Resumen Tema 3

Capa de transporte e internet

Autor: @BlackTyson

Índice

1. Introducción

Funciones y servicios de la capa de transporte

- Comunicación extremo a extremo
- Realiza la multiplexación/demultiplexación de aplicaciones (puerto)

Protocolo UDP

- Realiza la multiplexación/demultiplexación de aplicaciones.
- Ofrece servicio no orientado a conexión y no fiable.

Protocolo TCP

- Realiza la multiplexación/demultiplexación de aplicaciones
- Ofrece servicio orientado a conexión fiable, que incluye
  - Control de errores y flujo.
  - Control de congestión.
  - Control de la conexión.

2. Protocolo de datagrama de usuario UDP

Características

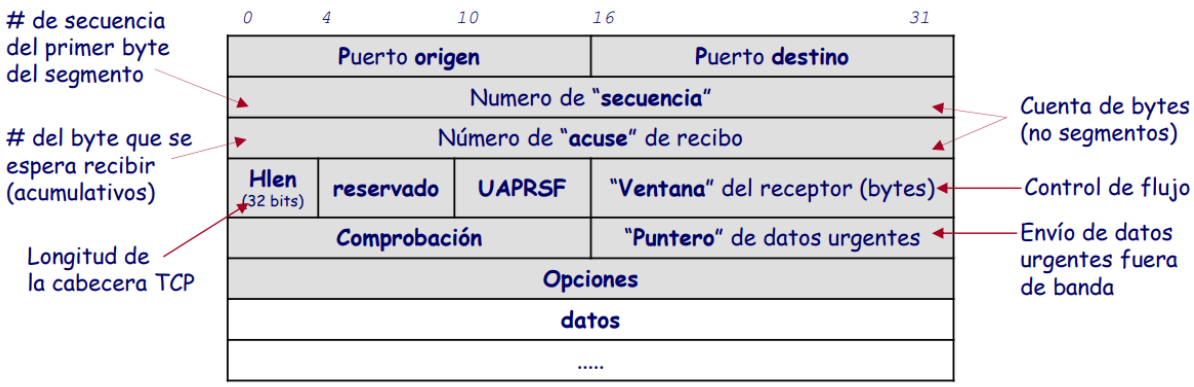
- Funcionalidad **best-effort**
- Servicio **no orientado a conexión**, no existe handshaking.
- **No hay retardos** de establecimiento y cada TPDU es independiente.
- Servicio **no fiable**, puede haber pérdidas de información.
- **No** hay garantías de **entrega ordenada**.
- **No** hay **controles**, ni de congestión ni de flujo. La **entrega** es lo más **rápida** posible.
- Realiza la multiplexación/demultiplexación. **Transporta** las TPDU al proceso correcto **mediante** los **puertos**, siendo estos algunos de los preasignados:
  - **53** (**DNS**): Servicio de nombres de dominio.
  - **69 (TFTP)**: Transferencia simple de ficheros.
  - **123 (NTP)**: Protocolo de tiempo de red.
  - **161 (SNMP)**: Protocolo simple de administración de red
  - **520 (RIP)**: Protocolo de información de encaminamiento
- Se usa frecuentemente para **aplicaciones multimedia**.
- Cada segmento UDP se **encapsula** en un **datagrama IP**.

3. Protocolo de control de transmisión TCP

3.1. Características

- Servicio **punto a punto**, inútil en comunicaciones multicast.
- Implica servicio **orientado a conexión**, estado común entre emisor y receptor **(handshaking)** por medio de **3 fases**.
  1. **Establecimiento.**
  2. **Intercambio de datos**.
  3. **Cierre**
- Garantiza la **entrega ordenada** de información.
- Transmisión **full-duplex**, por lo que permite **comunicación simultánea** en ambas direcciones.
- Incluye **mecanismos** para **detección y recuperación de errores** (ARQ) con **confirmaciones positivas ACKs** y tiemouts adaptables.
- Servicio **fiable** mediante **control de congestión y flujo**.
- Usa incorporación de **confirmaciones** (**piggybacking**)
- Para mejorar **eficacia**, se adapta de manera **dinámica** a las **condiciones** de red

Segmentación TCP



- **Puerto origen y puerto destino:** Identifican el punto de inicio y destino de la comunicación en el dispositivo emisor y receptor, respectivamente.
- **Número de secuencia:** Indica el número correspondiente al primer byte de datos dentro del segmento, permitiendo el ordenamiento y seguimiento de los datos enviados.
- **Número de acuse de recibo:** Es un valor acumulativo que especifica el número del siguiente byte que el emisor espera recibir del receptor, facilitando la confirmación de datos ya enviados.
- **Hlen (Header Length):** Indica la longitud de la cabecera TCP en múltiplo de palabras de 32 bits.
- **Ventana del receptor:** Define el número de bytes que el receptor está dispuesto a aceptar, utilizado para el control de flujo.
- **Comprobación (Checksum):** Un código que verifica la integridad de los datos del segmento, garantizando que no se hayan producido errores durante la transmisión.
- **Puntero de datos urgentes:** Indica la posición en la que finalizan los datos marcados como urgentes, facilitando el manejo de información prioritaria.
- **Opciones:** Campo opcional utilizado para ajustes o configuraciones adicionales en la comunicación.
- **Datos:** Contenido real que se transmite al destino.

Cada segmento TCP se encapsula dentro de un paquete IP (también denominado datagrama), lo que permite su transmisión a través de la red y su manejo por las distintas capas del modelo TCP/IP.

3.2. Multiplexación/demultiplexación

El objetivo es **transportar TPDUs** al proceso correcto por **medio de puertos**. Existen algunos preasignados:

- **20 (FTP-DATA)**: Transferencia de ficheros :datos
- **21 (FTP)**: Transferncia de ficheros: controñ
- **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

3.3. Control de conexión

- TCP ofrece **servicio orientado a conexión**
- Fases:
  - **Establecimiento de conexión** (sincronizar)
  - **Intercambio de datos** (full-duplex)
  - **Cierre de conexión** (liberar recursos)

Diagrama Handshaking

El proceso de establecimiento de una conexión TCP utiliza el mecanismo conocido como **three-way handshake**, que consta de tres pasos básicos para sincronizar los números de secuencia y establecer una conexión confiable entre el cliente y el servidor. Este mecanismo se puede dividir en los siguientes pasos, como se ilustra en el diagrama:

1. **Primera fase (SYN):**
   - El cliente inicia la conexión enviando un segmento con el bit `SYN=1`.
   - Incluye un número de secuencia inicial (`secuencia = X`), elegido aleatoriamente.
   - Este paso se denomina **apertura activa** por parte del cliente.
2. **Segunda fase (SYN-ACK):**
   - El servidor responde al cliente con un segmento donde `SYN=1` y `ACK=1`.
   - El número de `acuse` (`acuse = X+1`) confirma que ha recibido el número de secuencia inicial del cliente.
   - El servidor incluye su propio número de secuencia inicial (`secuencia = Y`).
   - Este paso se denomina **apertura pasiva** por parte del servidor.
3. **Tercera fase (ACK):**
   - El cliente confirma la recepción del segmento `SYN-ACK` enviando un segmento con `ACK=1`.
   - El número de `acuse` ahora es `acuse = Y+1`, confirmando el número de secuencia del servidor.
   - Con esto, la conexión queda establecida, y ambas partes están listas para intercambiar datos.

Campos involucrados:

- **Bit S (SYN):** Indica la solicitud para establecer la conexión.
- **Campo secuencia:** Valor inicial aleatorio para los números de secuencia.
- **Campo acuse:** Utilizado para confirmar la recepción de datos.
- **Bit A (ACK):** Marca la confirmación de recepción en un segmento.

Este procedimiento asegura que ambas partes acuerden los números de secuencia iniciales y puedan comenzar la transmisión de datos de manera confiable.

Números de secuencia

- Campos de 32 bytes. No empieza normalmente en 0, sino en un valor denomaminado ISN elegido .ªl azar".
- El ISN es **elegido por el sistema**, y se sugiere utilizar un contado entero que incremente en 1 cada 4 μs.
- La elección de ISN **no es fiable** frente a sabotajes, ya que es **fácil averiguarlo**.
- TCP incrementa el INS de cada segmento según los bytes del segmento anterior, salvo con los **flags SYN y FIN que incrementan en 1.**
- Los segmentos **ACK no incrementan el ISN**

Diferencias entre incidencias

El establecimiento de una conexión TCP puede desarrollarse de diferentes maneras dependiendo de las circunstancias. A continuación, se explican tres casos relevantes:

1. **Caso sin incidencias (normal):**
   - En este caso, la conexión sigue el proceso estándar del **three-way handshake**.
   - TCP A envía un segmento `SYN` con un número de secuencia inicial (`seq = 100`).
   - TCP B responde con un segmento `SYN-ACK`, indicando `seq = 300` y `ack = 101`, confirmando la recepción del segmento de TCP A.
   - Finalmente, TCP A responde con un segmento `ACK` (`seq = 101, ack = 301`), estableciendo la conexión.
2. **Caso de conexión simultánea:**
   - Aquí, ambos extremos intentan establecer la conexión al mismo tiempo.
   - TCP A envía un `SYN` (`seq = 100`) mientras TCP B también envía un `SYN` (`seq = 300`).
   - Ambos reciben los segmentos `SYN` del otro, respondiendo con `SYN-ACK` (`ack = 301` para TCP A y `ack = 101` para TCP B).
   - Finalmente, ambos intercambian `ACK`, completando la conexión simultáneamente.
3. **Caso con `SYN` retrasados y duplicados:**
   - En este caso, se generan duplicados debido a retransmisiones o retrasos en la red.
   - TCP A envía un `SYN` (`seq = 90`) inicialmente, pero debido a un `timeout`, reintenta con un nuevo `SYN` (`seq = 100`).
   - TCP B recibe ambos `SYN` y responde en consecuencia, causando un estado intermedio con `SYN-ACK` duplicados.
   - Finalmente, TCP A y TCP B sincronizan los números de secuencia correctamente, descartando los duplicados, y establecen la conexión (`seq = 400, ack = 101`).

Control de conexión (cierre)

El cierre de una conexión TCP es un proceso que asegura la correcta liberación de recursos y la finalización ordenada de la comunicación entre dos extremos. Si no se realiza adecuadamente, pueden ocurrir pérdidas de información. Existen dos tipos de cierre:

- **Cierre activo:** Iniciado por uno de los extremos enviando un segmento con el bit `FIN` activado.
- **Cierre pasivo:** El otro extremo responde aceptando el cierre, y eventualmente también cierra su lado de la conexión.

Campos involucrados:

- **Bit F (FIN):** Indica la solicitud para finalizar la conexión.
- **Campo secuencia:** Utilizado para rastrear el estado de los segmentos.
- **Campo acuse:** Confirma la recepción de los segmentos.
- **Bit A (ACK):** Marca la confirmación de recepción.

Caso normal:

1. El extremo que inicia el cierre (TCP A) envía un segmento con `FIN` (`seq = 100, ack = 300`).
2. El otro extremo (TCP B) responde con un `ACK` confirmando el cierre (`seq = 300, ack = 101`) y luego envía su propio `FIN` para cerrar su lado de la conexión.
3. Finalmente, TCP A envía un último `ACK` (`seq = 101, ack = 301`), completando el cierre de la conexión.

Tras el cierre, TCP A entra en el estado `TIME-WAIT`, que dura **2 MSL** (Maximum Segment Lifetime, típicamente 2 minutos) para garantizar que no queden segmentos duplicados en la red.

Diagrama de estados finitos TCP: El automáta de estados finitos del protocolo TCP describe todas las transiciones posibles entre los diferentes estados durante la conexión y el cierre. Las principales etapas del cierre incluyen:

- `CLOSE-WAIT`: Espera el cierre del lado pasivo tras recibir un `FIN`.
- `FIN-WAIT-1`: El extremo activo espera el `ACK` de su `FIN`.
- `LAST-ACK`: El extremo pasivo espera el `ACK` tras enviar su `FIN`.
- `TIME-WAIT`: El extremo activo espera un tiempo para evitar problemas con paquetes duplicados antes de liberar completamente los recursos.

3.4. Control de errores y de flujo

Características

- **Mejoran el rendimiento** mediante ventanas deslizantes.
- **Controlan errores** mediante esquemas ARQ con confirmaciones positivas (ACKs) y acumulativas.
- Campos involucrados:
  - **Secuencia**: offset dentro del mensaje
  - **Acuse**: número de byte esperado en el receptor
  - **BIT a (ACK)** del campo de control
  - **Campo comprobación**: checksum de todo el segmento y uso de pseudo-cabecera TCP

Control de errores

Evento	**Acción del TCP receptor**
Llegada ordenada de segmento, sin discontinuidad, todo lo anterior confirmado.	Retrasar el ACK hasta 500 ms esperando el siguiente segmento. Si no llega, enviar un ACK.
Llegada ordenada de segmento, sin discontinuidad, pero con ACK retrasado.	Enviar de inmediato un único ACK acumulativo para confirmar los datos recibidos.
Llegada desordenada de segmento con un número de secuencia mayor al esperado.	Enviar un ACK duplicado indicando el número de secuencia del próximo byte esperado.
Llegada de un segmento que completa una discontinuidad parcial o total.	Confirmar con un ACK inmediato si el segmento completa el extremo inferior de la discontinuidad.

Estimación de timeouts

- Mayor que el tiempo de ida y vuelta
- Si es demasiado pequeño: tiemout prematuro $\Longrightarrow$ retransmisiones innecesarias
- Si es demasiado grande: reacción lenta $\Longrightarrow$ baja eficacia.
- Para situaciones cambiantes $\Longrightarrow$ adaptarse dinámicamente

Formula estimacion timeouts

El protocolo TCP utiliza un algoritmo para calcular de manera dinamica los **timeouts**, basado en el tiempo de ida y vuelta (**RTT, Round Trip Time**) y su desviacion. Las formulas utilizadas son:

$$RTT_{nuevo} = (1 - \alpha) \cdot RTT_{viejo} + \alpha \cdot RTT_{medido}, \quad \alpha \in [0, 1]$$

Esta formula actualiza el RTT estimado combinando el valor previo con el RTT medido recientemente, ponderados segun el factor α.

$$Desviacion_{nueva} = (1 - \beta) \cdot Desviacion_{vieja} + \beta \cdot |RTT_{medido} - RTT_{nuevo}|, \quad \beta \in [0, 1]$$

Aqui, la desviacion calcula la variacion promedio entre el RTT medido y el RTT estimado. Esto ayuda a ajustar la confianza en el tiempo calculado.

$$Timeout = RTT_{nuevo} + 4 \cdot Desviacion_{nueva}$$

El timeout final es el RTT actualizado mas un margen de seguridad basado en la desviacion multiplicada por 4.

Problema con ACKs repetidos: Cuando se reciben **ACKs duplicados**, puede surgir confusion al interpretar los tiempos. Para esto, se utiliza el **Algoritmo de Karn**, que propone dos reglas:

- Solo actualizar el RTT para segmentos que no sean ambiguos.
- En caso de retransmitir un segmento, duplicar el timeout anterior:

$$t_{out_{nuevo}} = \gamma \cdot t_{out_{viejo}}, \quad \gamma = 2$$

De esta forma, TCP maneja los timeouts de forma eficiente y confiable, incluso en condiciones de red adversas.

Control de flujo

- Evita que el emisor sature al receptor con el envío de mucha información o muy rápida.
- Esquema crediticio: el erceptor informa al emisor sin esperar respuesta.
- Se utiliza el campo ventana del receptor en la cabecera TCP
- El emisor utiliza la ventana útil:

```
venta útil emisor = ventana ofertada receptor - bytes en tránsito
```

Temporizador de persistencia

El **temporizador de persistencia** es un mecanismo utilizado por el protocolo TCP para manejar situaciones en las que la ventana de recepción del receptor se cierra temporalmente (por ejemplo, cuando el buffer del receptor está lleno) y no puede aceptar más datos. Este temporizador asegura que la conexión no quede bloqueada de manera indefinida debido a la pérdida de mensajes de actualización de la ventana (anuncios de ventana).

Funcionamiento:

- Si el receptor anuncia una ventana de tamaño `W=0` (es decir, no puede recibir datos adicionales), el temporizador de persistencia se activa en el lado emisor.
- El emisor periódicamente reenvía un pequeño segmento para "sondear.ª]l receptor y determinar si la ventana ha sido actualizada.
- Una vez que el receptor puede aceptar más datos, actualiza el tamaño de la ventana (`W >0`) y el emisor reanuda la transmisión.

Posibles problemas:

- **Síndrome de la ventana tonta:** Ocurre cuando el receptor permite el envío de segmentos muy pequeños debido a un espacio reducido en su buffer, lo que afecta la eficiencia de la conexión.

Mejoras y características relacionadas:

- **Ventana optimista (RFC 813):** Proporciona una solución al síndrome de la ventana tonta al optimizar el manejo del buffer y la actualización de la ventana.
- **Entregas "no ordenadas":** Utilizando el bit `URG` y el campo puntero, se pueden realizar entregas urgentes sin necesidad de esperar el orden de los datos.
- **Entregas inmediatas:** El bit `PSH` puede ser usado para solicitar que los datos lleguen inmediatamente a la aplicación del receptor.

Este mecanismo es esencial para garantizar la confiabilidad y evitar bloqueos en situaciones donde las actualizaciones de ventana se pierden o el receptor tiene problemas temporales de buffer.

3.5. Control de congestión

Fundamentos

- Surge de la insuficiencia de recursos.
- Protege a la red debido a las limitaciones.
- Puede tener naturaleza adelante-atrás
- La congestión se manifiesta en retrasos en la ACKs y/o pérdidas de segmentos
- Solución extremo a extremo: limitar de forma adaptable el tráfico generado por el emisor, sin perder eficacia, imparcialidad y estabilidad, liimtando el tamaño de la ventana de emisión.

Control de Congestion TCP-Tahoe

El algoritmo **TCP-Tahoe** utiliza dos ventanas y un umbral para controlar la congestión en la red:

$$Bytes\_permitidos\_enviar = \text{mín}\{\text{VentanaCongestion, VentanaDelReceptor}\}$$

Definiciones:

- **VentanaDelReceptor:** Determina el número máximo de datos que el receptor puede aceptar (se ajusta dinámicamente según los `ACKs`).
- **VentanaCongestion:** Controla cuántos datos el emisor puede enviar sin saturar la red. Inicialmente es igual a 1 (en número de segmentos).
- **Umbral:** Un límite que regula el cambio entre las fases de crecimiento exponencial y lineal.

Fases del algoritmo:

- **Inicio lento:** Mientras `VentanaCongestion <umbral`, por cada `ACK` recibido se incrementa la ventana de congestión (`VentanaCongestion++`), resultando en un **crecimiento exponencial**.
- **Prevención de congestión:** Si `VentanaCongestion >umbral`, el incremento ocurre de forma más lenta (uno por ciclo de recepción de `ACKs`), resultando en un **crecimiento lineal**.
- **Si ocurre congestión:** En caso de `timeout`, el umbral se reduce a la mitad (`umbral = VentanaCongestion/2`) y `VentanaCongestion` se reinicia a 1.

Este algoritmo asegura un control eficiente de la congestión al combinar crecimiento controlado y reacción rápida ante problemas en la red.