

Grado en Ingeniería del Software
Doble Grado en Matemática Computacional e Ingeniería del Software
Doble Grado en Física Computacional e Ingeniería del Software



Redes de Ordenadores

Tema 3

Dr. Constantino Malagón Luque
Dr. Rafael Socas Gutiérrez

Septiembre 2024



3

Nivel de Transporte

1) Redes de Ordenadores e Internet

2) Nivel de Aplicación

3) Nivel de Transporte

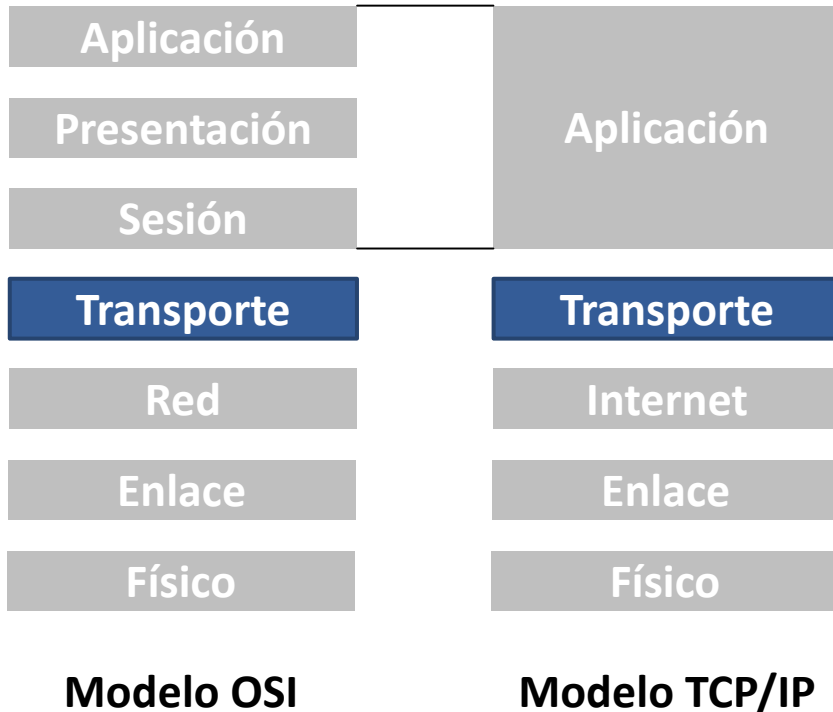
4) Nivel de Red

5) Nivel de Enlace: Redes de Acceso y LAN

6) Redes Inalámbricas y Redes Móviles

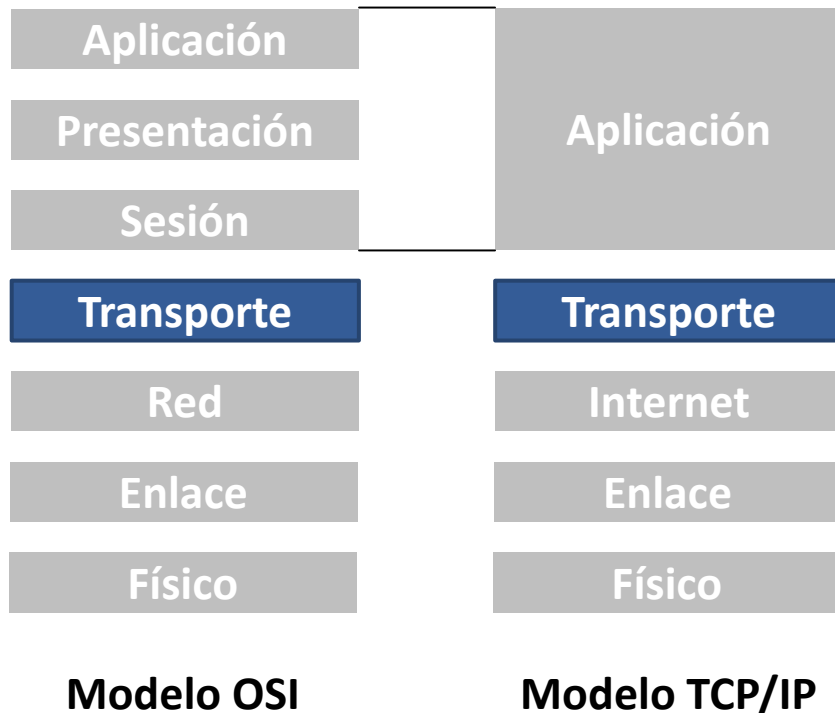
7) Seguridad en Redes de Ordenadores

The **Transport Layer** provides application-to-application communication



- **Capa de Transporte (Transport layer)**
- Servicio: transporta mensajes de la capa de aplicación entre diferentes hosts.
- **Protocolos: TCP y UDP**
- Los paquetes de transporte se llaman segmentos/datagramas.
- TCP (Transport Control Protocol): protocolo orientado a la conexión y confiable.
 1. Utiliza el Three-way handshake y Four-way handshake (Teardown) para establecer y finalizar la conexión.
 2. Y los flags junto con el sequence number y el ACK number como PDUs de control para regular esta conexión.
- UDP (User Datagram Protocol): protocolo no orientado a la conexión y no confiable (pero más rápido).
 1. No utiliza el Three-way handshake ni el Four-way handshake (Teardown) para establecer y finalizar la conexión.
 2. Ejemplos: DNS y DHCP.

The **Transport Layer** provides application-to-application communication



- **Capa de Transporte (Transport layer)**
- En función de los protocolos de la capa de aplicación se utiliza TCP o UDP y un puerto determinado.

<Protocolo Aplicación>:<Protocolo Transporte>:<Puerto>

HTTP : TCP : 80

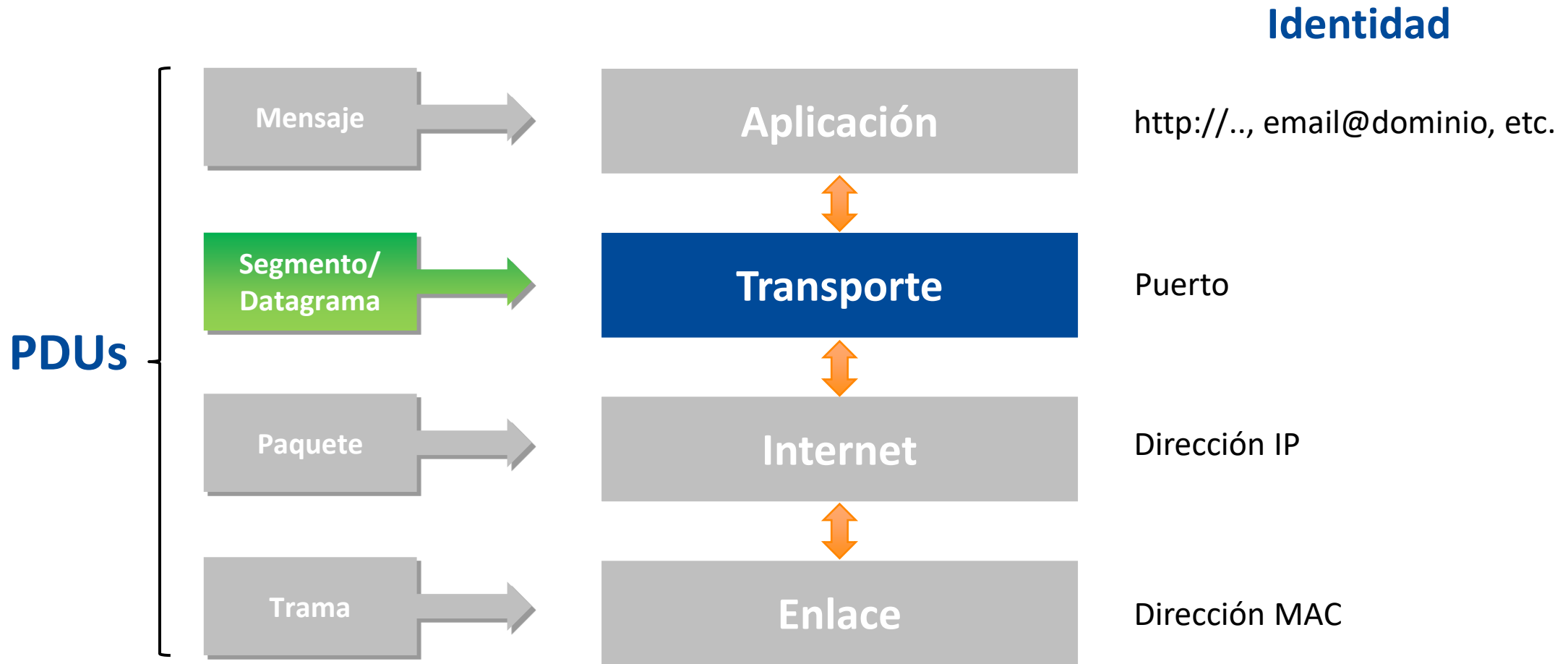
HTTPS : TCP : 443

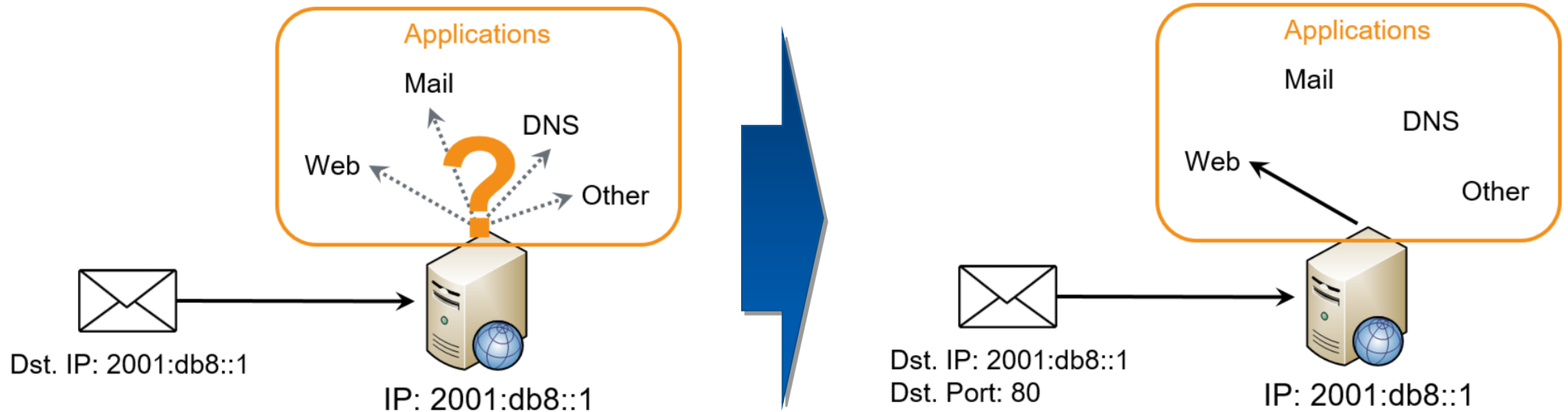
FTP : TCP : 20/21

Telnet : TCP : 23

SSH : TCP : 22

DNS : UDP : 53





Para identificar de forma única una conexión de Capa 4, no sólo se tienen en cuenta los puertos, sino los cinco parámetros siguientes:

- **Dirección IP Origen y Destino** (en los mensajes de capa 3).
- **Puerto Origen y Destino.**
- **Protocolo de Nivel 4** (p.e. TCP/UDP).

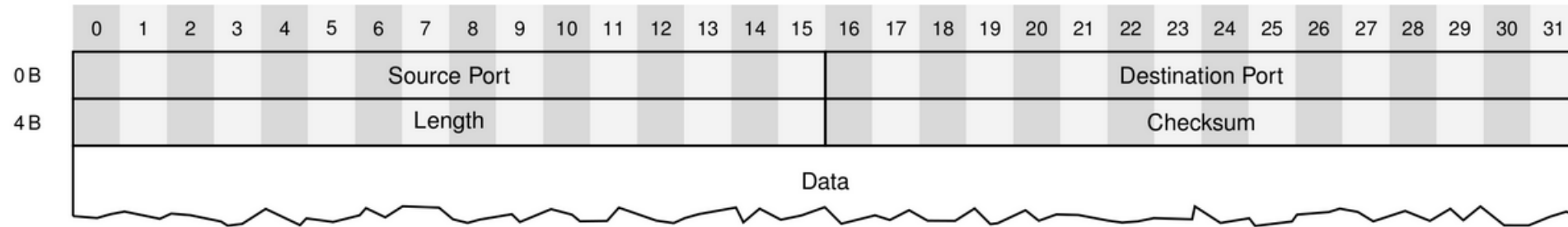
Este grupo de parámetros se llama **5-tupla e identifica de manera única una conexión.**

Los protocolos más comunes de Capa 4 son el **Transmission Control Protocol (TCP)** y el **User Datagram Protocol (UDP)**. Se usan para encapsular los datos recibidos de o para una aplicación. Los veremos con detalle más adelante. Por ahora, es suficiente con saber que ambos usan **Puertos** con una longitud de **16 bits**, lo que significa que se pueden usar hasta $2^{16}=65536$ **Puertos diferentes (por protocolo!)**.

Los puertos se dividen lógicamente en tres rangos:

- Los Puertos entre **0** y **1023** se definen como **Well Known Ports (Puertos Bien Conocidos)**: los Puertos Bien Conocidos son asignados a aplicaciones específicas y deberían ser usados solo por éstas. Por ejemplo, un web browser usa el Puerto 443 para hablar con un servidor web. Este Puerto está reservado para HTTPS, un protocolo que se usa para direccionar servidores web de forma segura. Otro ejemplo muy común es el Puerto 80 para HTTP. Estos Puertos **son asignados** por la **IANA** (Internet Assigned Numbers Authority. Con la estandarización de los Well Known Port los Puertos más importantes son reservados para las diferentes aplicaciones (protocolos de aplicación).
- Los Puertos entre **1024** y **49151** se denominan **Registered Ports (Puertos Registrados)**: los Puertos Registrados pueden asignarse a una aplicación/protocolo concreto por la IANA, pero en la práctica cualquiera podría usarlos.
- Finalmente, los Puertos entre **49152** y **65535** se denominan **Ephemeral Ports (Puertos Efímeros)**: los Puertos Efímeros pueden usarse libremente p.e. como Puerto **origen**. Estos Puertos normalmente no están asignados a aplicaciones concretas.

- El **User Datagram Protocol (UDP)** ofrece la funcionalidad mínima requerida por un protocolo de Capa 4: direccionamiento de aplicaciones. UDP es uno de los dos protocolos más importantes de la capa de transporte.
- UDP es un protocolo “**poco fiable**” (**unreliable**), lo que significa que no da ninguna garantía al emisor (o receptor) sobre un datagrama con respecto a la entrega. El término **Datagrama** se usa como nombre para los mensajes UDP. UDP es un protocolo **sin estado** (**stateless**), esto significa que cada datagrama es independiente.
- El protocolo de Internet (nivel 3) no garantiza ningún orden en los paquetes enviados. Eso significa que al enviar dos o más paquetes de la capa 3 éstos pueden llegar al destino en cualquier orden. UDP envía los paquetes a la aplicación en el orden recibido. Como veremos en TCP, un protocolo de capa 4 puede también ordenar los paquetes para entregarlos ordenados al nivel de aplicación. Con UDP el nivel a aplicación tiene que gestionar los paquetes desordenados, UDP tampoco comprueba si un **datagrama se ha perdido** por el camino.
- A pesar de sus inconvenientes, UDP es un **protocolo ampliamente usado** en Internet. Algunas de las aplicaciones que usan UDP son:
 - **VoIP** (Voice over IP).
 - **RIP** (Routing Information Protocol).
 - **DHCP** (IPv4 address configuration and more).
 - **DNS** (Domain Name System).
- UDP es utilizado por aquellas aplicaciones que prefieren **baja sobrecarga de red** sobre todo lo demás y tienen otros mecanismos para asegurarse de que los datagramas se procesan en el orden correcto si es necesario.



La cabecera UDP tiene **8 bytes** de longitud y consiste en **4 campos de 2 bytes (16 bits) cada uno** seguido de los datos (payload):

- **Puerto Origen:** especifica que aplicación envía el datagrama UDP.
- **Puerto Destino:** indica la aplicación en el host que recibe el mensaje.
- **Length:** contiene la longitud total del datagrama UDP en bytes, incluyendo la cabecera y los datos.
- **Checksum:** se utiliza para detectar errores de transmisión en el datagrama recibido.
- **Datos:** contiene la información procedente o para la capa de aplicación.

UDP es un **protocolo de la Capa de Transporte ligero** con muchas ventajas y desventajas. Dependiendo de los requisitos de las diferentes aplicaciones, las desventajas podrían no importar tanto y las ventajas podrían ser más importantes:

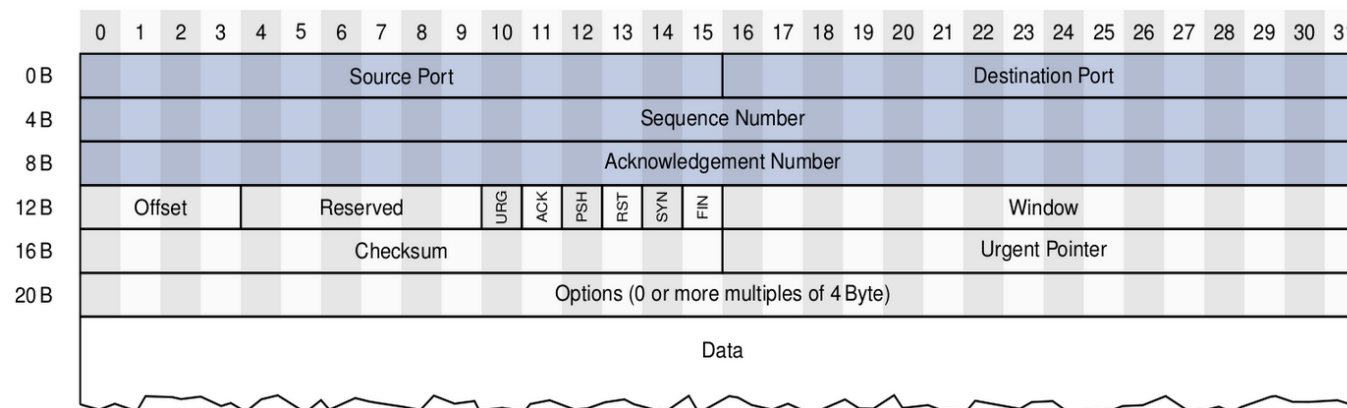
Ventajas:

- **Baja sobrecarga de red.**
- Simplemente “dispara y olvida” para los datagramas, **sin establecimiento de sesión, retransmisiones or reordinación** de datagramas.
- Muy adecuado para **aplicaciones en tiempo real** (p.e. juegos online, Voz sobre IP) donde no importa si algunos datagramas se pierden.

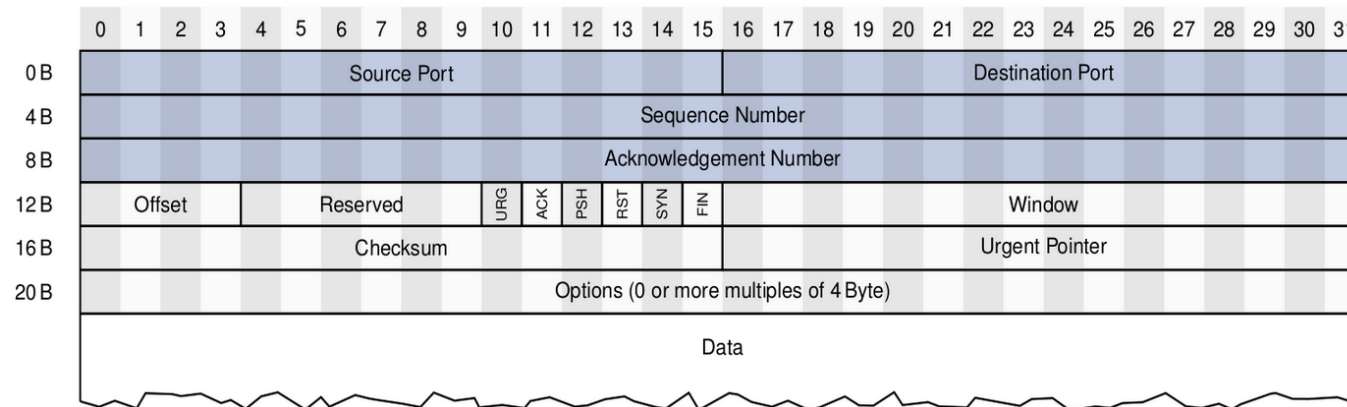
Desventajas:

- **No fiable:**
 - Posible una **pérdida de datagramas** arbitraria.
 - **Ordenación no garantizada** de los datagramas recibidos.
- **No se preocupa por el receptor**, podría, por ejemplo, no ser capaz de recibir datagramas tan rápido como el remitente puede enviar.

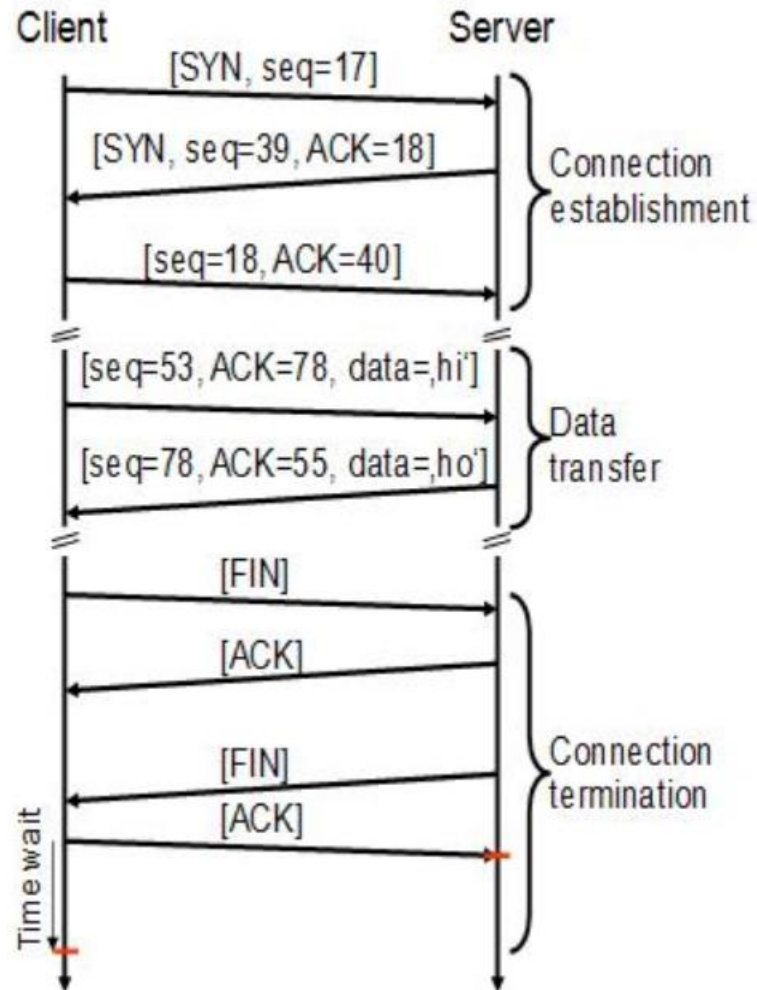
- TCP es un protocolo con **estado (stateful)**. Significa que la aplicación que envía y la que recibe comparten contexto sobre la comunicación. Para hacer esto se **establece** una **conexión** vía **handshake**. TCP utiliza un establecimiento de conexión dedicado antes de la transmisión de datos.
- TCP no maneja individualmente cada paquete IP recibido como hace **UDP** con sus **datagramas**. En su lugar, utiliza **streams** entre las aplicaciones en ambos hosts. Un stream se compone de todos los mensajes TCP desde que se establece la conexión hasta que se libera. Para habilitarlo, los paquetes denominados **segmentos**, proporcionan más información que los datagramas UDP, esto junto con el estado que se mantiene entre ambos extremos de la conexión TCP, proporciona una alternativa **fiable** a UDP.
- Mientras que en UDP cada datagrama es autónomo (lo que significa que es independiente del datagrama previo y del posterior) los segmentos TCP no lo son. Una conexión TCP puede verse como un **stream de bytes** dentro de una **secuencia de segmentos TCP**. Por eso la importancia en el orden de los segmentos para TCP. Si una aplicación recibe segmentos en un orden diferente de lo esperado, la secuencia de bytes completa se estropearía o sería inutilizable. Con UDP sólo un paquete estaría faltando o colocado incorrectamente.
- **La fiabilidad** es una característica importante que es utilizada por muchas aplicaciones que utilizamos a diario. Por ejemplo, **HTTP(S)**, el protocolo que usa tu browser para hacer peticiones a los sites web usa TCP. **SSH** es otro ejemplo de una aplicación que se basa en TCP. Esto son sólo dos de los muchos ejemplos. Todas estas aplicaciones tienen en común que importa que **todos los datos se transmitan y se reciban** en el **orden correcto**. Incluso si sólo faltan algunos bytes de un sitio web, el contenido de la página web podría volverse completamente ilegible, ya que el navegador se basa en que todos los datos se transmiten correctamente.



- **Source Port** (16 bits): puerto origen, indica que aplicación envía el segmento.
- **Destination Port** (16 bits): puerto destino, aplicación destino en el host distante.
- **Sequence Number** (32 bits): el número de secuencia se utiliza para garantizar que el receptor detecta los segmentos que faltan y puede organizar todos los segmentos recibidos en el orden correcto.
- **Acknowledgement Number** (32 bits): indica el número de secuencia que se espera que se reciba a continuación. También confirma la recepción de todos los bytes anteriores en esa conexión.
- **Offset** (4 bits): indica donde empiezan los datos en múltiplos de 4 Bytes. Se necesita porque la cabecera tiene una longitud variable. El offset mínimo es 5 ($5 \times 4 \text{ bytes} = 20 \text{ bytes}$), como la cabecera TCP es de al menos 20 bytes, el máximo offset es 15 (60 bytes).
- **Reserved** (6 bits): actualmente, no hay uso para estos 6 bits. Están reservados para uso futuro y normalmente se ponen a 0.
- **Window** (16 bits): indica cuánto Bytes está dispuesto a recibir el remitente del segmento. Se utiliza para habilitar el control de flujo, lo que significa que el remitente de datos no envía más de lo que el receptor puede procesar.
- **Checksum** (16 bits): se utiliza para detectar errores de transmisión en los segmentos TCP recibidos.
- **Urgent Pointer** (16 bits): no relevante para este curso.
- **Options** (variable size): opciones adicionales.



- **Flags** (6 bits): los flags son muy importantes para realizar un seguimiento de la conexión TCP. El significado de ellos se muestra a continuación:
 - **URG**: un valor a 1 en esta posición indica que la información en el Urgent Pointer es relevante.
 - **ACK**: este flag se utiliza en todos los segmentos, excepto en el primero en una conexión. Se utiliza para confirmar la recepción de los datos. ACKs pueden ser piggybacked, lo que significa que un segmento transporta datos y reconoce los datos recibidos previamente (envío de datos y el acuse de recibo en un mismo paquete).
 - **PSH**: este flag indica que hay que enviar los datos a la aplicación. Si no se activa, TCP almacena en un búfer los segmentos y espera más segmentos durante un tiempo antes de entregar los datos a la capa de aplicación.
 - **RST**: si ocurre algo inesperado, la conexión se resetea utilizando este flag.
 - **SYN**: el indicador SYN indica que el segmento pertenece al establecimiento de conexión inicial antes de que se transmitan los datos reales.
 - **FIN**: se usa para terminar la conexión. El transmisor envía un segmento con este flag activo cuando termina de enviar sus datos.

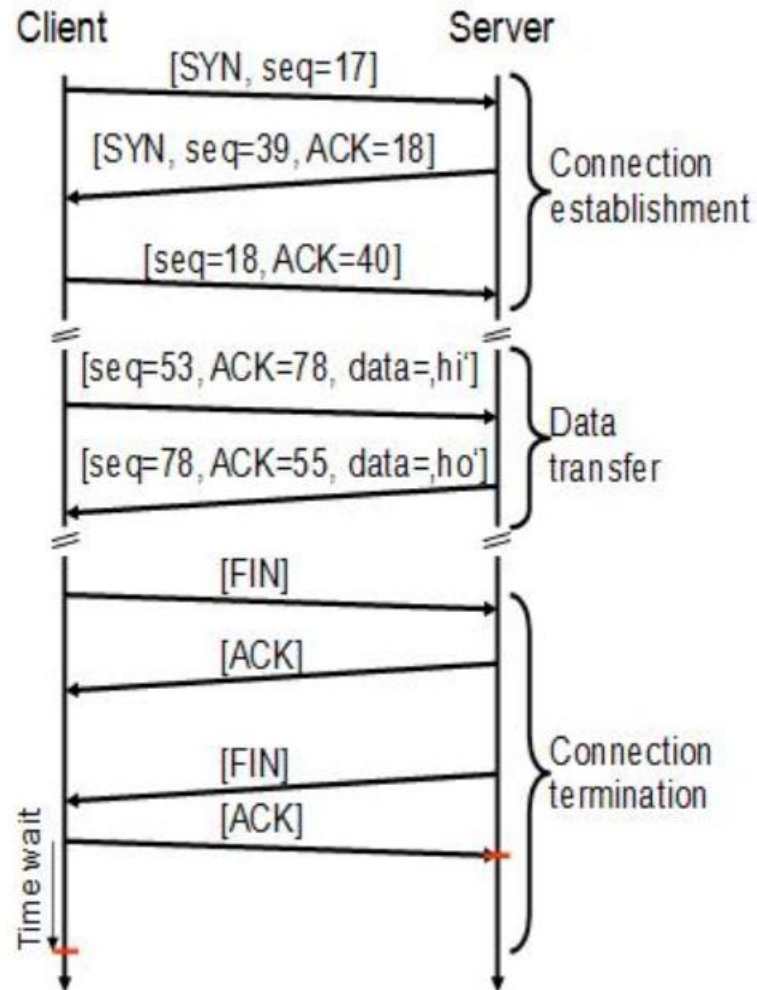


Como se mencionó anteriormente, los segmentos TCP pertenecen a una **conexión TCP**. Esta conexión garantiza que los segmentos se reciban en el **orden correcto** y **que no se pierda ninguno**.

Una conexión TCP se divide en tres fases:

1. El **Establecimiento de la Conexión**.
2. La **Transferencia de Datos**.
3. El **Cierre de la Conexión**.

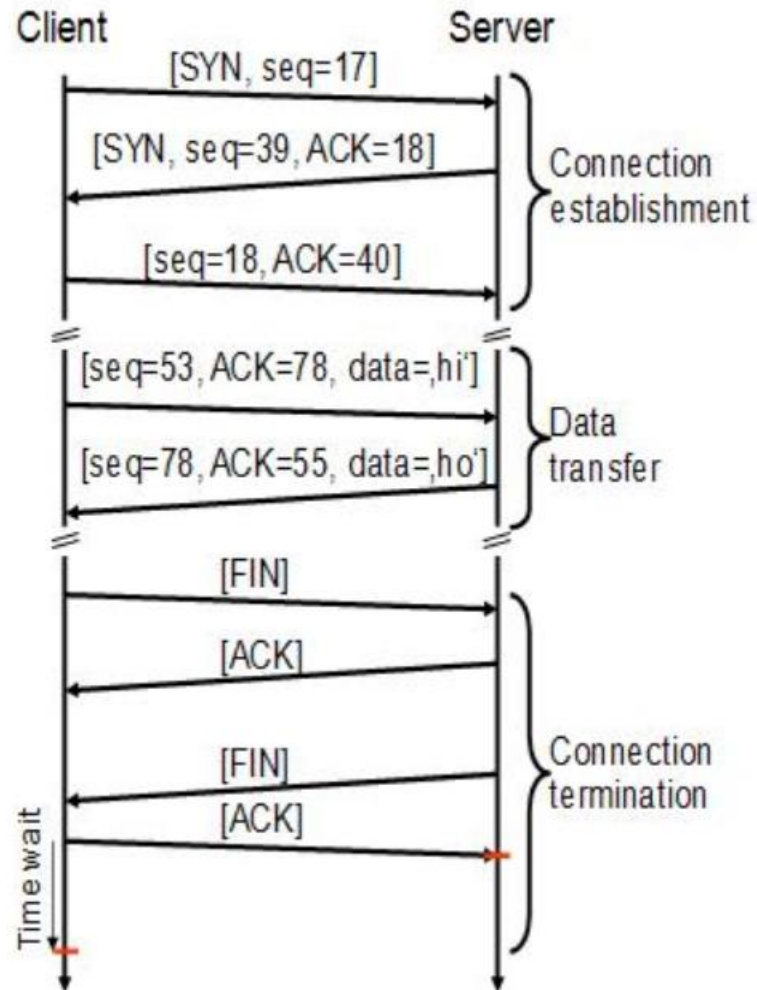
Mientras entramos en los detalles de estas fases, aprenderemos algunos detalles más sobre los **números de secuencia** y los **números de confirmación** que aparecen en la cabecera TCP.



El primer paso antes de que el transmisor pueda enviar los datos al receptor es **establecer una nueva conexión TCP** entre ambas partes. Esta fase consta de **tres mensajes** que aseguran que ambas partes comparten algunos parámetros para la conexión. En este caso, el establecimiento de la conexión recibe el nombre de **three-way handshake**.

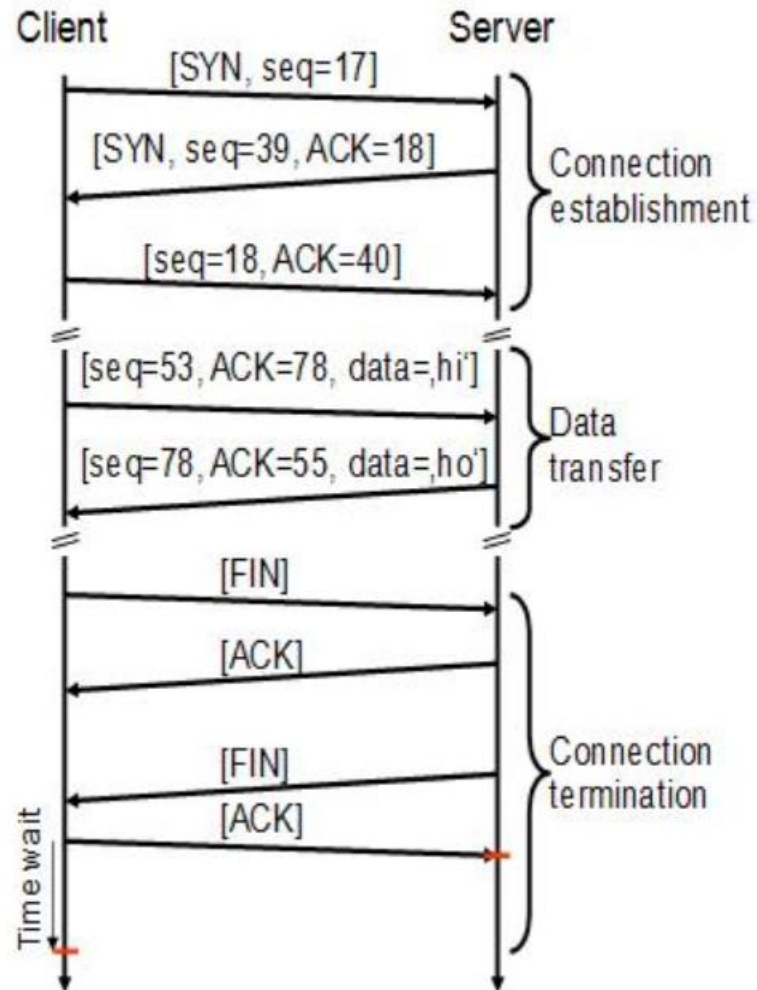
En resumen, el propósito del three-way handshake TCP es el siguiente:

- **Message 1:** el cliente elige el número de secuencia inicial (ISN) y se **SYNcroniza** con el servidor.
- **Message 2:** el servidor **ACKnowledges** al cliente la recepción de su ISN.
- **Message 2:** el servidor elige su propio ISN y se **SYNcroniza** con el cliente.
- **Message 3:** el cliente **ACKnowledges** al servidor la recepción de su ISN.



Para resumir el comportamiento de los números de secuencia (seq) y acuse de recibo (ACK) es el siguiente:

- El **número de secuencia** de un segmento indica donde está localizado el payload actual en el stream total de la conexión TCP. Es decir, **número de secuencia inicial + bytes enviados en los segmentos previos**.
- El número de acuse de recibo (ACK number) de un segmento indica al receptor acerca del **número de secuencia que el emisor está esperando recibir**.



Una vez que se han transmitido los datos, la conexión tiene que **cerrarse**. Esta **finalización** de la conexión también está definida en el protocolo. Después que el transmisor envía su último segmento, éste envía un segmento con el **flag FIN activo** que se confirma por parte del receptor. Cuando el receptor ha recibido todos los datos, también envía un segmento con el **flag FIN activo** que también es confirmado. Esta última confirmación es la que termina la conexión TCP.



Establecimiento: Three-Way Handshake


Transferencia de Datos Cliente/Servidor

Cierre: Four-Way Handshake (Teardown)


- Puesto que TCP **garantiza** que cada segmento que se envía se **entrega a su destino**, veamos el mecanismo que se inicia cuando se **pierde** un segmento. Los segmentos se pueden perder por varias razones. Podría haber un error al transmitirlo, pero también puede suceder que la red esté **congestionada** y los segmentos por lo tanto se descarten en su camino al destino. También puede suceder que los datos se reciban, pero el **acuse de recibo se pierda** en el camino de regreso. En cualquiera de estos escenarios, el remitente del segmento original no obtiene una confirmación y debe asumir que el segmento se perdió.
- Para determinar si un segmento se ha perdido, TCP tiene un mecanismo que hace una retransmisión después de un cierto **timeout**. Por supuesto, la entrega de un segmento lleva algún tiempo, así como la entrega del accuse de recibo correspondiente. Por lo tanto, TCP necesita determinar cuánto tiempo espera hasta que se supone que se ha perdido un segmento. Las implementaciones modernas de TCP utilizan el **Round Trip Time (RTT)** para determinar este tiempo de espera. El RTT es el tiempo que tarda un segmento en enviarse al destino más el tiempo que tarda el accuse de recibo en volver al remitente.
- TCP mide el RTT para los segmentos enviados con éxito donde recibió un accuse de recibo y calcula una **RTT promedio**. Dependiendo de la implementación de TCP que se utilice, este RTT se usa para calcular un tiempo de espera, que debe ser mayor que el RTT pero no demasiado grande, ya que de lo contrario podríamos esperar innecesariamente mucho antes de retransmitirse un segmento perdido.
- Cuando se envía un segmento, se inicia el tiempo de espera. **Si no hay reconocimiento dentro de ese tiempo, el segmento se volverá a enviar.**
- Otro indicador para la pérdida de segmento es cuando **el transmisor recibe múltiples confirmaciones para el mismo número de secuencia**. Esto puede suceder cuando varios segmentos llegan al receptor y faltan uno o más segmentos en el medio. En este caso, el receptor confirma el número de secuencia del mensaje perdido, indicando que todavía está esperando por el segmento acordado. Entonces, el emisor tiene que **reenviar** el segmento. También hay otros algoritmos que hacen re-transmisiones algo diferentes, pero estos están fuera del alcance del curso.

- Como puedes imaginar, todos los nodos en una comunicación quieren tener la mejor calidad de comunicación posible. Esto incluye especialmente un alto rendimiento y una baja necesidad de retransmisión. Puesto que las máquinas en Internet tienen recursos diferentes, puede ser que haya escasez de recursos en la red, en el remitente o en el receptor de una conexión TCP.
- Si hay un problema de recursos dentro de la red, esto se denomina **congestión**. Imagine muchos paquetes IP que van sobre el mismo link. Esto es similar a muchos coches que pasan por la misma carretera, se produce una congestión.
- Si el receptor no puede procesar con la frecuencia necesaria los paquetes recibidos, esto es un problema denominado **control de flujo (flow control)**. Deben enviarse sólo los segmentos TCP que el receptor puede procesar.
- Como se mencionó anteriormente, TCP tiene mecanismos que aseguran que cada segmento puede entregarse. TCP también se ocupa de la congestión y el control de flujo. Para el **Control de Cogestión** TCP se asegura de enviar sólo el número de segmentos que el ancho de banda entre el transmisor y el receptor pueden manejar. En cambio, para el **Control de Flujo** se asegura que el remitente sólo envíe tantos bytes como el receptor pueda manejar.



 Calle Playa de Liencres, 2 bis
(entrada por calle Rozabella)
Parque Europa Empresarial
Edificio Madrid
28290 Las Rozas, Madrid

 900 373 379  info@u-tad.com

 [SOLICITA MÁS INFORMACIÓN](#)



CENTRO ADSCRITO A:

 **Universidad
Camilo José Cela**

PROYECTO COFINANCIADO POR:

