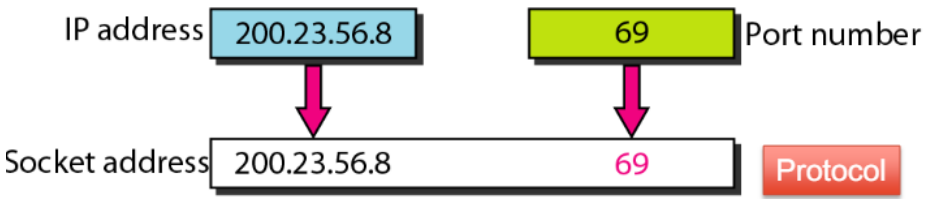
TEMA 4: SERVICIOS BÁSICOS PARA EL NIVEL DE TRANSPORTE EN INTERNET

1. PROTOCOLOS DE TRANSPORTE

Su objetivo es proporcionar servicios para permitir la comunicación lógica entre procesos que se ejecutan en la capa de aplicación, arreglar las carencias que presenta el protocolo de nivel y ofrecer al nivel superior una comunicación extremo a extremo fiable. Se ejecutan en los nodos terminales que se están comunicando. Los disponibles en TCP/IP son TCP, UDP, STCP.

1.1 Servicios que ofrecen

Direccionamiento: identificación de procesos. Es necesario tener una dirección de nivel y de transporte denominada TSAP. Permite elegir entre los múltiples procesos que se ejecutan en una máquina destino. Los números de puerto son enteros sin signo de 16 bits [0 – 65535] y hay tres tipos: bien conocidos, registrados y dinámicos. El proceso cliente define un número de puerto aleatorio (puerto efímero) y el proceso servidor define el suyo con un número de puerto bien conocido (estándar o no estándar). La comunicación proceso a proceso necesita tres identificadores: dirección IP, protocolo y número de puerto. La combinación se llama "dirección de socket" <protocolo, IP, puerto>. Un protocolo de transporte necesita un par de direcciones de sockets (para cliente y servidor).



Seguimiento de la comunicación individual entre procesos origen y destino:

- Protocolos no orientados a la conexión: UDP.
- Protocolos orientados a la conexión: TCP. Transporte fiable de datos y control de congestión y control de fujo.

Control de errores: controla los errores extremo a extremo. Asegura que todo el mensaje llega al nivel del receptor sin errores. Los errores se corrigen con retransmisiones.

Control de flujo: controla el flujo extremo a extremo. Permite al receptor controlar la velocidad (cantidad) de datos enviados por el emisor. El flujo se controla mediante técnicas basadas en ventanas deslizantes.

Segmentación y reensamblado de datos: dividir los datos de usuario en varios segmentos.

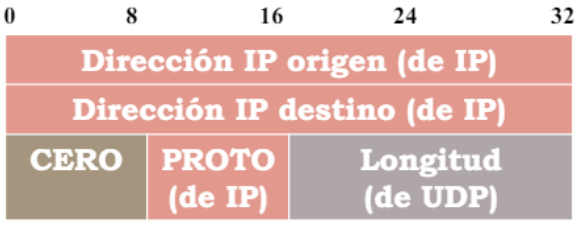
Multiplexación: Multiplexar varias conexiones de transporte en una sola conexión de red. Dos tipos:

- Multiplexación (relación muchos-uno): el protocolo acepta el mensaje de distintos procesos diferenciados por el nº de puerto asignado.
- Demultiplexación (relación uno-muchos): el protocolo recibe datagramas del nivel de red y entrega cada mensaje al proceso adecuado basándose en el nº de puerto.

1.2 El protocolo UDP

Envía datagramas, no está orientado a la conexión, no es fiable y no incorpora control de gestión. Sus servicios son:

- **Sendto** (puerto origen, puerto destino, datos)
- **Recvfrom** (puerto origen, puerto destino, datos)

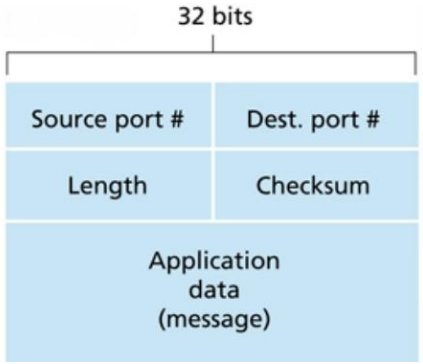


Y sus extremos son:

- Extremo 1: <udp, IPlocal, Plocal>
- Extremo 2: <udp, IPremoto, Premoto>

Cabecera:

- Puerto origen y destino
- Longitud total
- Checksum todo el datagrama UDP y pseudo cabecera IP
- Datos del nivel superior



Funcionamiento de UDP: no hay control de flujo, ni de errores, ni fase de conexión ni desconexión. El mensaje es encapsulado en un datagrama IP.

Colas: Cada puerto lleva asociada dos colas, la de entrada y la de salida. UDP saca mensajes de las colas de salida, añade la cabecera y entrega IP. Al llegar un mensaje, UDP comprueba si hay cola de entrada para el puerto. Si hay cola deposita el mensaje al final de la cola, pero sino descarta el datagrama y pide a ICMP que envíe un mensaje de puerto no alcanzable. Todos los mensajes enviados por un mismo proceso son encolados en la misma de salida independientemente del destino. Y todos los mensajes recibidos por un mismo proceso son encolados en la misma cola de entrada independientemente del origen.

Ventajas frente a TCP: es más rápido y tiene menor penalización en la transmisión al tener una cabecera más corta.

Uso: envío d mensajes pequeños, protocolos de nivel de aplicación tipo mensaje/respuesta, protocolos de nivel de aplicación tolerantes a fallos y hay necesidad de envíos Multicast/Broadcast.

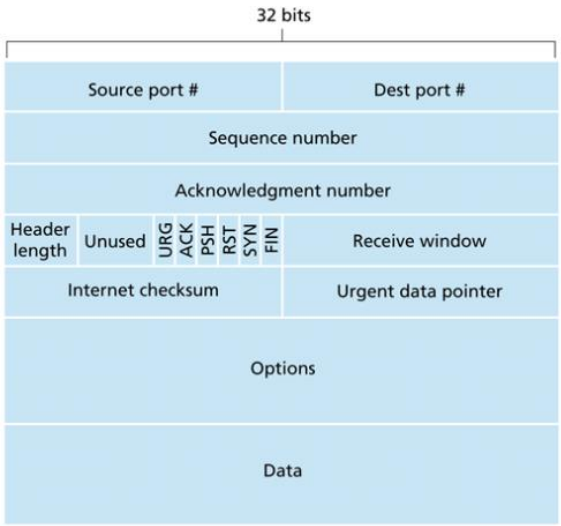
1.3 El protocolo TCP

Envía segmentos TCP. Es un protocolo orientado a la conexión, sus conexiones son full-duplex, tiene canales punto a punto (no broadcasting ni multicasting), ofrece un servicio fiable y tiene una conexión orientada a flujos de bytes.

Buffering: los segmentos TCP se envían a un buffer de envío. TCP enviará los datos en el buffer de envío. Existe un buffer de recepción donde los datos se lee a petición del protocolo de la capa superior.

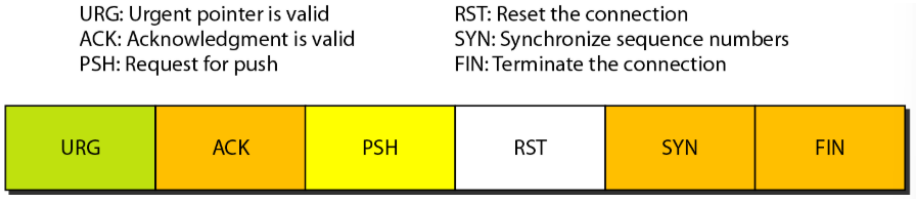
MSS (Maximum Segment Size): máxima cantidad de datos que puede llevar un segmento TCP. Se negocia durante el establecimiento de conexión y depende de la implementación.

Estructura del segmento TCP: cabecera de 20 bytes + opciones.



Cabecera:

- Puerto origen y destino: 16 (TSAPs que identifican los extremos de la conexión)
- Número de secuencia: 32
- Número de confirmación: 32
- Longitud de la cabecera TCP: 4 (en palabras de 32 bits)
- Tamaño de la ventana: 16 (para control de flujo)
- Bit ACK: el valor del campo ACK es válido
- Bits RST, SYN, FIN: establecimiento/fin de conexiones



Número de secuencia: TCP utiliza el número de byte para numerar los datos. La numeración no comienza en 0, sino en un número aleatorio [0 → $2^{32} - 1$]. El número de secuencia es el del primer byte que transporta el segmento (contando desde el valor de inicio). El número ACK (cuando el flag está inactivo) indica el siguiente byte que se espera recibir.

Si un segmento no lleva datos de usuario no consume número de secuencia, excepto los segmentos de control (SYN y FIN de conexiones).

Establecimientos de conexiones: se usa un protocolo de negociación a tres bandas.

Transferencia de datos TCP:

- ACKs: número de secuencia del siguiente byte que se espera recibir. ACK = SEQ + long(datos)
- Transmisión de datos bidireccional: ambos extremos pueden enviar datos y confirmaciones. La confirmación se incluye con los datos (piggybacking).
- Envío inmediato de datos (pushing, PSH): el emisor no debe esperar a completar el buffer. El receptor entrega los datos lo antes posible.

Finalización de las conexiones:

- Cierre en tres pasos: cualquiera de los dos extremos inicia la desconexión.
- Semicierre: uno de los extremos deja de enviar datos mientras sigue recibiendo. Actualmente incluso sin tener datos pendientes que enviar, se suele utilizar este esquema.

1.4 Control de flujo TCP:

La ventana deslizante: se usa una ventana deslizante para hacer la transmisión más eficiente. Las ventanas son orientadas a byte y son de tamaño variable (los bytes dentro de la ventana se pueden enviar sin preocuparse de la confirmación). La ventana de recepción es el valor notificado por el segmento opuesto en un segmento que contiene una confirmación y representa el número de bytes que puede aceptar antes de que su almacén se desborde y tenga que descartar datos. Hay 4 ventanas para una conexión full-dúplex.

Silly Window Syndrome: ocurre cuando el emisor o el receptor envía o procesa datos de manera muy lenta lo que provoca ajustes excesivos en las ventanas de transmisión y recepción. Cuando la cantidad de datos de usuario enviados son menores que el tamaño de la cabecera de TCP (20 bytes) decimos que la transacción está sufriendo el Silly Window Syndrome. Está provocado por implementaciones pobres de TCP.

Algoritmo de Nagle: es la solución al síndrome de Silly Window de lado del emisor. Es un método eficiente para evitar la transmisión de segmentos demasiado pequeños.

- Si no existen datos esperando ser confirmados → `write(datos)` independientemente del tamaño de los datos.

- Si hay datos pendientes de ser confirmados → no se envían nuevos segmentos de datos hasta que o se reciba la confirmación de todos los datos pendientes de confirmación o la longitud de los datos a transmitir sea mayor que MSS.

- Siempre respetando que long(datos) ← tamaño ventana de recepción.

Solución de Clark: solución al síndrome de Silly Window en lado del receptor. Tras cerras una ventana, el receptor no debe anunciar una ventana distinta de 0 hasta que no pueda anunciar una ventana de tamaño suficiente.

```
T = min (MSS, buffer_recepción / 2)
```

1.5 Control de errores TCP:

TCP incluye mecanismos para detectar y corregir errores por segmentos: corruptos, perdidos, fuera de orden y duplicados. Los mecanismos son:

- Suma de comprobación de 16 bits: detecta segmentos corruptos que son descartados y considerados perdidos.

- Confirmación acumulativa (positiva): indican la recepción de segmentos de datos. Los segmentos ACK no se confirman.

- Segmentos fuera de orden: antes se descartaban los segmentos fuera de orden, que eran retransmitidos. Actualmente se almacenan temporalmente y se arcan como “fuera de orden” hasta que llega el segmento perdido. Ningún segmento fuera de orden es entregado al proceso destino.

- Retransmisión (con temporizadores): los segmentos ACK no se retransmiten. Se retransmite en dos ocasiones:

  - Después de un plazo de retransmisión (RTO): el emisor mantiene un temporizador RTO por conexión, cuando vence este temporizador envía el primer segmento sin confirmar. No se activa un RTO para los segmentos que sólo confirman. El valor del temporizador RTO es dinámico y se actualiza en base al tiempo de ida y vuelta (RTT): usa su medida (SRTT) y varianza (RTTVAR).

  - Retransmisión después de tres segmentos ACK: se pierde un segmento y se reciben algunos fuera de orden (lo que es un problema si el almacén del receptor es limitado). Se reciben 3 ACKs duplicados para el mismo número de secuencia y se reenvía el segmento perdido inmediatamente (retransmisión rápida).

1.6 Control de congestión:

Es diferente al control de flujo ya que este no envía más paquetes de los que puede recibir el receptor mientras que el control de congestión trata de no enviar más paquetes en una parte de la subred.

Control de congestión en TCP: la detección de congestión se basa en observar el aumento del RTT de los segmentos confirmados, respecto a anteriores segmentos y de la finalización de temporizadores de retransmisión. Se usa la ventana de congestión (cwnd) para controlar la velocidad de envío.

Tamaño de la ventana útil = mínimo (rwnd, cwnd)

Algoritmo de congestión de TCP (RFC 5681):

- Slow start (inicio lento): incremento exponencial de ventana. El tamaño de la ventana de congestión incrementa en 1 MSS cada ves que llega 1 ACK.

  TCP comienza con cwnd = 1 MSS y envía 1 segmento. Se recibe una confirmación, se libera 1 MSS del buffer de recepción y se incrementa cwnd en 1 MSS por cada ACK.

  cwnd = cwnd + 1 MSS = 2 MSS

  TCP envía 2 segmentos (tantos como le permite su ventana). Se reciben 2 confirmaciones, se liberan 2 MSS del buffer de recepción y se incrementa cwnd en 1 MSS por cada ACK.

  cwnd = cwnd + 2 MSS = 4 MSS

- Congestion avoidance (evitación de la congestión): incremento aditivo, es obligatorio. El tamaño de la ventana de congestión se incrementa en 1 si se confirma toda la ventana enviada. Si llega un ACK:

  cwnd = cwnd + (1 / cwnd) * MSS

- Fast recovery (opcional).

Cambios entre algoritmos: TCP ejecuta inicialmente slow start, pero cuando cwnd llega a cierto umbral pasa a congestion avoidance. En cualquiera de las fases, si hay un evento de pérdida se vuelve a slow start con cwnd = 1 MSS y el umbral se actualiza al cwnd / 2 que provocó la pérdida.

Protocolo STCP: Stream Control Transmission Protocol, fue definido por SIGTRAN de IETF (2000). Está orientado al mensaje y a la conexión, es confiable, con control de flujo y de congestión y secuenciación. Permite opcionalmente el envío de mensajes fuera de orden. Entre sus ventajas están el multihoming (posibilidad de varias IP en cada extremos de la comunicación), la selección y monitorización de caminos (múltiples flujos) y mecanismos de validación y protección contra ataques. Está implementado en diversos sistemas tipo UNIX.