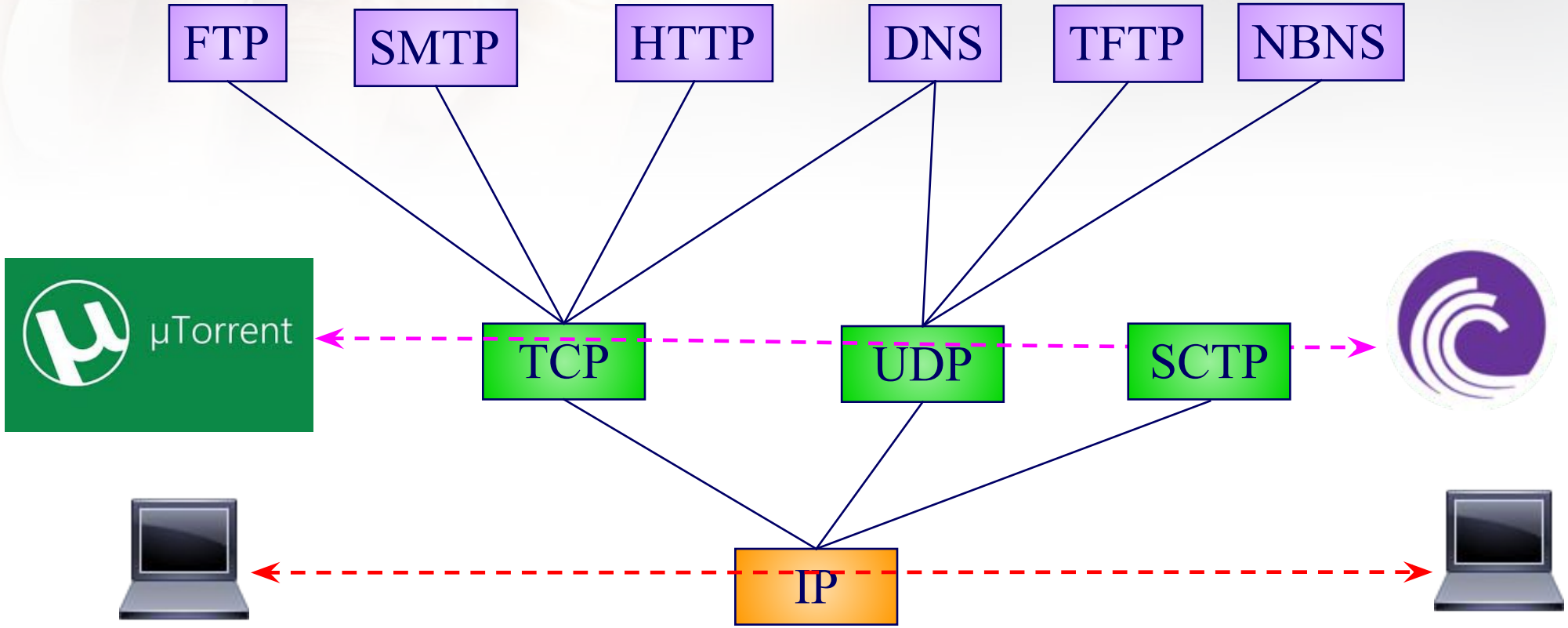
Protocolos de transporte

- Servicios
- Multiplexación
- UDP
- TCP

Protocolos de transporte



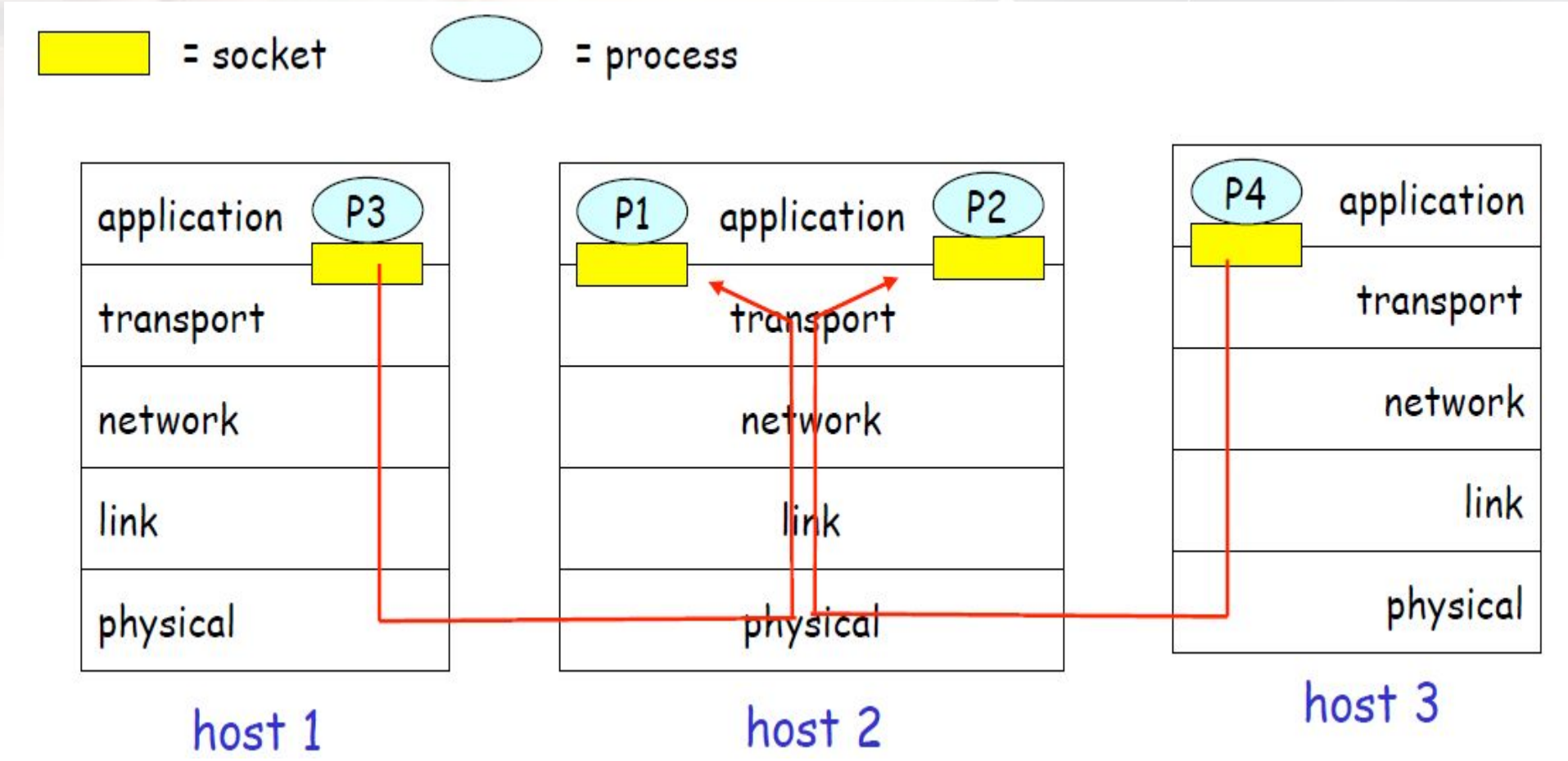
Servicios

- Provee la comunicación lógica entre dos procesos que corren en distintos hosts
- Ejecutan en las «puntas finales»
  - El que envía: separa los mensajes en segmentos y los entrega al nivel de red
  - El que recibe: reensambla los segmentos en mensajes y los pasa al nivel de aplicación
- Protocolos de transporte: UDP, TCP y SCTP

Servicios

- TCP ofrece:
  - Control de congestión
  - Control de flujo
  - Establecimiento de conexión
- UDP ofrece:
  - «mejor esfuerzo» (como IP)
- NO ofrecen:
  - Mínimo de demora o latencia
  - Mínimo de ancho de banda

Multiplexación / Demultiplexación



DeMultiplexación

- Un host recibe datagramas IP
  - Cada datagrama tiene IP origen e IP destino
  - Cada datagrama contiene un segmento del protocolo de transporte
  - Cada segmento contiene un puerto de origen y un puerto de destino
- El host utiliza IP + puerto para dirigir el segmento al socket apropiado

Multiplexación: puertos

Números de puerto (RFC 1700)

Puerto	Uso
< 255	aplicaciones públicas
[255, 1023]	asignados a empresas para aplicaciones comerciales.
>1023	no regulados, aunque muchos están reservados

Gran parte de los puertos están definidos tanto para TCP como para UDP, aunque algunas aplicaciones utilicen sólo uno de ellos.

Multiplexación: puertos

Ejemplo de algunos puertos.

Puerto	Nombre
7	ECHO
9	DISCARD
13	DAYTIME
20	FTP-DATA
21	FTP
23	TELNET
25	SMTP

Puerto	Nombre
67	BOOTPS
68	BOOTPC
69	TFTP
80	HTTP
109	POP2
110	POP3
119	USENET

Multiplexación: puertos

Ejemplo de algunos puertos.

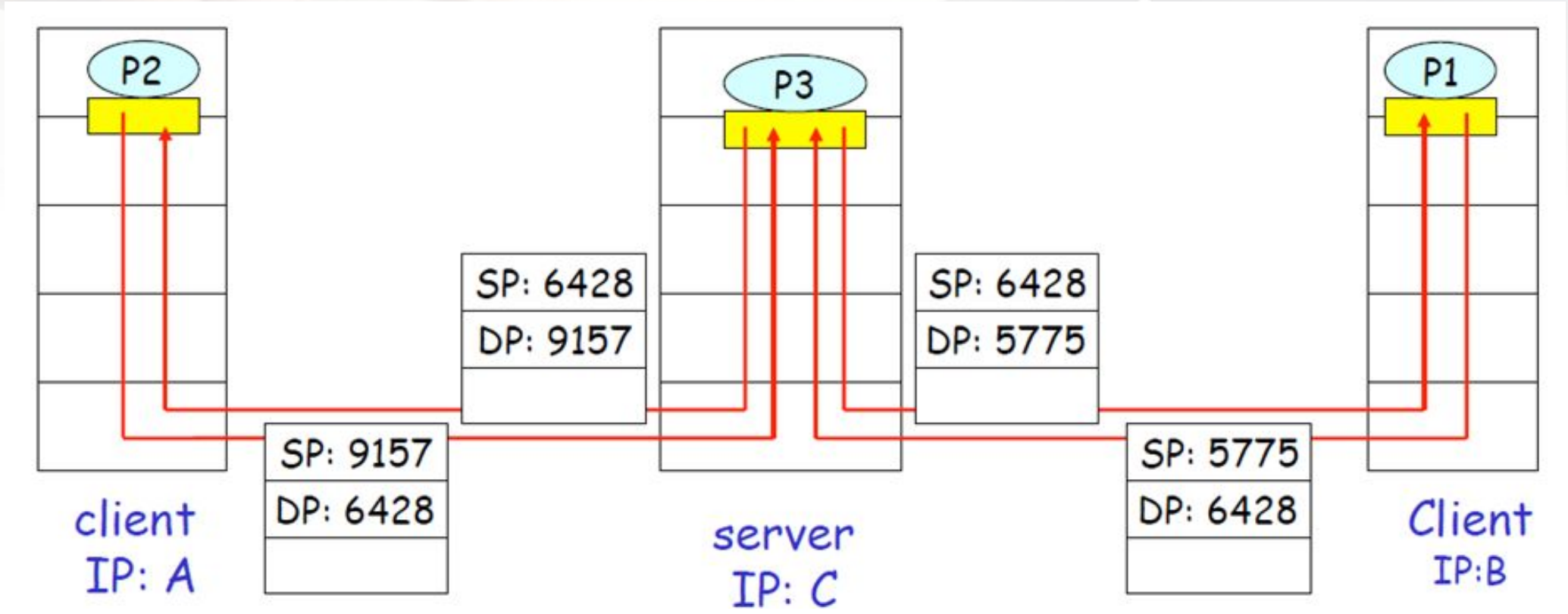
Puerto	Nombre
161	SNMP
162	SNMPTrap
377	NEC corporation
385	IBM application
530	RPC
531	chat
749	kerberos admin.

Puerto	Nombre
1352	Lotus Note
1433	MS SQL Server
1524	Ingres
1525	Oracle
2041	Interbase
5190	AOL
…	…

Demultiplexación sin conexión

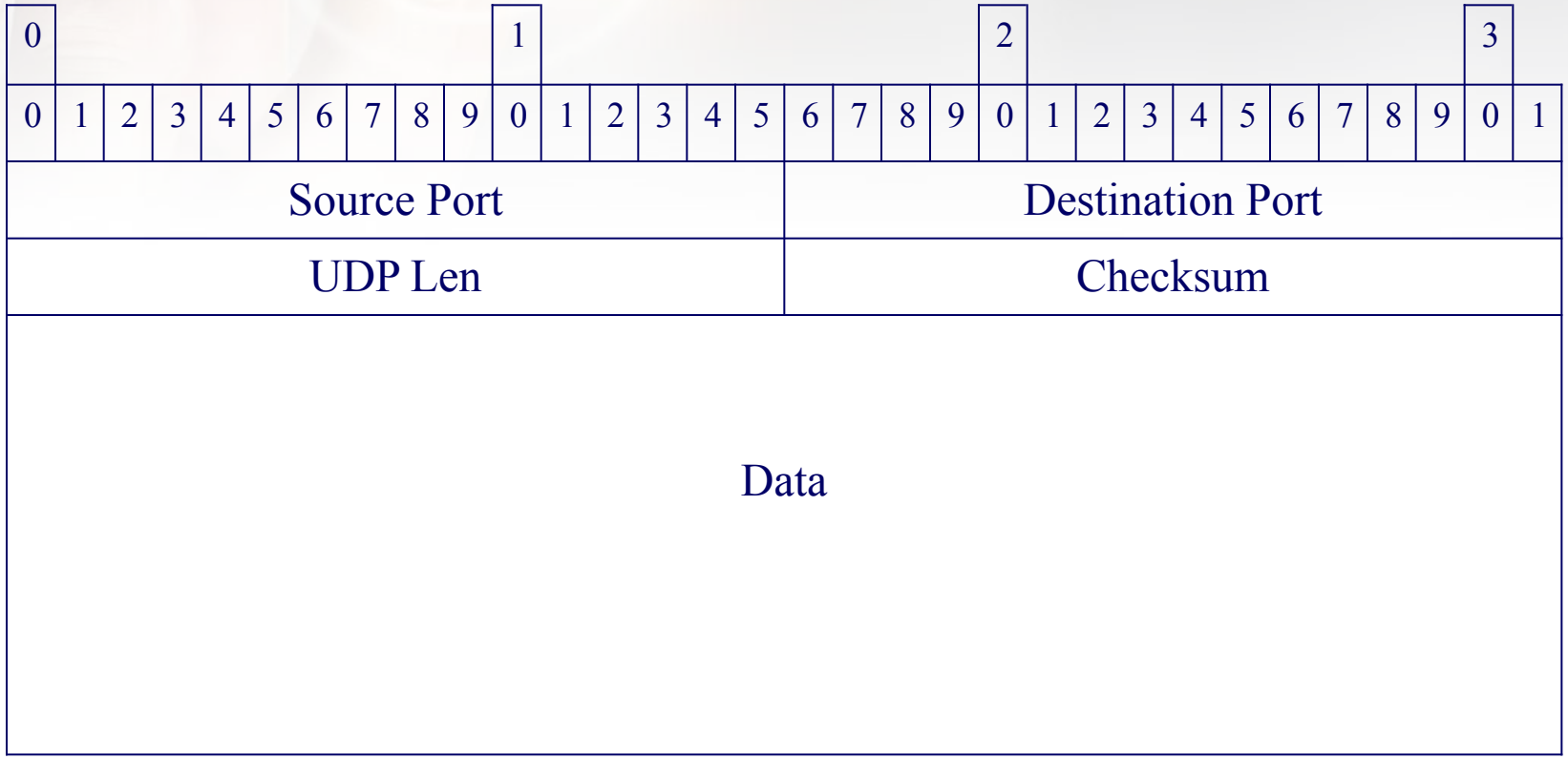
- Crear socket con número de puerto
- El socket se identifica por el par <IP destino, Puerto destino>
- Cuando el host recibe segmento UDP
  - Dirige el segmento al socket que atiende <IP, puerto>
- Datagramas con distinto IP origen son atendidos por el mismo socket

Demultiplexación sin conexión



UDP

Formato de un mensaje UDP



UDP

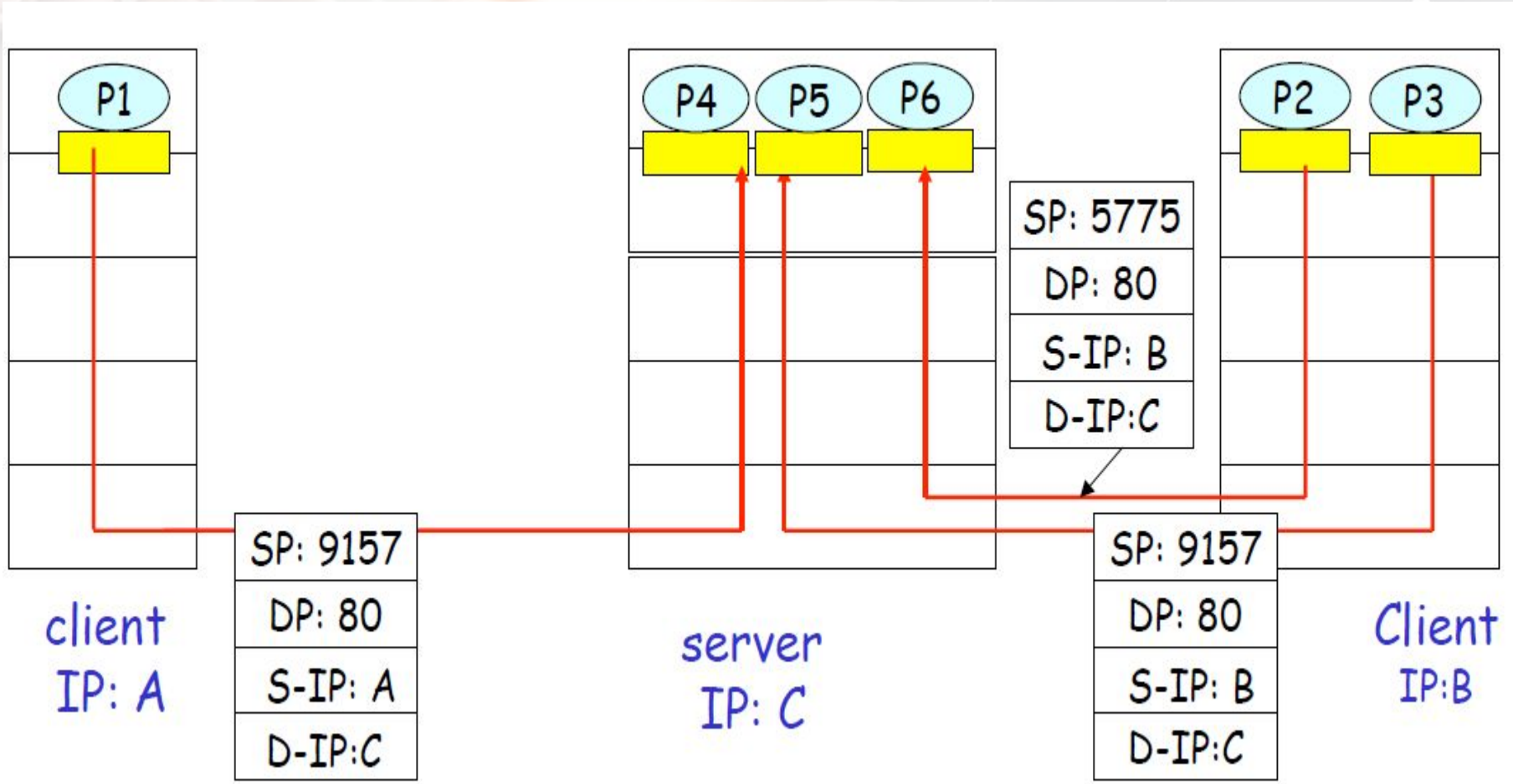
UDP transporta datos de manera no confiable entre hosts.

- No orientado a conexión
- No confiable
- No ofrece verificación de software para la entrega de segmentos
- No reensambla los mensajes entrantes
- No utiliza acuses de recibo (*ACK*)
- No proporciona control de flujo

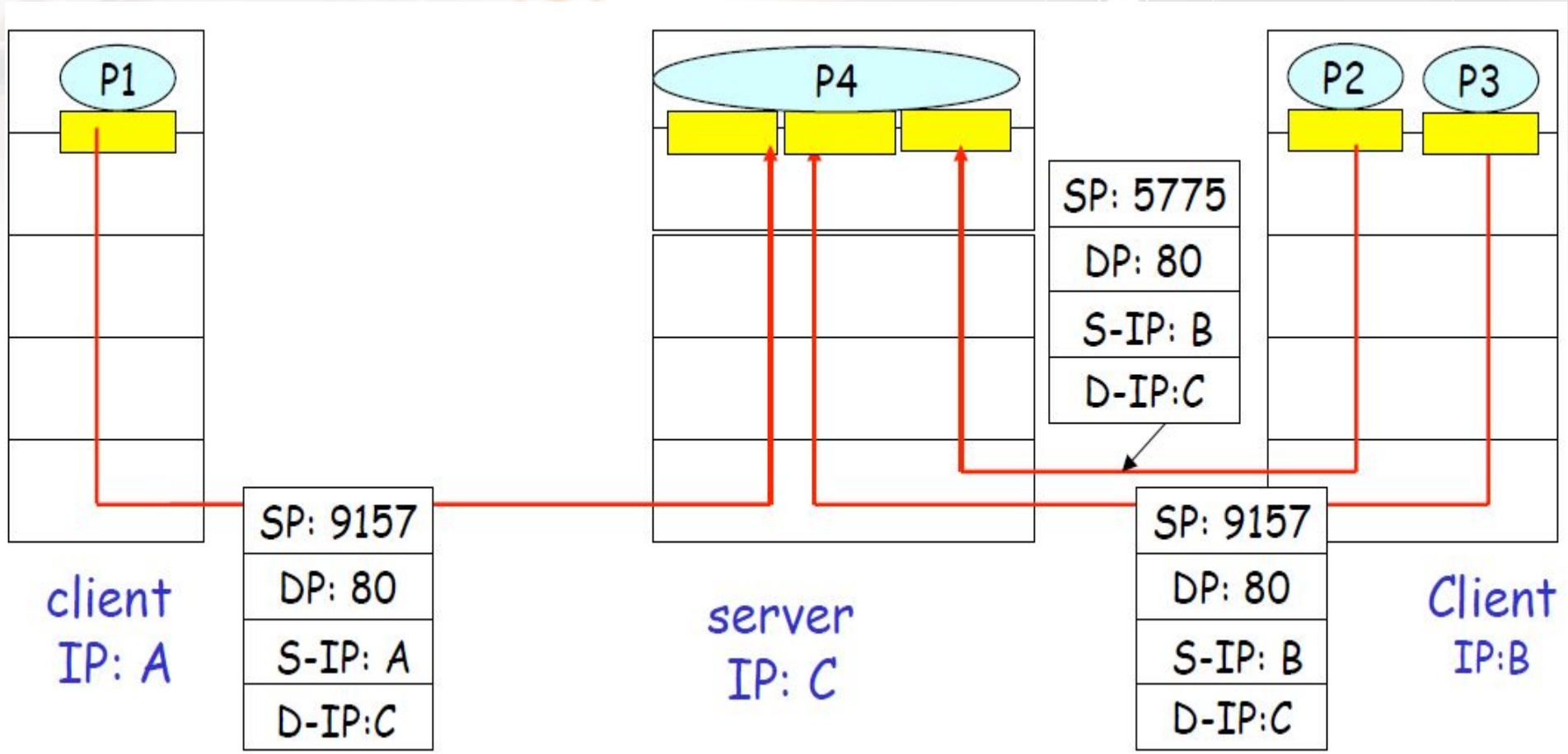
Demultiplexión orientado a conexión

- Cada socket identificado por
  - IP origen
  - Puerto origen
  - IP destino
  - Puerto destino
- Host que recibe usa los 4 valores para dirigir el segmento al proceso que corresponde
- Conexiones simultáneas: c/u con su socket

Demultiplexión TCP



Demultiplexión TCP: multithreading



Transferencia de datos confiable

- Si el protocolo de red es confiable
  - No se alteran bits
  - No se pierden paquetes
  - Los paquetes llegan en orden

$\Rightarrow$ La lógica a implementar es muy simple

```
1.  Esperar llamada de aplicación
2.  Leer datos a enviar
3.  Armar paquete
4.  Pedir a capa de red que lo envíe
```

Sender

```
1.  Esperar llamada de nivel de red
2.  Leer datos recibidos
3.  Extraer datos del paquete
4.  Enviar datos a aplicación
```

Receiver

Canal con posibles errores

- Protocolo de red puede alterar algunos bits
  - Detectar errores: agregar CRC
  - ¿Cómo recuperarse de los errores?
    - ACK
    - NAK
    - Sender debe retransmitir ante NAK o falta de ACK
- Los algoritmos para Sender y Receiver ya no son tan simples

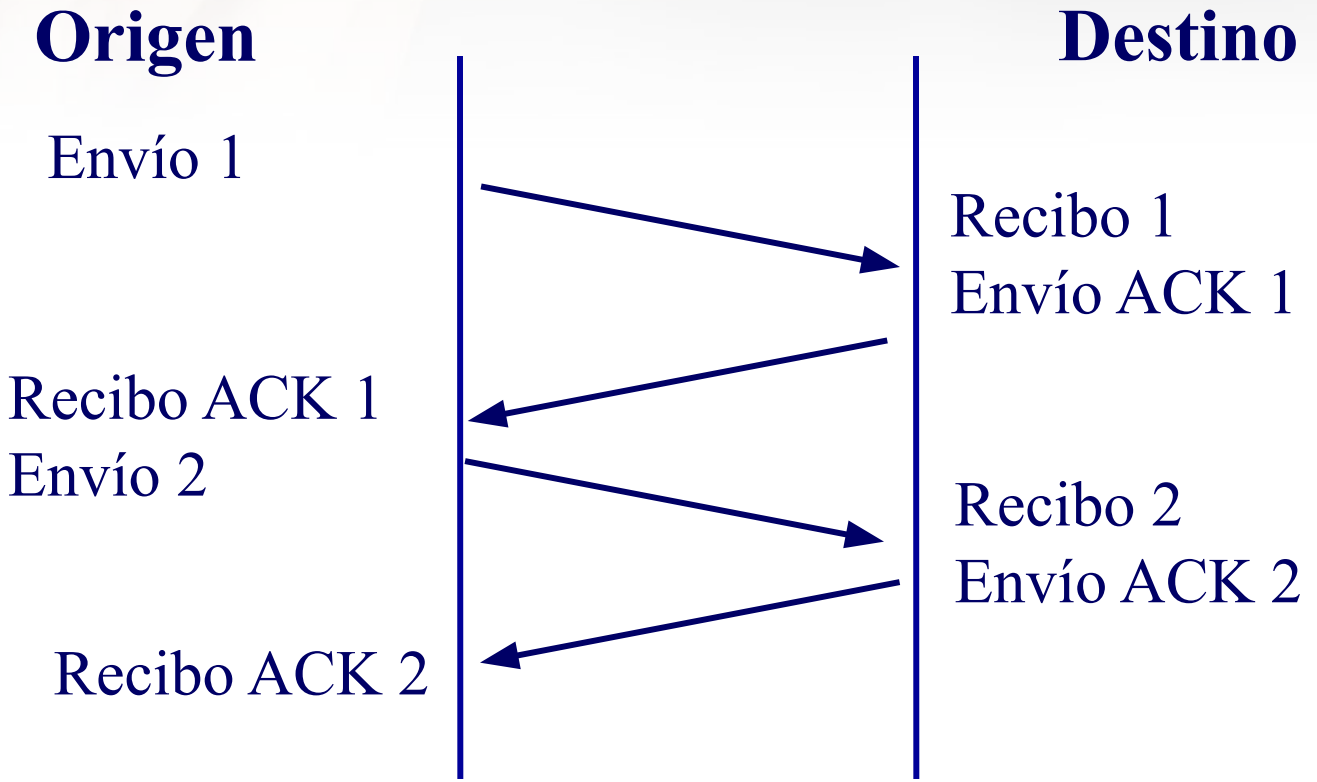
¿Qué sucede si se corrompe ACK/NAK?

¿Y si se envían paquetes duplicados?

Números de secuencia en cada paquete

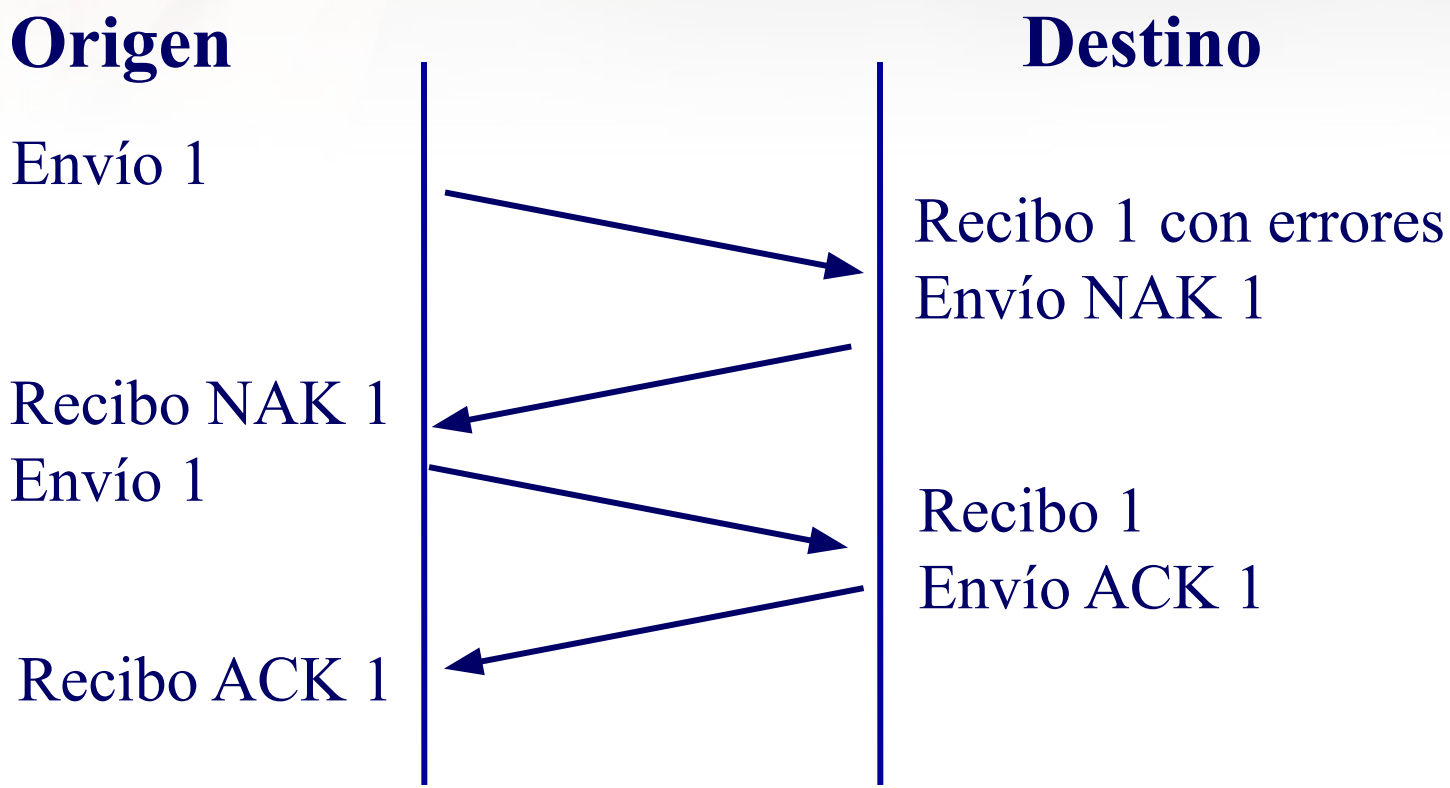
Control trivial: stop & wait

Acuse de recibo



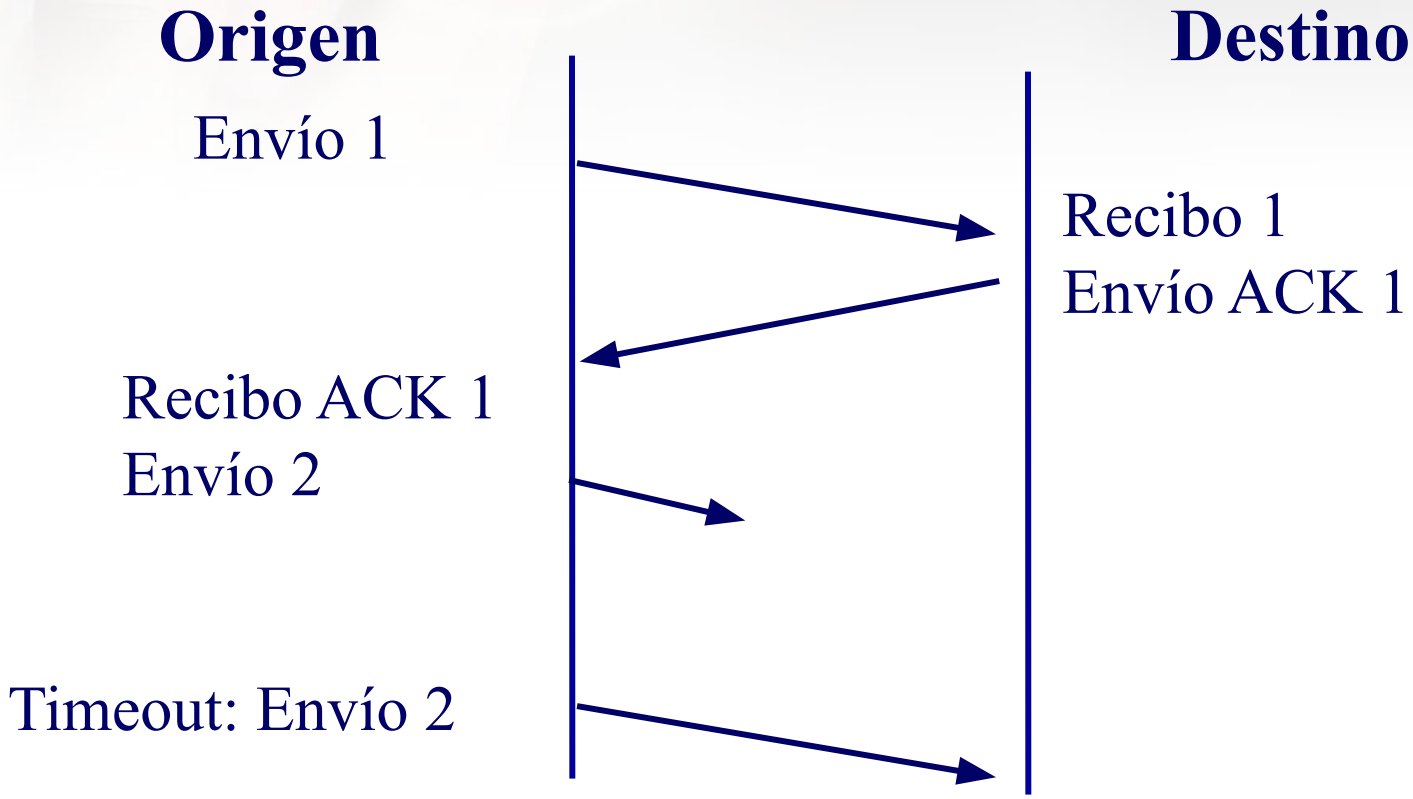
Control trivial: stop & wait

Acuse de recibo



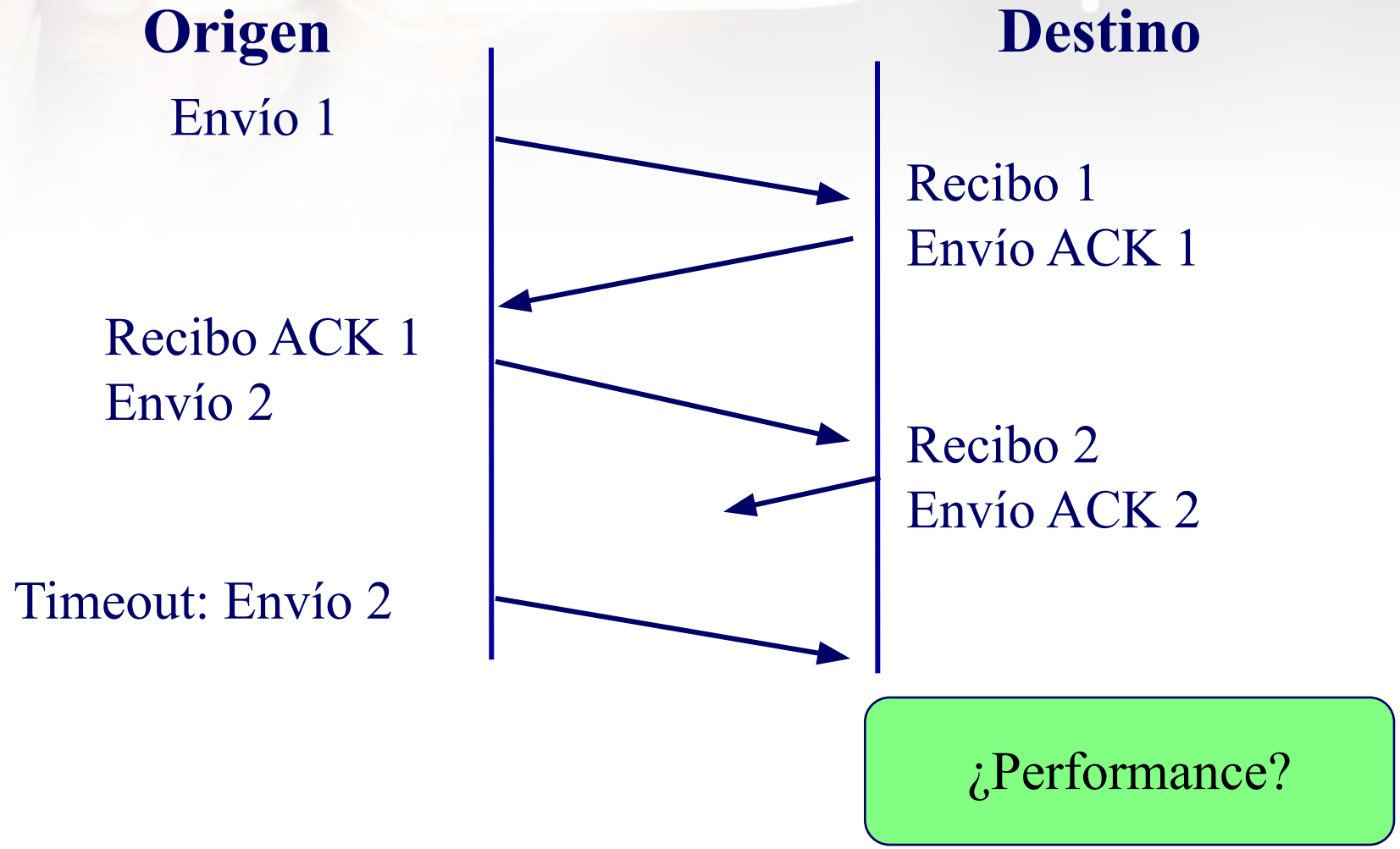
Control trivial: stop & wait

Acuse de recibo



Control trivial: stop & wait

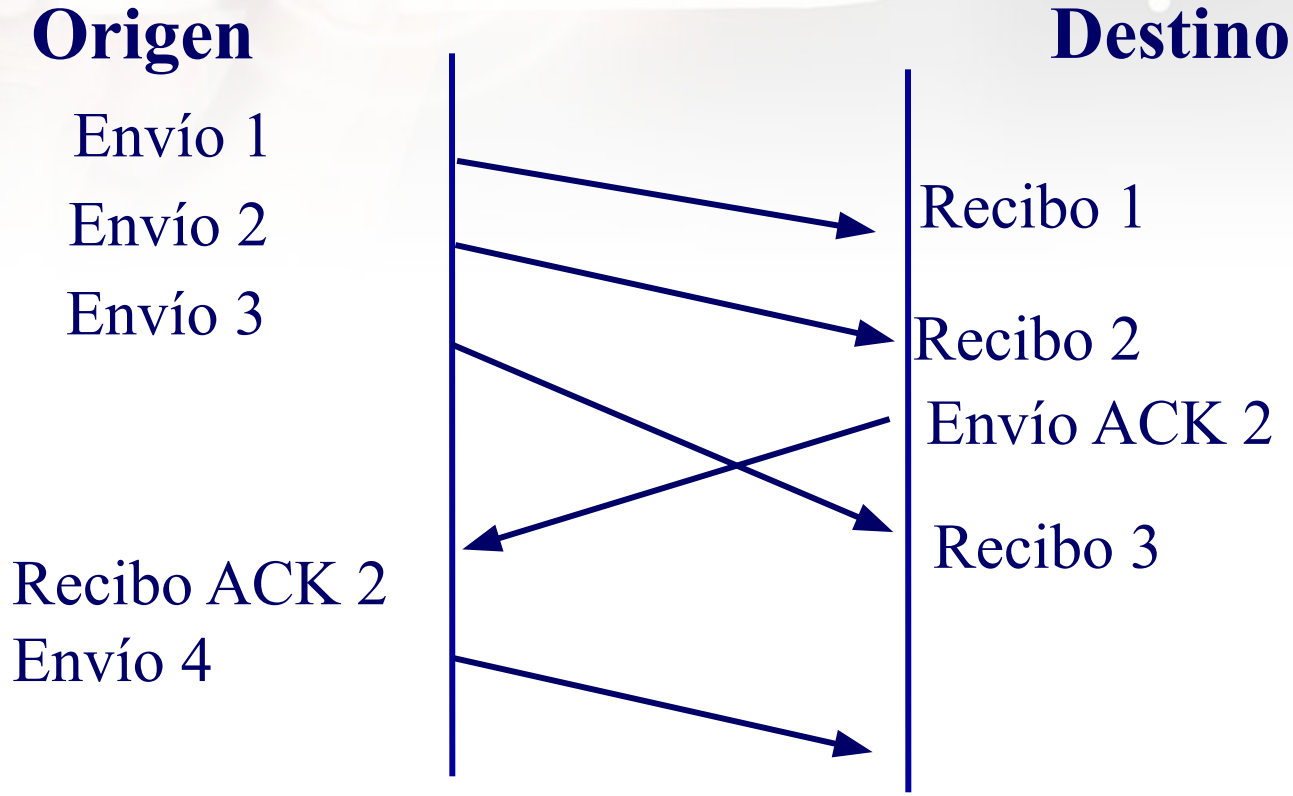
Acuse de recibo



¿Performance?

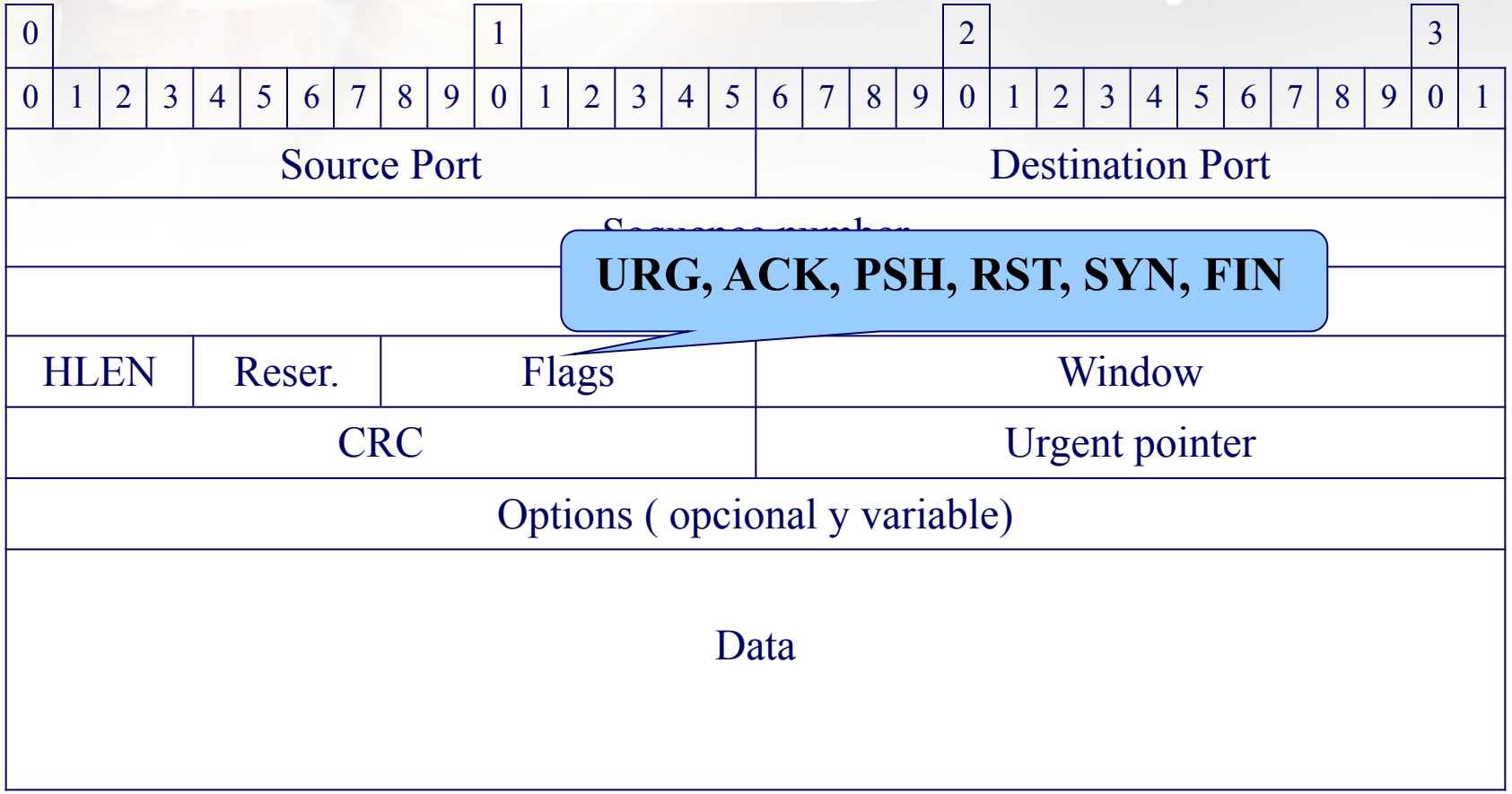
Pipelining

Acuse de recibo



TCP

Formato de un mensaje TCP

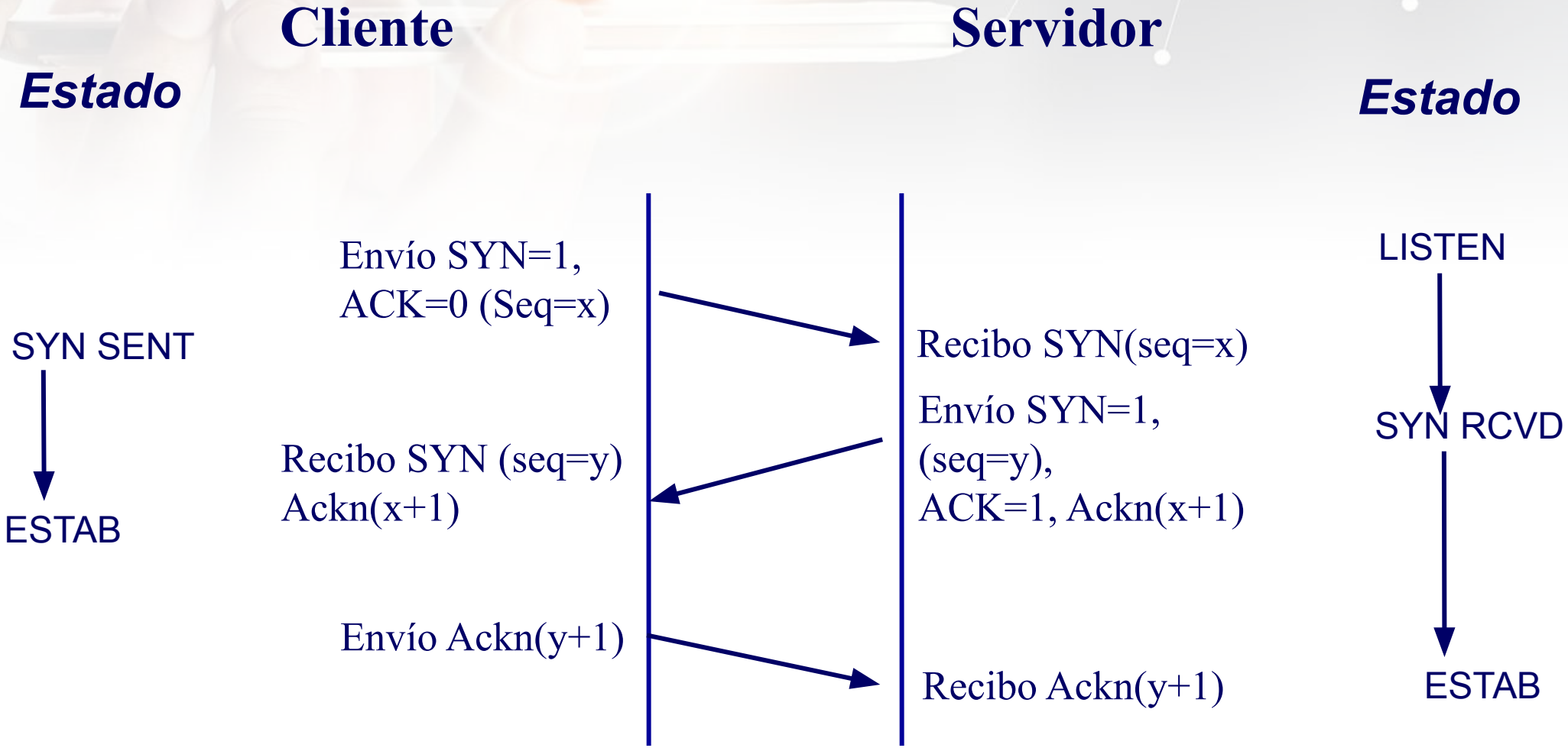


TCP

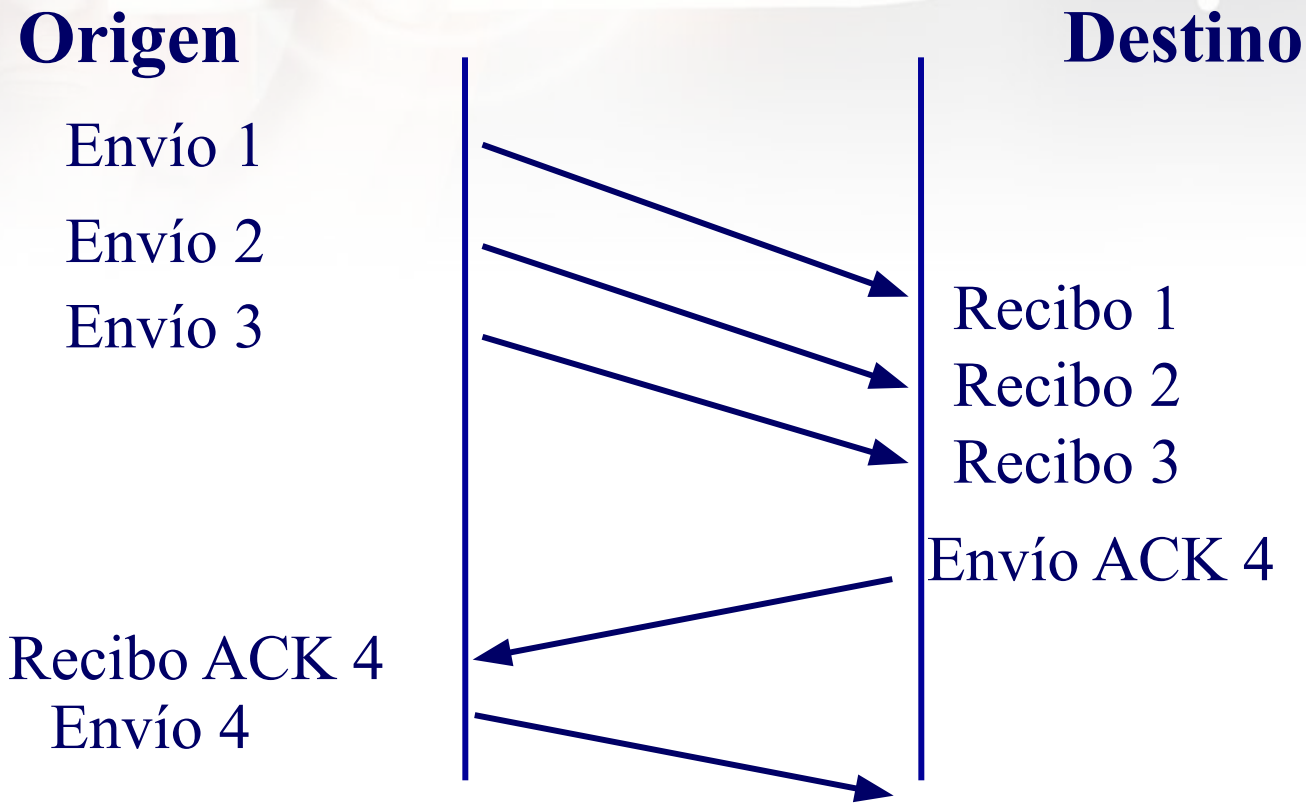
TCP ofrece un circuito virtual entre aplicaciones de usuario final.

- Orientado a conexión
- Confiable
- Divide los mensajes salientes en segmentos
- Reensambla los mensajes en el host destino
- Vuelve a enviar lo que no se ha recibido
- Control de flujo (rfc 7323)
- Control de congestión (rfc 5681)

TCP: establecer conexión

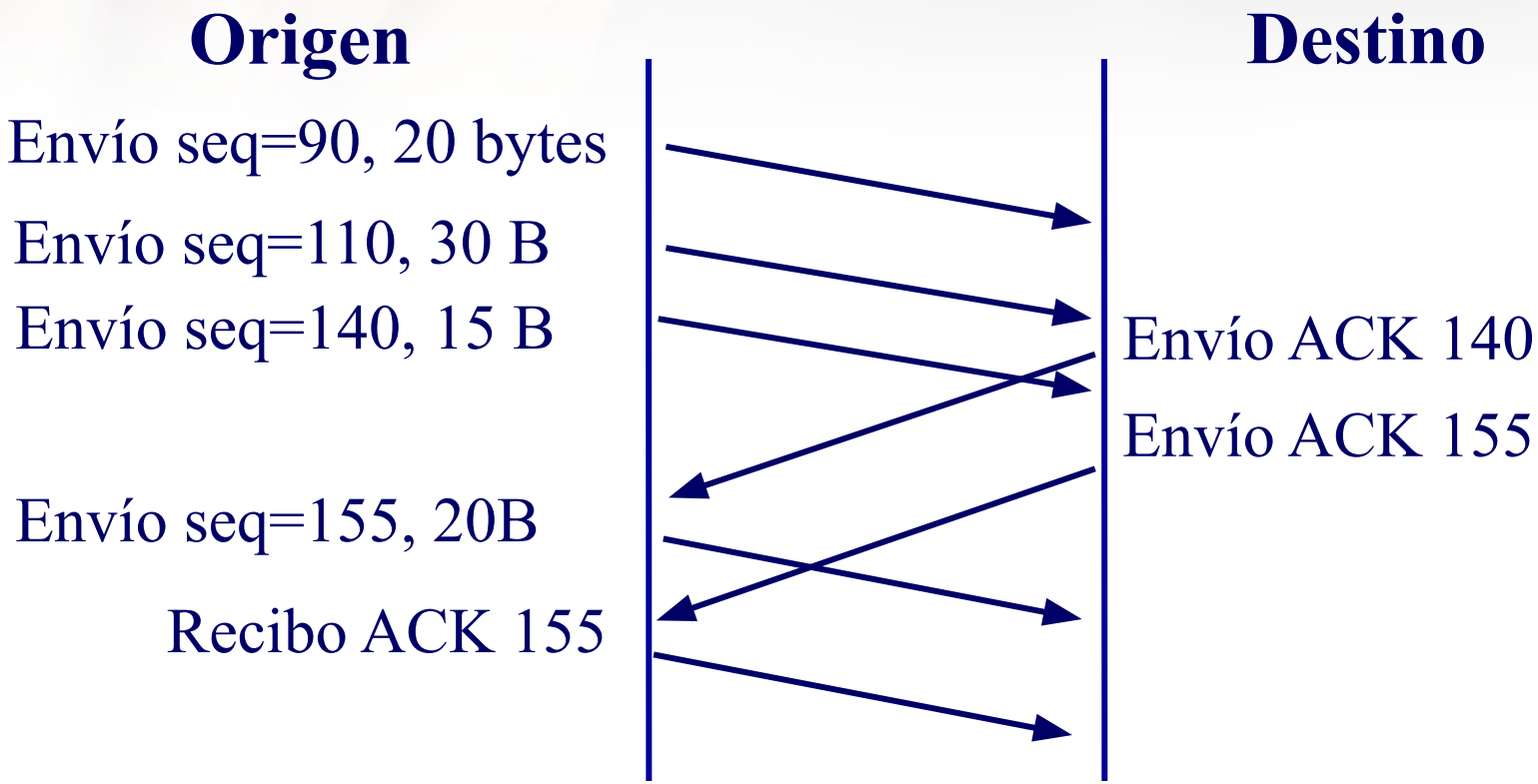


TCP: acuse de recibo

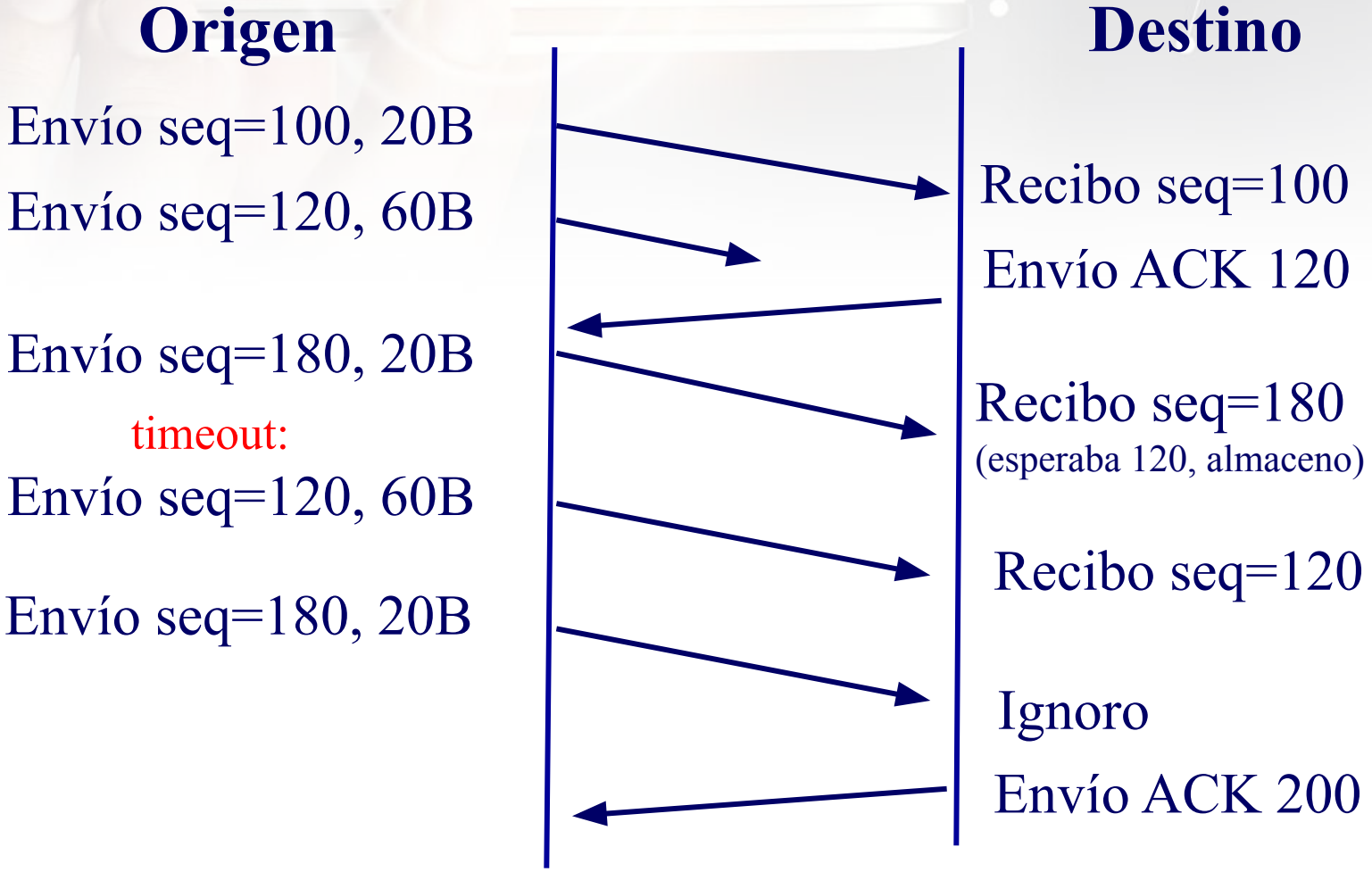


TCP: acuse de recibo

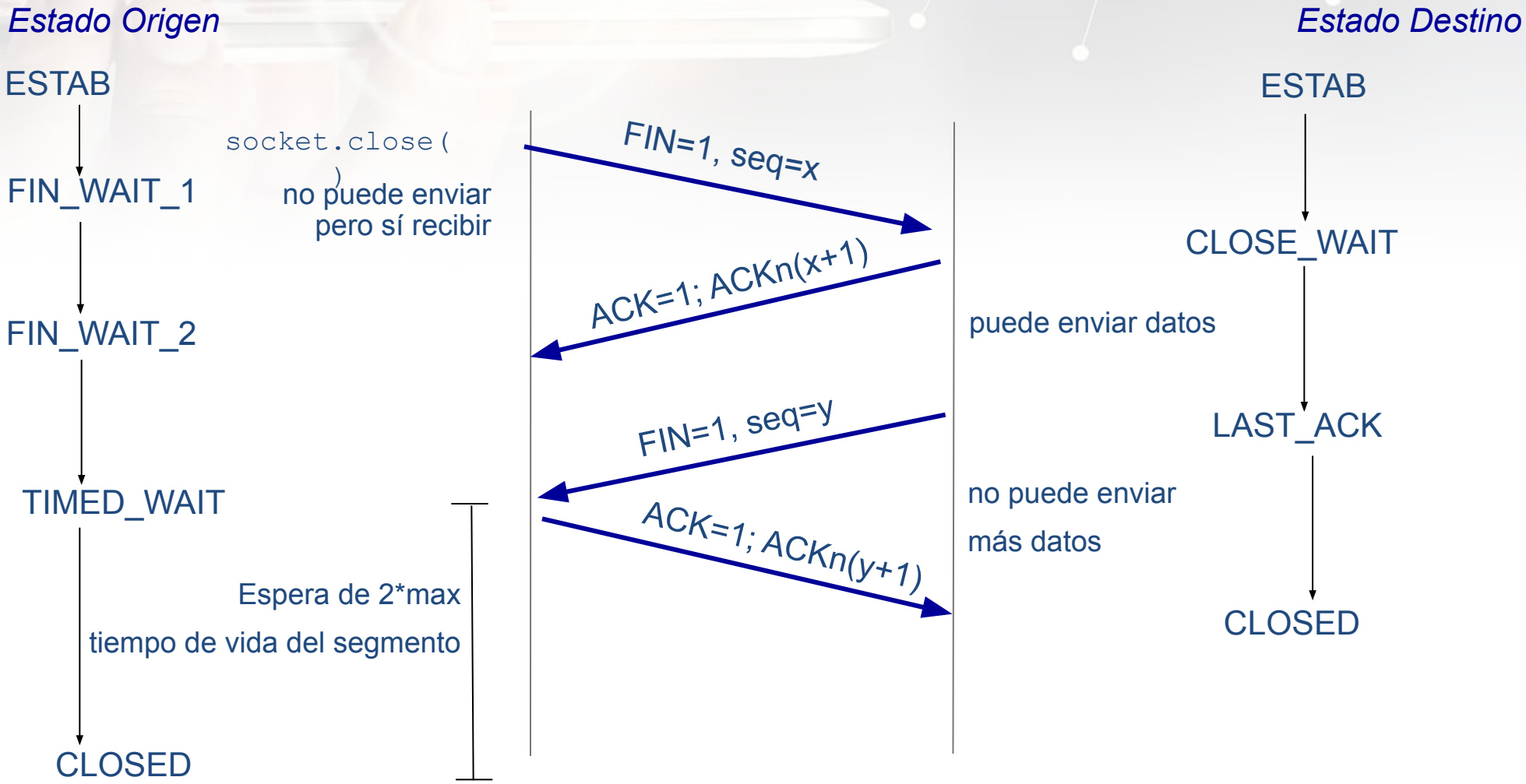
Como el canal es un «flujo de datos» no se envía ACK por segmento sino por el próximo byte a recibir



TCP: pérdida de segmentos

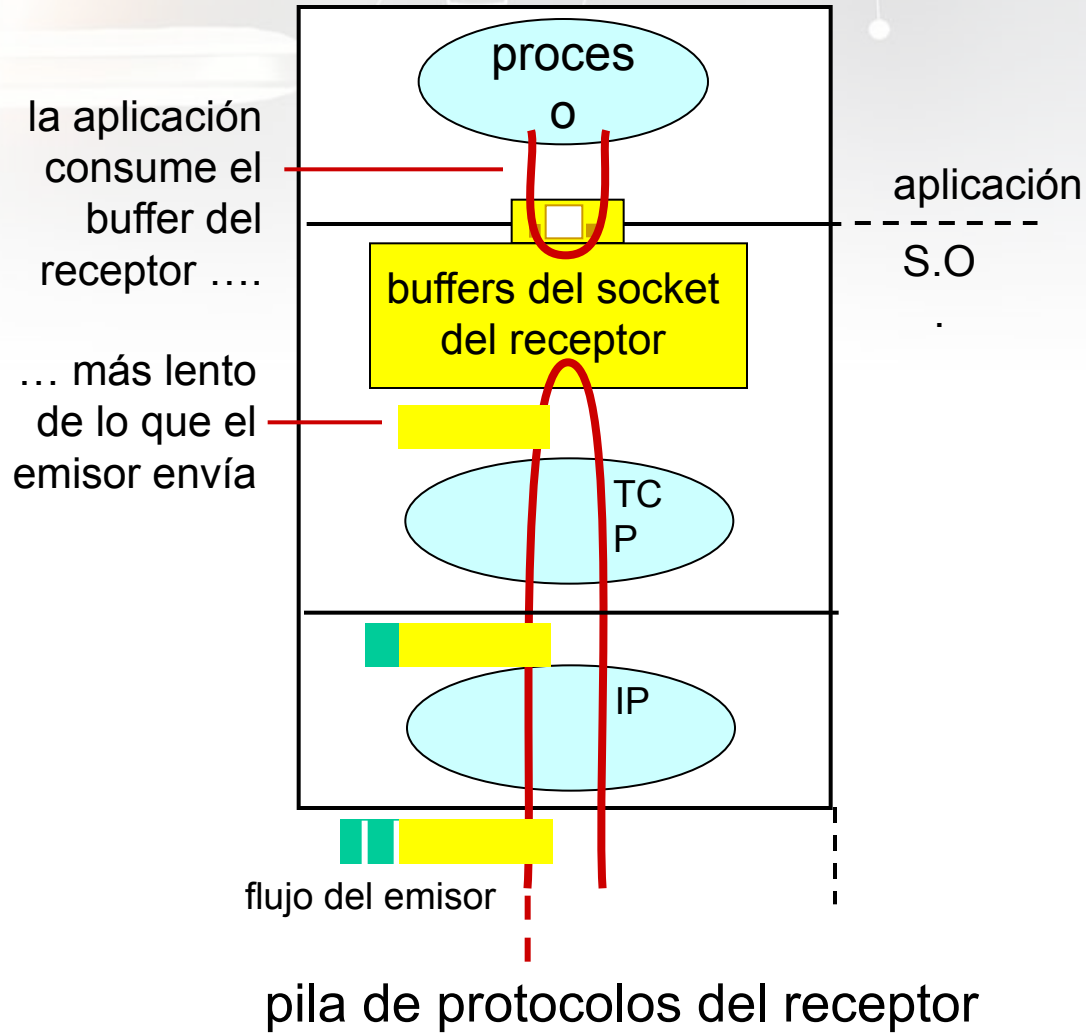


TCP: cierre de conexión



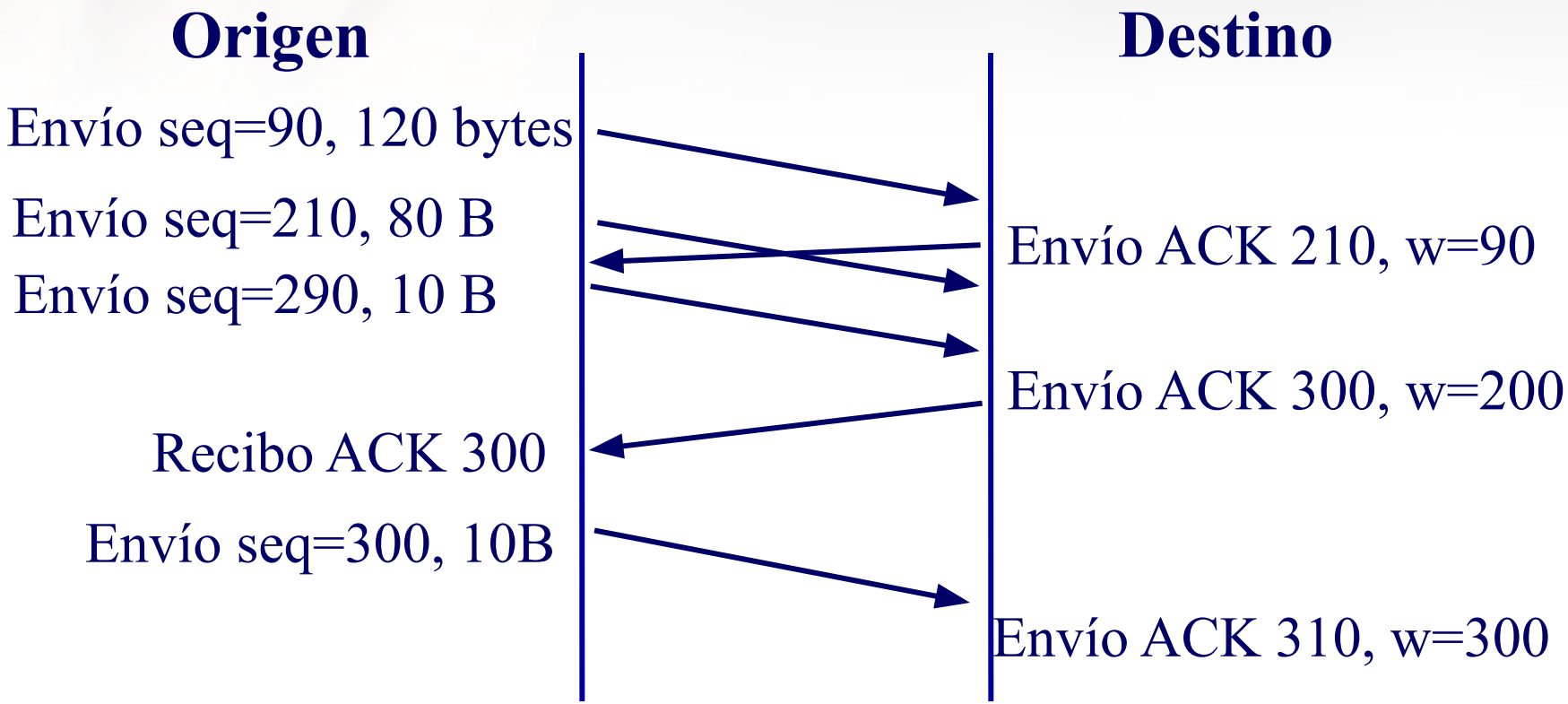
TCP: control de flujo

El receptor controla al emisor, de modo que el emisor no sobrecargue el buffer del receptor



TCP: control de flujo

Junto con el ACK enviamos tamaño de “ventana”



TCP: control de flujo

El campo Window es de 16 bits, lo cual permite informar un buffer de hasta 64 KB.

El valor suena apropiado para la época en que el ancho de banda máximo era de 56Kbps.

¿Es adecuado ese tamaño máximo hoy en día?

¿En qué afecta a la performance un buffer de recepción acotado?

Control de congestión

Hace referencia al tráfico IP

- Muchas fuentes enviando muchos datos y a una velocidad que la capa **de red** no puede manejar.
- No es lo mismo que control de flujo
- Cómo se manifiesta
  - pérdida de paquetes (descartados por el router)
  - demoras (mucho tiempo almacenado en buffers de routers)
- IP no se hace cargo

Lo ideal sería que TCP emita segmentos que no sean descartados, y así evitar la retransmisión

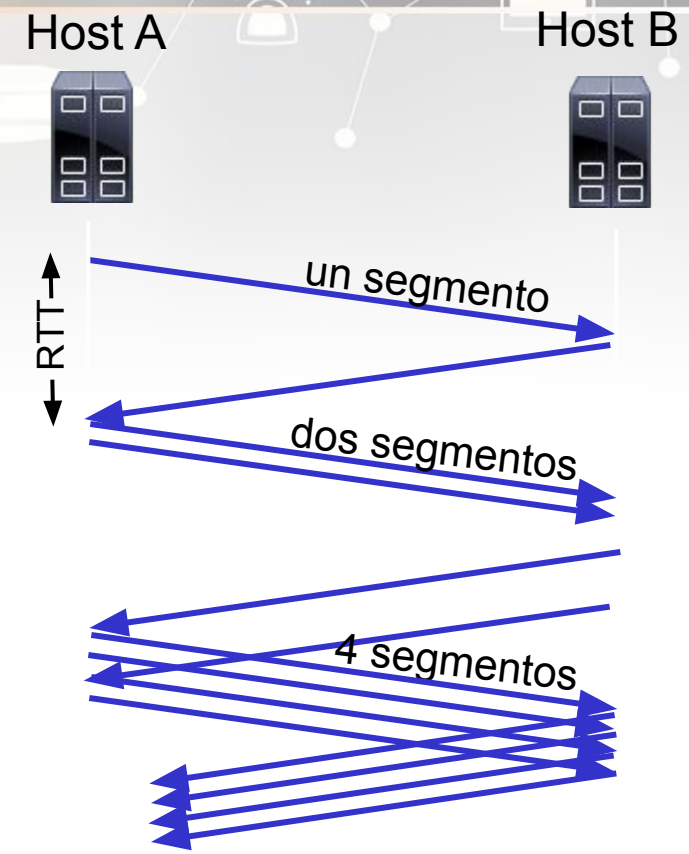
Control de congestión en TCP

AIMD: *addtive increase multiplicative decrease*

- RFC 5681
- El emisor incrementa tasa de transmisión (tamaño de ventana) hasta que detecta pérdida de paquete (timeout)
  - *additive increase*: incrementa tamaño de ventana en relación al MSS (*maximum segment size*) por cada RTT (*round trip time*) hasta que detecta pérdida
  - *multiplicative decrease*: al detectar pérdida reduce tamaño de ventana a la mitad

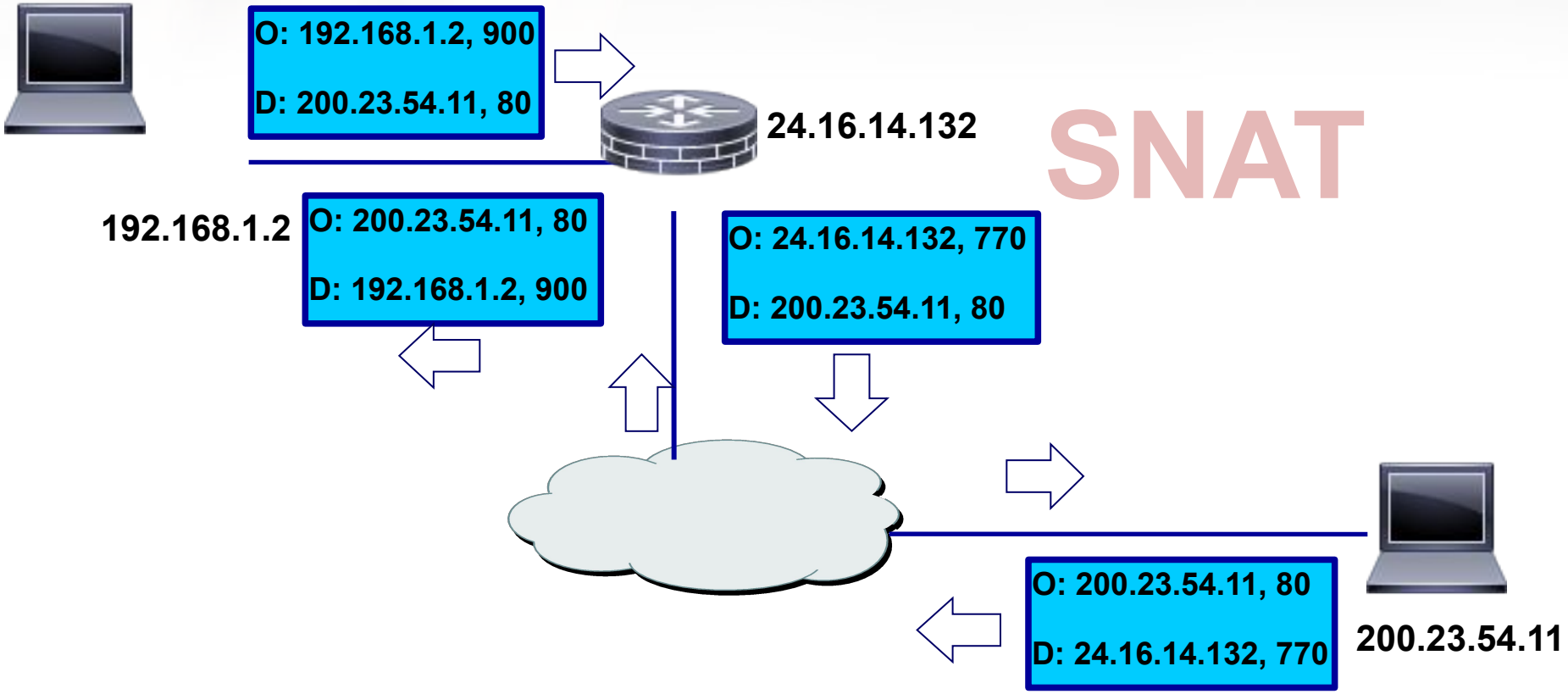
TCP slow start

- cuando la conexión inicia incrementar la tasa hasta el primer timeout:
  - **cwnd** inicial = 1 MSS
  - duplicar **cwnd** cada RTT
  - se hace incrementando **cwnd** por cada ACK recibido



NAPT (*aka* NAT)

Hasta mediados de los '90, para usar Internet se necesitaba una IP pública en el host.



NAT / NAPT

Tabla que mantiene el firewall para NAPT

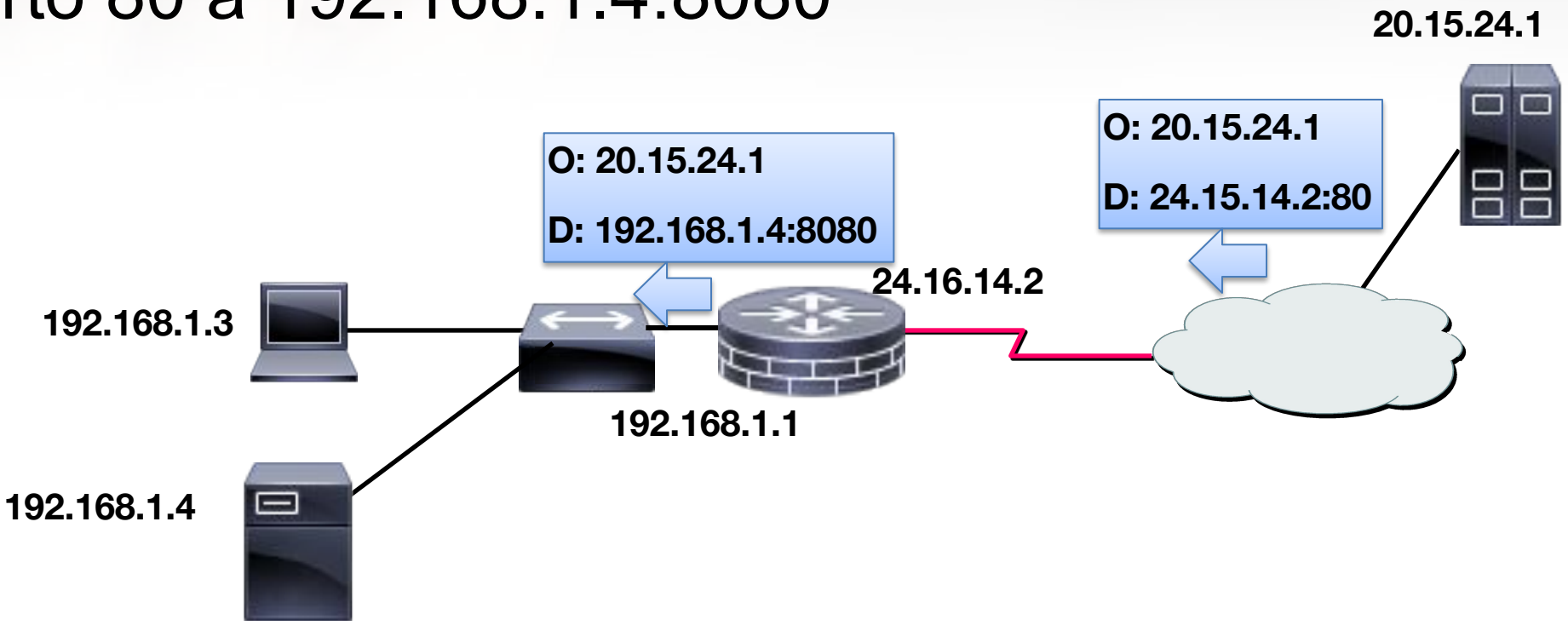
IP origen	Pto origen	IP destino	Pto destino	IP salida	Pto salida
192.168.1.2	900	200.32.54.11	80	24.16.14.132	770
192.168.1.3	900	200.32.54.11	80	24.16.14.132	771

+ DNS Caching-only Server

+ Firewall

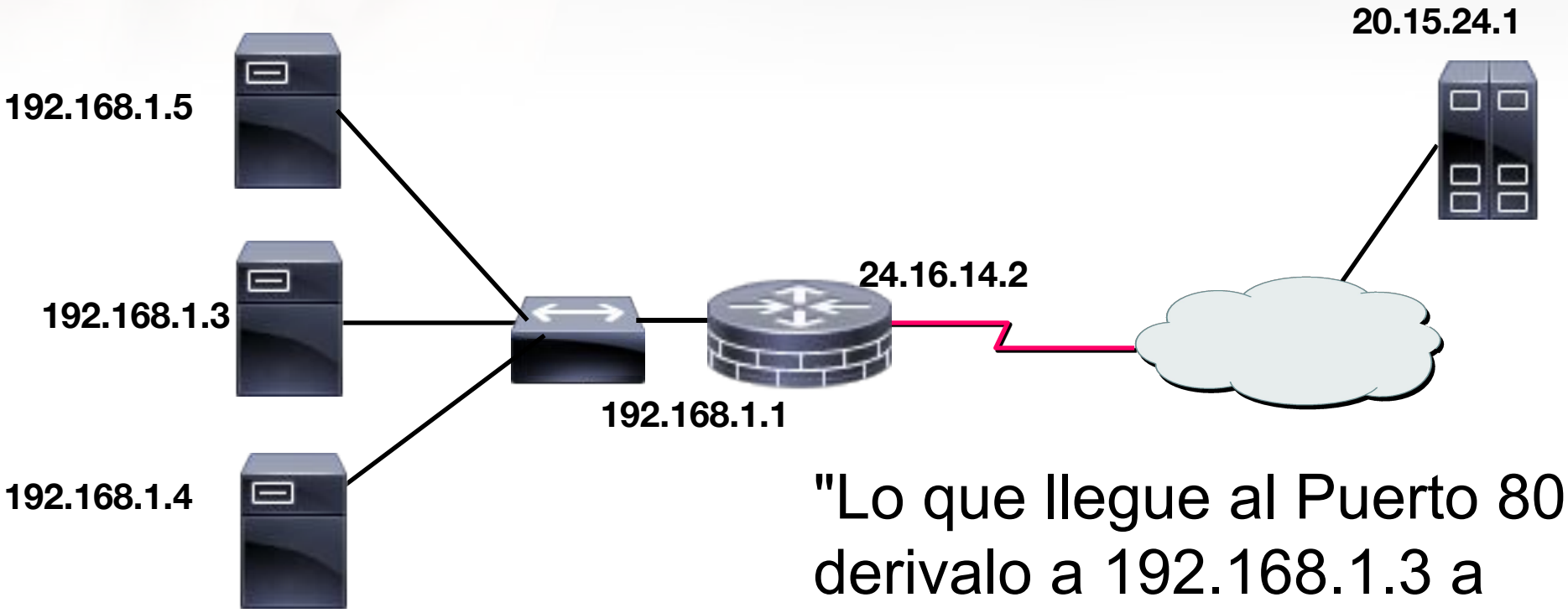
DNAT (*port forwarding*)

- Quiero conectarme a un servidor en 192.168.1.4:8080
- configuro NAT para reenviar conexiones entrantes de puerto 80 a 192.168.1.4:8080



DNAT (*port forwarding*)

- Permite el balance de carga
- Internamente n servidores, a una única IP pública

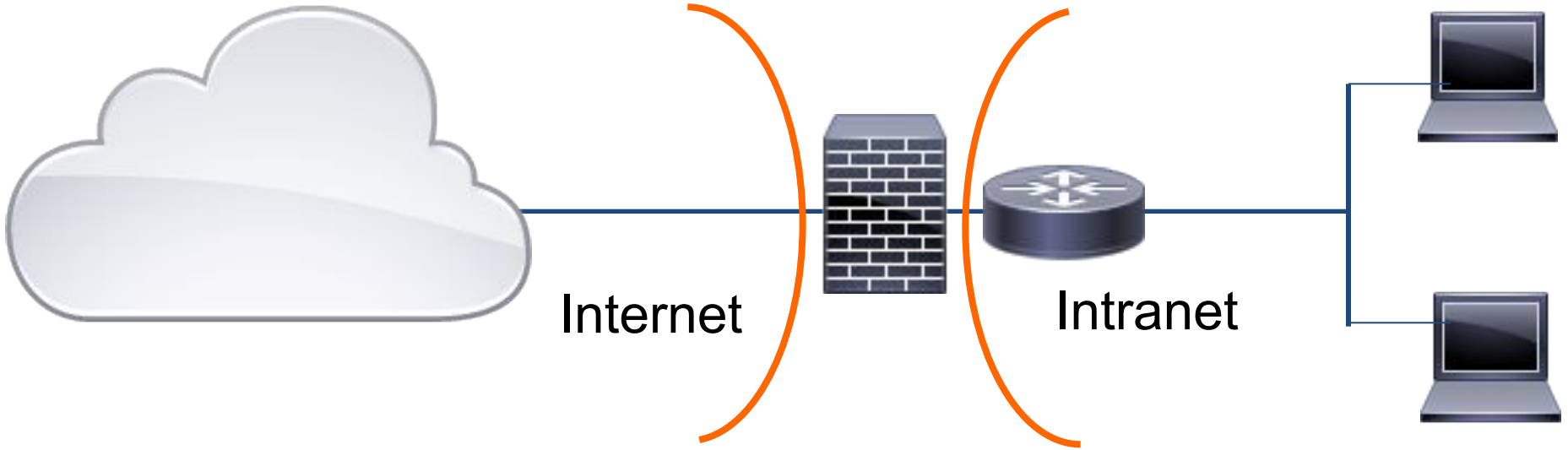


"Lo que llegue al Puerto 80 derivalo a 192.168.1.3 a 192.168.1.5"

Firewalls

Un firewall opera en el nivel más bajo de red:

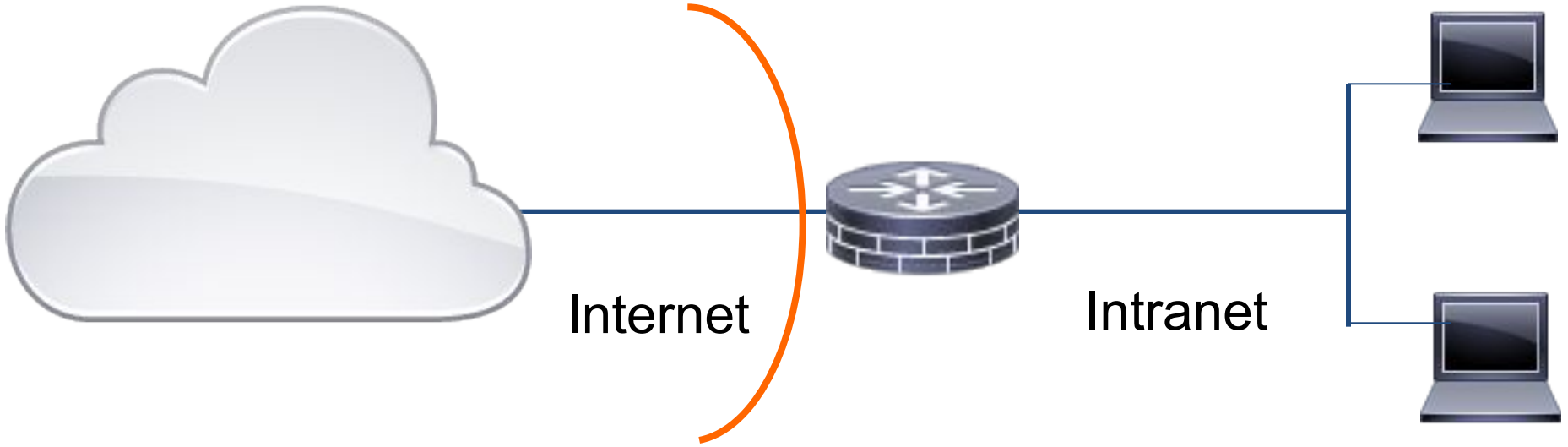
- Evita accesos no autorizados a la Intranet
- Permite definir qué tráfico entra y sale de nuestra red



Firewalls

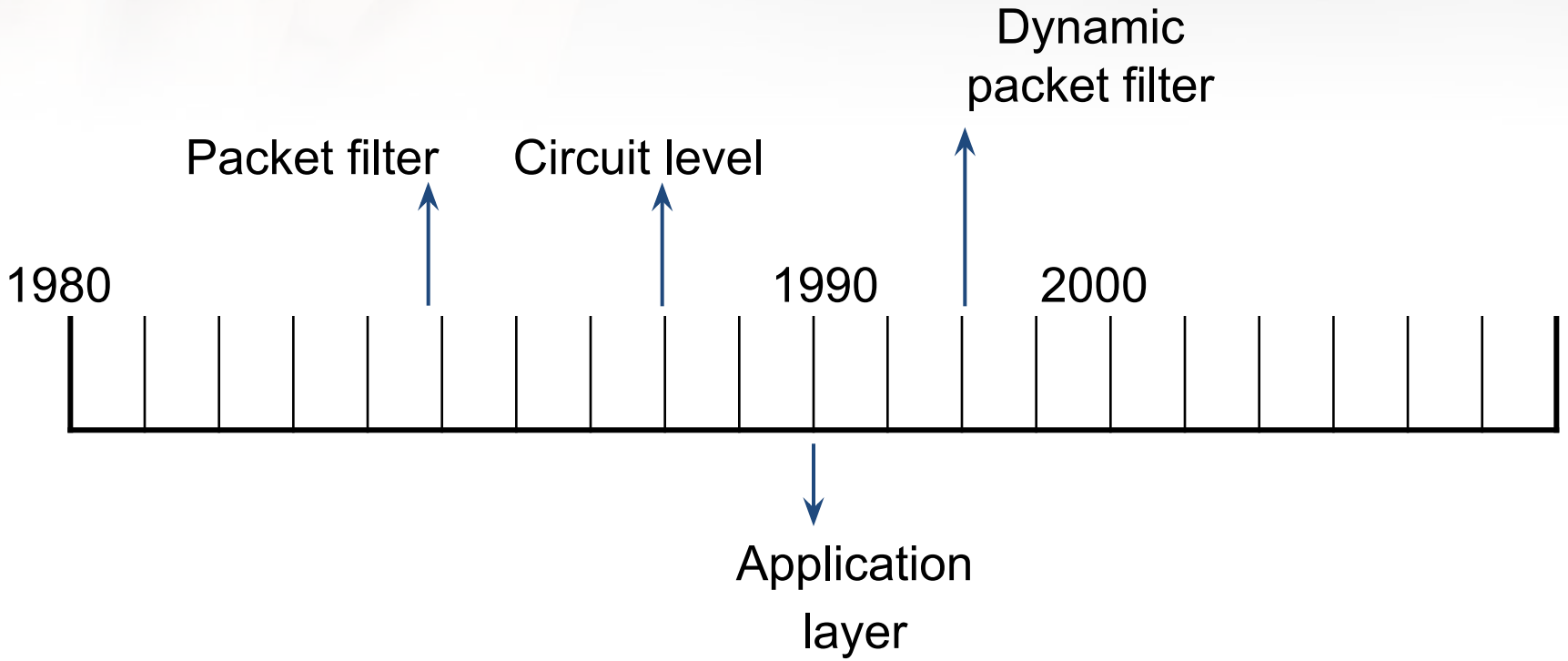
Un firewall opera en el nivel más bajo de red:

- Evita accesos no autorizados a la Intranet
- Permite definir qué tráfico entra y sale de nuestra red



Firewalls

Evolución de la arquitectura de firewalls

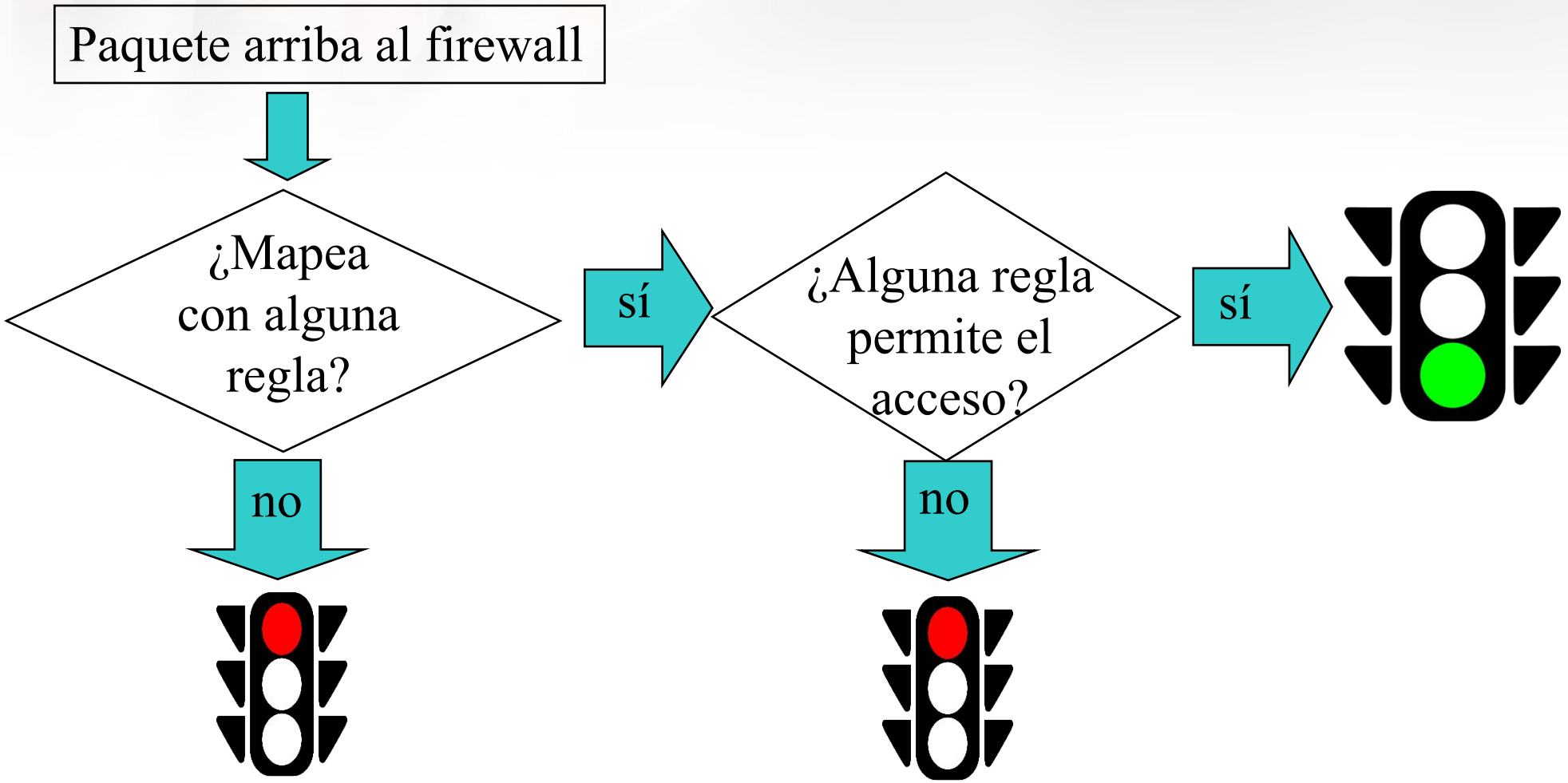


Firewalls: *packet filter*

- Analiza tráfico de red a nivel de transporte
- Cada paquete IP es evaluado para ver si satisface o no una serie de reglas
- Las reglas se pueden basar en:
  - Interface en la cual arriba
  - Dirección de origen y/o destino (dirección IP)
  - Protocolo que transporta (UDP, TCP, ICMP)
  - Puerto de origen y/o destino
- La acción a seguir podrá ser *deny* o *allow*.

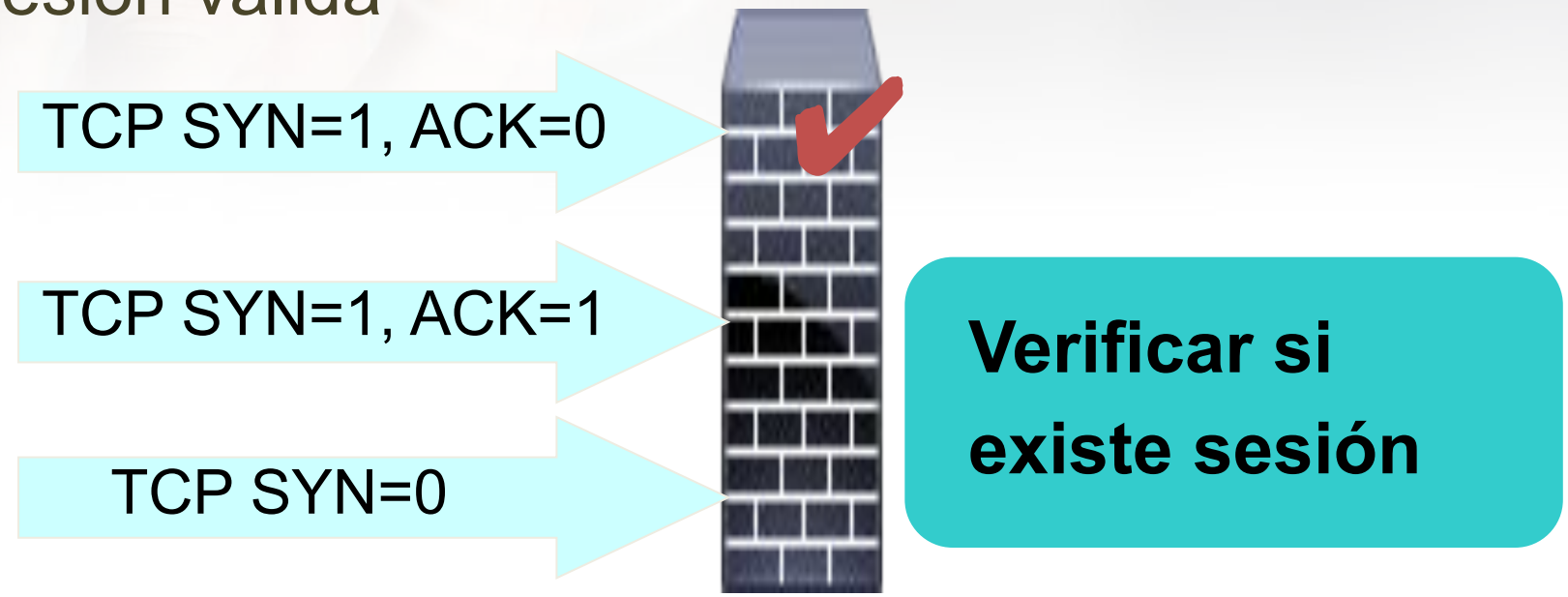
Firewalls: *packet filter*

Algoritmo de inspección de paquetes



Firewalls: *circuit level*

Pueden validar si un paquete pertenece a una sesión válida



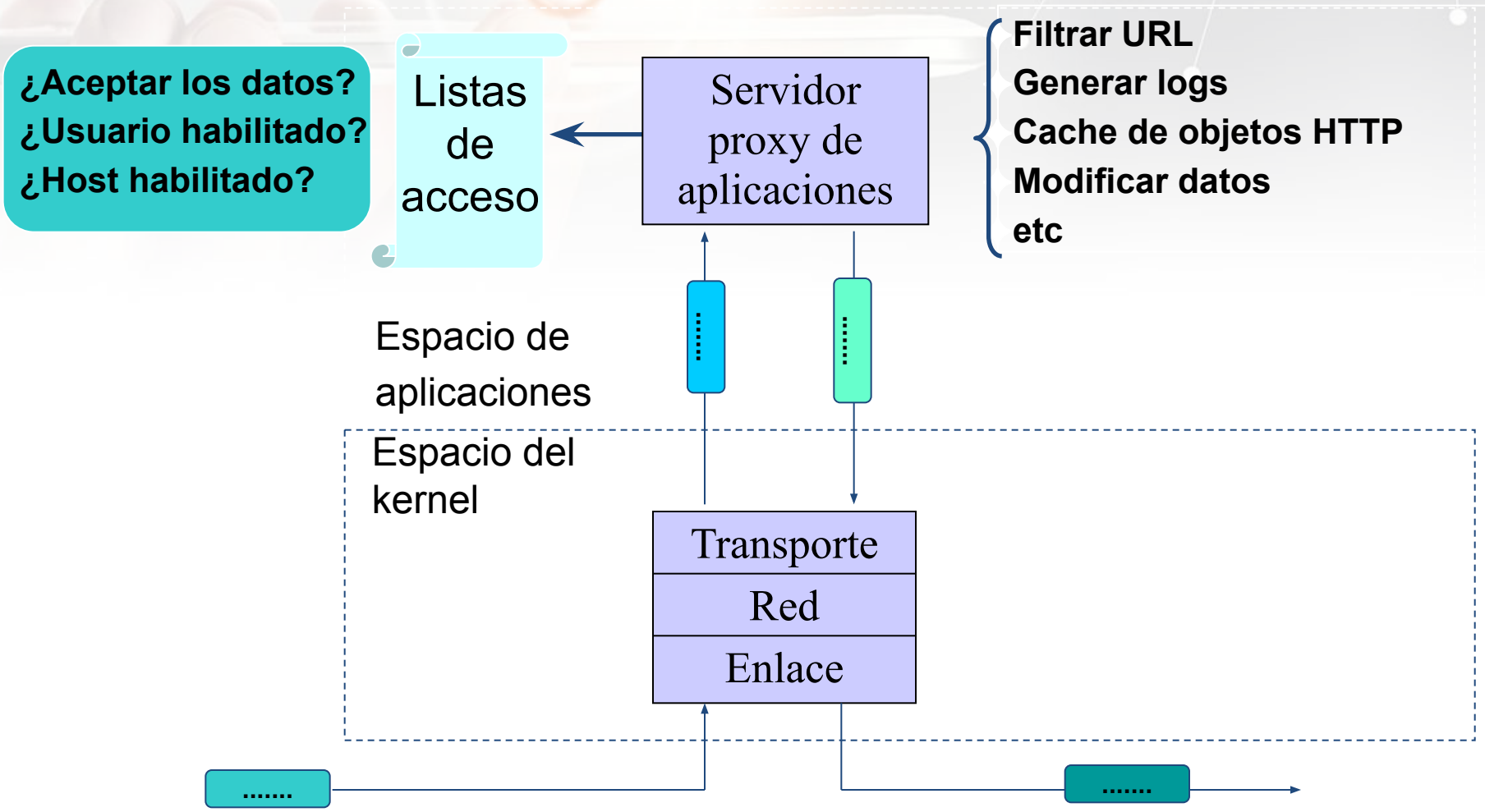
Firewalls: *Circuit level*

Una vez establecida una sesión se suele almacenar:

- Identificador de la conexión
- Estado de la conexión: *handshake, established, closing*
- Números de secuencia
- IP origen y destino
- Puertos de origen y destino
- Interface de red por la que arriban los paquetes
- Interface de red por la que salen los paquetes

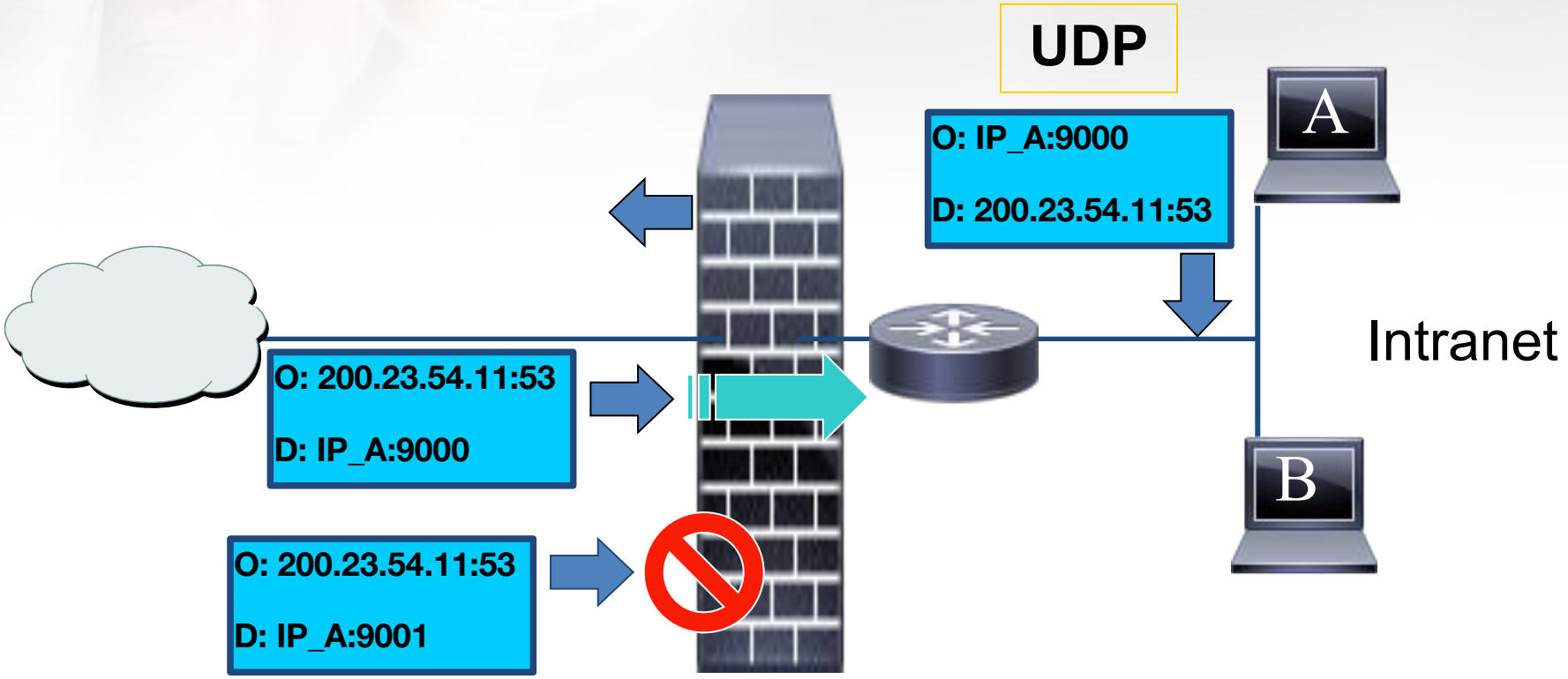
¿UDP?

Firewalls: *application layer*



Firewalls: *Dynamic packet filter*

Permiten la modificación de reglas "*on the fly*"



Material de lectura

Capítulos 3.1 a 3.3 y 3.5 a 3.7 inclusive de la bibliografía