TCP (Transmision Control Protocol)

Es un protocolo orientado a la conexion, diseñado para ofrecer una comunicación confiable entre 2 sistemas de una red. Establece una conexion antes de comenzar a transferir datos (3WayHandshake) y asegura que los paquetes lleguen en orden y sin errores (a diferencia de UDP). Es ideal para aplicaciones que requieren precisión como la web, emails, y transferencia de archivos.

El protocolo busca evitar situaciones donde una conexión acapare más recursos a costa de otras

Estados de una conexión TCP

Listen: El servidor está esperando activamente solicitudes de conexión entrantes. Este estado es común en el lado del servidor.

SYN-Sent: El cliente ha enviado un paquete de solicitud de conexión (SYN) y está esperando una respuesta del servidor. Este estado se presenta en el cliente al iniciar una conexión.

SYN-Received: El servidor ha recibido el paquete SYN del cliente y ha enviado una respuesta SYN-ACK, esperando el ACK final del cliente para establecer la conexión.

Established: La conexión está completamente establecida y ambos lados pueden comenzar a intercambiar datos. Este es el estado normal durante la transmisión de datos.

FIN-Wait-1: Una de las partes ha enviado un mensaje de finalización (FIN) y está esperando un ACK del otro lado o un FIN de vuelta.

FIN-Wait-2: Después de recibir el ACK de su FIN, el lado que inició el cierre está esperando el FIN del otro lado.

Close-Wait: Una de las partes ha recibido un FIN y está esperando a que la aplicación cierre la conexión. Este estado es común en el lado que no inicia el cierre.

Closing: Ambos lados han enviado un FIN, pero están esperando la confirmación ACK del otro para finalizar la conexión.

Last-Ack: La parte que ha recibido el FIN del otro lado ha enviado su propio FIN y está esperando el último ACK para completar el cierre de la conexión.

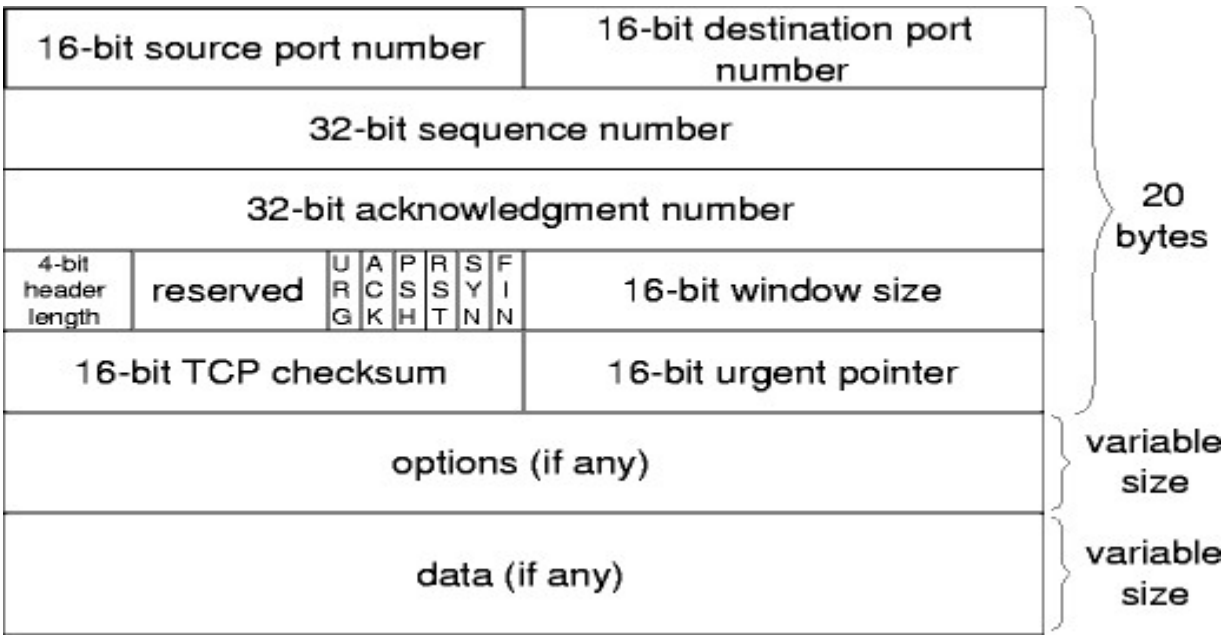
Time-Wait: El lado que cierra la conexión espera durante un periodo determinado (usualmente el doble del tiempo máximo de vida del segmento, 2 * MSL) para asegurarse de que cualquier paquete perdido o duplicado haya sido recibido antes de liberar los recursos.

Closed: La conexión ha sido completamente cerrada y todos los recursos han sido liberados.

Aclaraciones:

- **SYN-Received**: Ocurre típicamente en el lado del servidor cuando recibe el SYN del cliente y responde con un SYN-ACK.
- **FIN-Wait-2**: Se presenta después de que el cliente ha enviado el FIN y recibió un ACK del servidor, quedando a la espera del FIN del servido

Segmentos TCP



3-Way Handshake (Abrir Conexión)

"El 3-Way Handshake es el proceso estándar que TCP utiliza para establecer una conexión confiable entre un cliente y un servidor."

(El ISN es el número de secuencia inicial que se utiliza en el primer segmento del handshake. En TCP, cada conexión tiene un número de secuencia único, que asegura que los paquetes lleguen en el orden correcto y se detecten duplicados.)

Involucra tres pasos o segmentos:

- SYN (Cliente a Servidor): El cliente inicia la conexión enviando un segmento con la bandera SYN activada, junto con su número de secuencia inicial (ISN, Initial Sequence Number).
- SYN-ACK (Servidor a Cliente): El servidor responde con un segmento que tiene ambas

banderas SYN y ACK activadas, confirmando la solicitud del cliente (ACK) y enviando su propio número de secuencia inicial (SYN).

- ACK (Cliente a Servidor): El cliente confirma la recepción del número de secuencia del servidor enviando un segmento con la bandera ACK activada. Después de esto, la conexión está establecida y ya se puede enviar información.

4-Way Handshake (Cierre ordenado)

1. El cliente o el servidor puede iniciar el cierre enviando un FIN
2. El otro extremo responde con un ACK para confirmar la recepcion del FIN
3. Y ademas envia su propio FIN
4. El primer extremo responde con un ACK final

Resumen de las funciones principales de TCP:

- Control de flujo.
- Control de congestión.
- Control de errores.
- Establecimiento y finalización de conexiones.
- Multiplexacion.
- Entrega confiable y en orden.
- Control de duplicación

Control de Flujo

El control de flujo en TCP es un mecanismo diseñado para regular la tasa a la que el transmisor envía datos, asegurando que el receptor no se sobrecargue con más información de la que puede procesar. Este mecanismo es clave para evitar que se llene el buffer del receptor y se pierdan datos, lo que mejora la eficiencia en la utilización de los recursos de red.

Objetivo del Control de Flujo

- **Prevenir sobrecarga del receptor**: El control de flujo garantiza que el emisor no envíe más datos de los que el receptor puede manejar, asegurando que haya suficiente espacio en el **buffer de recepción (Rx Buffer)** para los datos entrantes.

Mecanismo de Activación

- **Iniciado por el receptor**: Cuando el receptor detecta que está recibiendo más datos de los que puede procesar, envía una señal al emisor para que reduzca la tasa de transmisión.
- **Ventana de recepción (rwnd)**: El control de flujo en TCP se implementa mediante la **ventana de recepción (receive window o rwnd)**, que el receptor incluye en los paquetes de confirmación (**ACKs**) enviados al emisor. Este valor indica la cantidad de datos adicionales que el receptor puede recibir sin problemas.

Funcionamiento del Control de Flujo

1. **Ajuste por parte del emisor**: El emisor monitorea el valor de la ventana de recepción y ajusta la cantidad de datos enviados en consecuencia. Si el tamaño de la ventana es pequeño o llega a cero, el emisor deja de enviar datos hasta que el receptor libere más espacio.
2. **Duración y desactivación**: El control de flujo está activo durante toda la conexión y se ajusta dinámicamente según cambien las capacidades del receptor. Se "relaja" temporalmente cuando el receptor actualiza su ventana de recepción, indicando que ha procesado los datos en su buffer y puede recibir más.

Principales Características del Control de Flujo

1. **Ventana deslizante (Sliding Window)**: TCP utiliza este mecanismo para gestionar el control de flujo. El receptor indica el tamaño del buffer disponible (ventana anunciada o **Advertised Window**) en los segmentos TCP. Este valor indica cuántos datos puede enviar el emisor sin esperar confirmación de recepción.
    - ⑩ Si el buffer del receptor se llena, la ventana se reduce, limitando la cantidad de datos que el emisor puede enviar.
2. **Recepción y confirmación**: Los segmentos recibidos por el receptor se almacenan en el **Rx Buffer**, y el receptor confirma la recepción mediante un **ACK**. A medida que el receptor procesa los datos del buffer, libera espacio, lo que permite al emisor enviar más datos.
3. **Actualización dinámica de la ventana**: El tamaño de la ventana de recepción cambia dinámicamente en función del espacio disponible en el buffer del receptor. Este tamaño se comunica al emisor mediante los **ACKs**, garantizando que siempre se envíe una cantidad de datos que el receptor pueda manejar.

Algoritmos de Mejora del Control de Flujo

1. **Delayed ACKs**: Este mecanismo retrasa el envío de ACKs para combinarlos con datos que deben ser enviados (piggybacking), reduciendo la cantidad total de segmentos enviados.

2. **Algoritmo de Nagle**: Evita el envío de segmentos muy pequeños, acumulando datos hasta que se puede enviar un bloque mayor. Mejora el rendimiento, pero puede aumentar la latencia en aplicaciones interactivas.
3. **Escalado de ventana (Window Scaling)**: En redes de alta capacidad y alta latencia, TCP utiliza un escalado de ventana (RFC-7323) que permite tamaños de ventana superiores a 64 KB, mejorando la eficiencia en este tipo de entornos.

Control de Congestión

El control de congestión es un mecanismo esencial para evitar la saturación de la red, permitiendo que los transmisores ajusten dinámicamente la cantidad de datos que envían en función de la percepción de congestión en la red. A través de señales como los **ACKs duplicados** o los **timeouts**, el emisor adapta su comportamiento para mantener un flujo eficiente de datos.

Principios Básicos

1. **Control extremo a extremo**: TCP ajusta la cantidad de datos enviados (ventana de congestión o **cwnd**) según la retroalimentación del receptor, sin depender de dispositivos intermedios en la red.
2. **Retroalimentación implícita o explícita**: El emisor se basa en señales de la red para detectar posibles problemas de congestión, como la pérdida de paquetes o el retraso en la recepción de ACKs.

Disparadores de Control de Congestión

1. **Pérdida de paquetes**: Indicada por la no recepción de ACKs a tiempo o la llegada de múltiples ACKs duplicados.
2. **Retransmisiones**: Cuando el emisor debe retransmitir paquetes debido a timeouts o múltiples ACKs duplicados, se asume que hay congestión.

Mecanismo de Ajuste

- El emisor ajusta dinámicamente la **ventana de congestión (cwnd)**, que define la cantidad máxima de datos que puede enviar sin recibir un ACK.

Problemas de la Congestión en la Red

- **Pérdida de paquetes**: Se pierden datos, lo que obliga a retransmisiones.
- **Retransmisiones innecesarias**: Esto empeora la congestión ya existente.
- **Aumento de la latencia**: El tiempo de transmisión se alarga debido a la saturación.
- **Baja eficiencia de la red**: Los recursos de la red no se utilizan de manera óptima.

Comparación con el Control de Flujo

Mientras que el **control de flujo** se enfoca en ajustar la cantidad de datos que el emisor envía con respecto a la capacidad del receptor, el **control de congestión** se centra en la capacidad de la red en sí misma. Ambos son esenciales para garantizar la transmisión confiable y eficiente de datos en TCP.

Fases del Proceso de Control de Congestión en TCP

El proceso de control de congestión en TCP se puede dividir en varias etapas, que ayudan a controlar de manera eficiente el tráfico de red para evitar que se sobrecargue. Aquí te explico de manera más clara y detallada cada fase:

<u>Resumen del proceso:</u>

1. Partida lenta: crecimiento exponencial hasta llegar al umbral.
2. Evitación de congestión: crecimiento lineal después del umbral.
3. Fast Retransmit/Recovery: retransmisión rápida tras recibir tres ACKs duplicados y crecimiento controlado.
4. Timeout: reducción drástica de la ventana de congestión y reinicio del proceso.

1.Fase de Partida Lenta (Slow Start)**:**

- Esta fase ocurre al inicio de una conexión o tras un **timeout**.
- El tamaño inicial de la **ventana de congestión** (*CongWin*) es pequeño, usualmente 1 **MSS** (Maximum Segment Size).
- El tamaño de la ventana crece **exponencialmente**. Por cada **ACK** recibido, la ventana de congestión se duplica, permitiendo enviar más datos de forma rápida.
- Este crecimiento exponencial sigue hasta que se alcanza un umbral predefinido llamado **ssthresh** (*Slow Start Threshold*) o hasta que se detecta congestión.

2. Fase de Crecimiento Lineal o Evitación de Congestión (Congestion Avoidance)**:**

- Cuando el tamaño de la **ventana de congestión (CongWin)** supera el umbral **ssthresh**, TCP entra en esta fase.
- El crecimiento de la ventana es **lineal** y más conservador. El objetivo es evitar congestionar la red, ya que se asume que TCP está cerca de la capacidad máxima de la red.
- Aquí, el tamaño de la ventana se incrementa aproximadamente en 1 MSS por ciclo de

ida y vuelta (**RTT**).

3. Retransmisión Rápida y Recuperación Rápida (Fast Retransmit & Fast Recovery)**:**

- Si TCP recibe **tres ACKs duplicados**, indica que un paquete se ha perdido pero otros han llegado correctamente. Esto activa el **Fast Retransmit**.
- ➓ TCP retransmite inmediatamente el paquete perdido sin esperar que el temporizador de retransmisión expire.
- Después de retransmitir el paquete, TCP reduce la **ventana de congestión** a la **mitad** y entra en **Fast Recovery**.
- ➓ En esta fase, TCP asume que la congestión no es severa, por lo que no reduce la ventana a 1 MSS, sino que continúa aumentando de manera lineal, como en la fase de evitación de congestión.
- ➓ Si los ACKs no llegan después de la retransmisión rápida, TCP vuelve a la fase de partida lenta, asumiendo congestión más severa.

4. Timeout (Tiempo de espera)**:**

- Si no se reciben **ACKs** después de un tiempo determinado (se alcanza un timeout), TCP asume que hay congestión severa.
- En este caso, **CongWin** se reduce drásticamente a 1 MSS y se reinicia el proceso de **partida lenta**.
- El umbral **ssthresh** también se ajusta, generalmente a la mitad del tamaño de la ventana en el momento del timeout, para reflejar la nueva capacidad estimada de la red.

<u>TCP utiliza algoritmos para el control de la congestión</u>

Additive Increase Multiplicative Decrease (AIMD) es un algoritmo de control de congestión utilizado en TCP para ajustar dinámicamente la cantidad de datos que se transmiten por una red. Su objetivo es encontrar un equilibrio entre maximizar la velocidad de transmisión y evitar la congestión de la red.

Additive Increase (Incremento Aditivo):

Aumento lineal de la ventana de congestión: Cuando no hay señales de congestión en la red, el tamaño de la ventana de congestión (CongWin) aumenta de manera aditiva. Específicamente, se incrementa en un segmento por cada ciclo de ida y vuelta (RTT).

Este comportamiento permite que TCP aproveche el ancho de banda disponible sin generar

congestión.

Multiplicative Decrease (Disminución Multiplicativa):

Reducción rápida del tamaño de la ventana: Cuando TCP detecta congestión (por ejemplo, a través de la pérdida de paquetes), la ventana de congestión se reduce de manera multiplicativa. Generalmente, el tamaño de la ventana se reduce a la mitad, lo que permite que la red se recupere de la congestión rápidamente.

Esto garantiza que la transmisión se ralentice considerablemente cuando hay señales de congestión, evitando que la situación empeore.

- **Slow Start:** Este algoritmo se utiliza al inicio de una conexión o después de una pérdida de paquetes. TCP comienza enviando un pequeño número de paquetes (generalmente un segmento) duplicando el tamaño de la ventana de congestión cada ciclo de ida y vuelta (RTT). Esto continúa hasta que se detecta congestión o se alcanza un umbral predefinido llamado "ssthresh" (threshold de slow start).
- **Fast Retransmit:** En lugar de esperar que el temporizador de retransmisión expire, TCP retransmite un paquete inmediatamente si recibe tres duplicados de un ACK, lo que indica la pérdida de un paquete.
- **Fast Recovery**: Tras un fast retransmit, TCP no reduce la ventana de congestión al valor mínimo, sino que sigue enviando datos más lentamente mientras espera que la red se estabilice.
- **Explicit Congestion Notification (ECN):**Es una extensión opcional en la que los routers marcan los paquetes cuando hay riesgo de congestión, permitiendo que TCP reaccione a estos marcadores antes de que se pierdan

Algoritmos de Control de Congestión más Avanzados:

- **TCP Reno**: Introduce el concepto de Fast Retransmit y Fast Recovery, mejorando la respuesta ante la pérdida de paquetes.
- **TCP Tahoe**: Es una versión anterior que implementa slow start, evitación de congestión y AIMD, pero no incluye la recuperación rápida de Reno.
- **TCP NewReno**: Mejora TCP Reno para manejar mejor múltiples pérdidas de paquetes en una ventana.

- **TCP Cubic**: Utilizado por defecto en Linux, es un algoritmo más avanzado que optimiza el uso del ancho de banda en redes con alta capacidad y alta latencia, y es más agresivo en su recuperación de congestión.

- **TCP BBR** (Bottleneck Bandwidth and Round-trip propagation time): Un algoritmo más moderno que no depende de la pérdida de paquetes para detectar congestión, sino que mide la capacidad disponible en la red para ajustar la tasa de transmisión.paquetes.

Conceptos importantes en el control de congestion

- C**ongWin (Ventana de congestión):** Es la ventana que controla cuántos bytes puede enviar el transmisor antes de recibir un acuse de recibo (ACK). Su tamaño es dinámico y depende de las condiciones de la red. TCP utiliza eventos como ACKs duplicados o timeouts para ajustar esta ventana.

- **RcvWindow (Ventana de recepción):** Informa al transmisor cuántos bytes el receptor puede manejar en su buffer. Esto controla el flujo de datos hacia el receptor, asegurando que no se sobrecargue.

Control de Errores

El control de errores es fundamental en la comunicación a través de canales no confiables, como el protocolo IP, que opera bajo el modelo de "mejor esfuerzo". A continuación, se presentan los mecanismos y esquemas utilizados para gestionar los errores en la transmisión de datos:

Mecanismos de Control

ARQ (Automatic Repeat reQuest):

- Utiliza números de secuencia y confirmaciones (ACKs) para validar la correcta recepción de los datos.
- Implementa un sistema de retransmisiones automáticas en caso de que los datos se pierdan o se reciban con errores.
- Se utilizan temporizadores (RTO - Retransmission Timeout) para gestionar las retransmisiones.

Checksum/CRC:

- Herramientas que verifican la integridad de los datos, asegurando que no haya errores de bits en la recepción.

Esquemas de Control de Errores

- **Stop & Wait (S&W):** El emisor envía un segmento y espera la confirmación del receptor antes de enviar el siguiente. Puede ser ineficiente, ya que solo permite enviar un dato a la vez. En caso de no recibir un ACK, se reinicia el proceso de envío.

- **Pipelining/Sliding Window:** Permite enviar múltiples segmentos sin esperar confirmaciones de cada uno, aumentando la eficiencia. Se utiliza una ventana de transmisión que define cuántos segmentos se pueden enviar sin recibir ACKs. Se pueden generar confirmaciones negativas (NAK) en caso de errores.

- **Go-Back-N (GBN):** En este esquema, el emisor puede enviar varios segmentos, pero solo confirma los que se reciben en orden. Si se detecta un error, se retransmiten todos los segmentos a partir del segmento fallido. Se utiliza la técnica de piggybacking para optimizar las confirmaciones.

- **Selective Repeat (SR):** Similar a GBN, pero permite la recepción de segmentos fuera de orden. Se confirmarán solo aquellos segmentos que se hayan recibido correctamente, y los demás se retransmitirán individualmente.

"A medida que se utilizan esquemas más complejos, como GBN y SR, se mejora la eficiencia y la utilización del ancho de banda."

Multiplexación y Demultiplexación

Es el proceso por el cual el nivel de transporte maneja múltiples conexiones simultáneas. La multiplexación ocurre cuando varios flujos de datos de diferentes aplicaciones se agrupan en una sola conexión de red, y la demultiplexación es el proceso inverso, es decir, dividir esos flujos de datos en conexiones individuales basadas en los números de puerto.

Multiplexación:

- ⑩ Es el proceso mediante el cual múltiples flujos de datos de diferentes aplicaciones se combinan y se transmiten a través de una sola conexión física o lógica.
- En el **nivel de transporte**, esto significa que varias aplicaciones en un mismo dispositivo pueden enviar datos simultáneamente a través de una red utilizando un único enlace de red.
- ⑩ Cada flujo de datos es etiquetado con su número de puerto de origen (de la aplicación que envía los datos) y su número de puerto de destino (de la aplicación que recibe los datos), permitiendo que la red distinga entre los distintos flujos.

Demultiplexacion

- ⑩ Es el proceso inverso de la multiplexación. Ocurre cuando un host recibe datos desde la red y los separa en diferentes flujos de datos de acuerdo con su número de puerto, dirigiéndolos a la aplicación correcta.
- En el **nivel de transporte**, esto se hace utilizando los números de puerto de origen y destino que fueron añadidos en la multiplexación.
- ⑩ Así, aunque un dispositivo reciba datos desde diferentes fuentes a través de una sola conexión, puede dirigir los datos a las aplicaciones correspondientes basándose en los números de puerto.