	Confirmar ACK inmediatamente si el segmento comienza en el extremo inferior de la discontinuidad.

Cuadro 1: Eventos y acciones del receptor TCP

3.3.2. ¿Cómo estimar los *timeouts*?

La estimación correcta de los *timeouts* es crucial para el control de errores en las redes de computadoras. Un tiempo de espera (*timeout*) demasiado pequeño puede llevar a *timeouts* prematuros, mientras que uno demasiado grande puede resultar en una reacción lenta a la pérdida de segmentos. La mejor solución en situaciones cambiantes es adaptar los *timeouts* de manera dinámica.

RTTmedido: tiempo desde la emisión de un segmento hasta la recepción del ACK.

La fórmula para calcular un nuevo valor de RTT (Round Trip Time) es la siguiente:

$$\text{RTTnuevo} = \alpha \cdot \text{RTTviejo} + (1 - \alpha) \cdot \text{RTTmedido}, \quad \alpha \in [0, 1]$$

Donde:

- RTTnuevo: es el nuevo valor del RTT.
- RTTviejo: es el valor anterior del RTT.

- RTTmedido: es el RTT medido en la última transmisión.
- α: es un factor de ponderación entre 0 y 1.

Además, se debe calcular la desviación del RTT para tener en cuenta la variabilidad en los tiempos de transmisión. La fórmula es la siguiente:

$$\text{Desviación}_{\text{nueva}} = (1 - x) \cdot \text{Desviación}_{\text{vieja}} + x \cdot |\text{RTTmedido} - \text{RTTnuevo}|$$

Finalmente, el *timeout* se calcula utilizando la siguiente fórmula:

$$\text{Timeout} = \text{RTTnuevo} + 4 \cdot \text{Desviación}$$

Esta estimación dinámica del *timeout* asegura que los tiempos de espera se ajusten apropiadamente a las condiciones de la red, evitando tanto *timeouts* prematuros como retrasos excesivos.

Problema con ACKs repetidos: ambigüedad en la interpretación

La ambigüedad en la interpretación de ACKs repetidos puede complicar la estimación del RTT. La solución a este problema es utilizar el Algoritmo de Karn, que actualiza el RTT solo para aquellos segmentos que no presentan ambigüedad. Sin embargo, si es necesario repetir un segmento, se debe incrementar el *timeout* de la siguiente manera:

$$t_{\text{out}_{\text{nuevo}}} = \gamma \cdot t_{\text{out}_{\text{viejo}}}, \quad \gamma = 2$$

Donde:

- $t_{\text{out}_{\text{nuevo}}}$: es el nuevo valor del *timeout*.
- $t_{\text{out}_{\text{viejo}}}$: es el valor anterior del *timeout*.
- γ: es un factor de multiplicación que generalmente se establece en 2.

Estas fórmulas y procedimientos permiten ajustar dinámicamente los tiempos de espera en las transmisiones TCP, mejorando la eficiencia y fiabilidad de las comunicaciones en redes de datos.

3.3.3. Control de flujo

Se trata del procedimiento en el que se controla lo que el emisor envía al receptor. Para ello, se utilizan ventanas de flujo, que son un mecanismo que permite al receptor controlar la cantidad de datos que el emisor puede enviar antes de recibir una confirmación. Es un esquema *crediticio*, es decir, el receptor informa al emisor sobre los bytes autorizados a emitir sin esperar respuesta.

$$\text{Ventana útil emisor} = \text{Ventana ofertada receptor} - \text{bytes en tránsito} \tag{1}$$

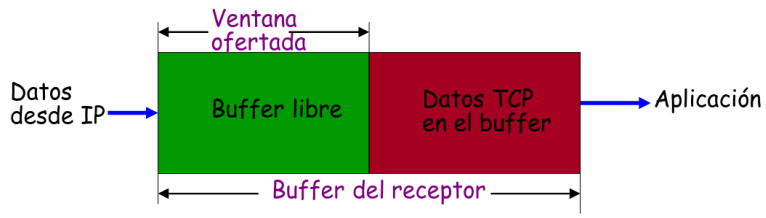

Figura 7: Control de flujo en TCP

3.3.4. Ejemplo Práctico: Control de Flujo en TCP

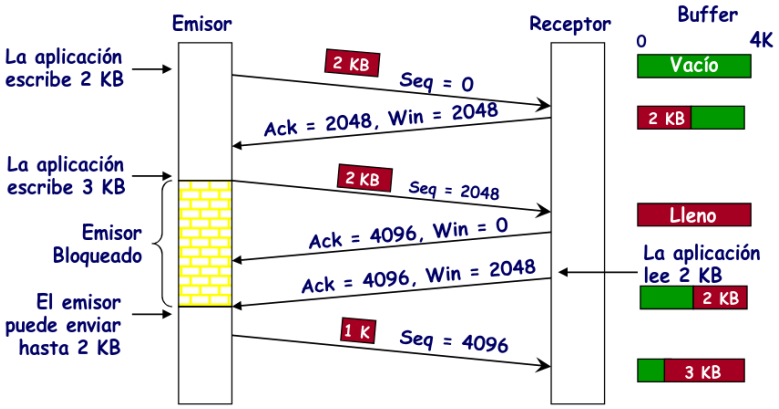

Figura 8: Ejemplo práctico de control de flujo en TCP

Control de Flujo y Gestión de Buffer

La imagen muestra un diagrama de comunicación entre un emisor y un receptor, con un enfoque en la gestión del buffer y el control de flujo en una red de transmisión de datos. A continuación, se detalla lo que sucede en cada paso:

1. **La aplicación escribe 2 KB**: El emisor envía 2 KB de datos al receptor con un número de secuencia (Seq) de 0.

2. **Ack = 2048, Win = 2048**: El receptor envía un acuse de recibo (Ack) indicando que ha recibido 2048 bytes y que su ventana de recepción (Win) es de 2048 bytes.

3. **La aplicación escribe 3 KB**: El emisor intenta enviar 3 KB de datos adicionales, pero solo puede enviar 2 KB debido a la ventana de recepción del receptor.

4. **Emisor Bloqueado**: El emisor está bloqueado porque no puede enviar más datos hasta que reciba un nuevo acuse de recibo del receptor.

5. **Ack = 4096, Win = 0**: El receptor envía un acuse de recibo indicando que ha recibido 4096 bytes, pero su ventana de recepción está llena (Win = 0).

6. **El emisor puede enviar hasta 2 KB**: El emisor recibe un nuevo acuse de recibo (Ack = 4096, Win = 2048) indicando que puede enviar hasta 2 KB más de datos.

7. **Seq = 4096**: El emisor envía 1 KB de datos adicionales con un número de secuencia de 4096.

El buffer del receptor se muestra en diferentes estados:

- Inicialmente vacío.
- Luego con 2 KB de datos.
- Finalmente lleno con 4 KB de datos.
- Después de que la aplicación lee 2 KB, el buffer tiene 2 KB de datos y espacio para 2 KB más.

Este diagrama es relevante porque ilustra cómo se maneja el control de flujo y la gestión de buffers en una red de transmisión de datos, asegurando que los datos se envíen y reciban de manera eficiente sin perder información.

3.3.5. Síndrome de la ventana tonta

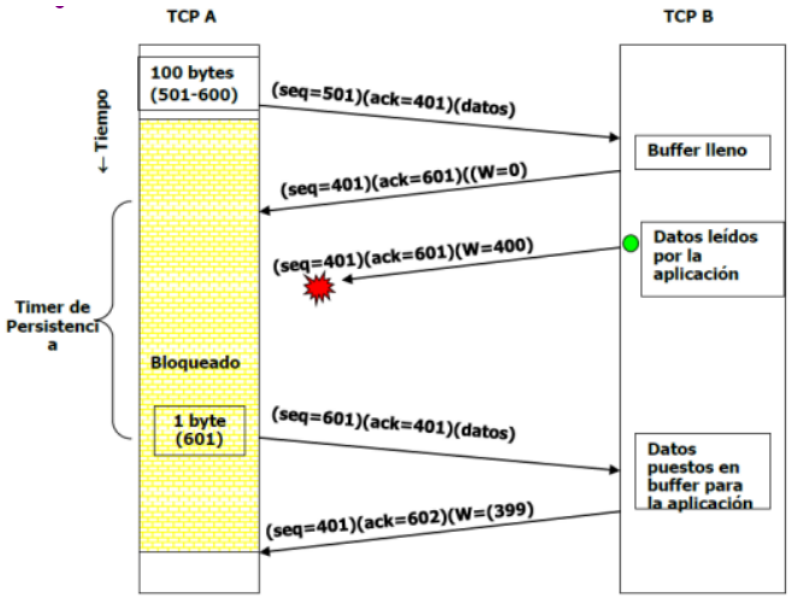

Figura 9: Síndrome de la ventana tonta en TCP

Gestión del Flujo y del Tamaño de Ventana en TCP

La imagen muestra una secuencia de eventos en una comunicación TCP (Transmission Control Protocol) entre dos entidades, TCP A y TCP B. Se ilustra el proceso de

transmisión de datos, acuse de recibo (ACK) y gestión del tamaño de ventana (W). El diagrama está dividido en dos secciones principales: TCP A a la izquierda y TCP B a la derecha. En este caso en concreto se hace referencia al síndrome de la *ventana tonta*, un problema que puede surgir en la gestión del flujo y del tamaño de ventana en TCP, que significa que el emisor envía datos a una velocidad más rápida que la q ue el receptor puede procesar, lo que resulta en una baja eficiencia y un rendimiento subóptimo. Se puede solucionar con *la ventana optimista*, que permite al emisor enviar datos a la máxima velocidad posible y ajustar la tasa de transmisión en función de la capacidad del receptor. Se pueden hace entregas no ordenadas y solicitar una entrega inmediata a la aplicación.

En TCP A:

- **100 bytes (501-600)**: Inicialmente se envían 100 bytes con un número de secuencia (Seq) de 501 a 600.
- **(seq=501)(ack=401)(datos)**: TCP A envía datos con el número de secuencia 501 y espera un acuse de recibo para 401.
- **(seq=401)(ack=601)(W=0)**: TCP A recibe un acuse de recibo indicando que ha recibido 601 bytes, pero la ventana de recepción es 0, lo que significa que TCP A debe dejar de enviar datos temporalmente.
- **(seq=401)(ack=601)(W=400)**: TCP A envía un nuevo segmento con datos y recibe un acuse de recibo indicando que la ventana de recepción ha aumentado a 400 bytes.
- **(seq=601)(ack=401)(datos)**: TCP A envía datos adicionales con el número de secuencia 601.
- **(seq=401)(ack=602)(W=399)**: TCP A recibe un acuse de recibo indicando que ha recibido 602 bytes y que la ventana de recepción es de 399 bytes.

En TCP B:

- **Buffer lleno**: El estado del buffer se indica mostrando que está lleno.
- **Datos leídos por la aplicación**: Los datos en el buffer son leídos por la aplicación.
- **Datos puestos en buffer para la aplicación**: Los datos son colocados en el buffer para ser leídos por la aplicación.

3.4. Control de Gestión

El control de congestión es un mecanismo fundamental en TCP para manejar la saturación en la red, descrito en la *RFC 2001*. Este problema surge debido a la insuficiencia de recursos, como el ancho de banda de las líneas de transmisión, la capacidad de los buffers en los routers y los sistemas finales. La congestión puede degradar significativamente el rendimiento de la red si no se gestiona adecuadamente. Se trata de un problema debido a la insuficiencia de recursos (ancho de banda de las líneas como

buffer en routers y sistemas finales.)

Diferencias con el control de flujo. Aunque puede parecer similar al control de flujo, el control de congestión aborda un problema diferente. Mientras que el control de flujo regula la cantidad de datos enviados para no sobrecargar al receptor, el control de congestión involucra tanto a la red como a los sistemas finales. Es una problemática que abarca todo el sistema de comunicación y no se limita a un enlace específico.

Naturaleza adelante-atrás. El control de congestión tiene una naturaleza de comunicación bidireccional (adelante-atrás), ya que los efectos de la congestión pueden propagarse en ambas direcciones del flujo de datos. Esto se manifiesta en:

- **Pérdidas de paquetes:** Los segmentos que no pueden ser almacenados en los buffers de los routers debido a la congestión son descartados.
- **Retrasos en las ACKs:** La congestión también provoca un incremento en los tiempos de transmisión de las confirmaciones (ACKs), lo que puede impactar el rendimiento del protocolo TCP.

Solución: limitar de forma adaptable el tráfico generado. La solución al problema de la congestión reside en que la fuente (el emisor) limite dinámicamente el tráfico generado, adaptándose a las condiciones de la red. Este enfoque incluye varios mecanismos clave:

- *Congestion Window (cwnd):* TCP utiliza una ventana de congestión que controla cuántos datos pueden enviarse sin recibir ACKs. La ventana se ajusta dinámicamente según la retroalimentación de la red.

- *Mecanismos de detección de congestión:* TCP responde a los indicadores de congestión, como pérdidas de paquetes o retrasos en las ACKs, disminuyendo la ventana de congestión para reducir el tráfico enviado.

- *Recuperación adaptativa:* Después de detectar congestión, TCP incrementa gradualmente la ventana de congestión a medida que las condiciones de la red mejoran, utilizando técnicas como el *Slow Start* y el *Congestion Avoidance*.

En el emisor se utilizan dos ventanas y un umbral:

- Bytes_permitidos_enviar = min(cwnd, rwnd[4]).
- Ventana del receptor = se usa para el control de flujo de tamaño variable según el campo de ventana recibido.
- Inicialmente la ventana de congestión es de 1 MSS[5].
- Inicio Lento:

[4]Ventana del receptor
[5]Maximum Segment Size

```
    if (ventana de congestión < umbral, POR CADA ACK RECIBIDO){
        ventana de congestión ++ (crecimiento exponencial)
    }
```

- Prevención de la congestión:

```
    if (ventana de congestión > umbral, CADA VEZ QUE SE RECIBEN TODOS
       LOS ACKS PENDIENTES){
        ventana de congestión ++ (crecimiento lineal)
    }


    if (hay timeouts) {
        umbral = ventana de congestión / 2
        ventana de congestión = 1
    }
```

4 Extensiones TCP

Una de las definiciones de tcp es aquella en la que se expone que tiene múltiples *sabores*, es decir, diferentes versiones que se adaptan a las necesidades de los usuarios. Estas versiones no afectan a la *interoperabilidad* de los extremos. Se realizaron diversas modificaciones de manera que:

- **Ventana escalada:** Esta opción se utiliza en los segmentos SYN de TCP para permitir una ventana de recepción más grande. Con la ventana escalada, el tamaño de la ventana puede aumentar hasta $2^{14} \times 2^{16} = 2^{30} = 1$ GB, lo que mejora significativamente el rendimiento en redes de alta capacidad y alta latencia.
- **Estimación de RTT:** La opción de sello de tiempo en TCP permite una estimación más precisa del RTT (Round Trip Time). Al incluir un sello de tiempo en todos los segmentos, TCP puede medir el tiempo exacto que tarda un segmento en ser reconocido, mejorando así la eficiencia del control de congestión y la retransmisión.
- **PAWS (Protection Against Wrapped Sequence Numbers):** Esta extensión utiliza sellos de tiempo para proteger contra la reutilización de números de secuencia. PAWS rechaza los segmentos duplicados que tienen sellos de tiempo más antiguos, asegurando que los datos se entreguen de manera ordenada y sin duplicaciones.