Contenido

- Internet Protocol
  - Características
  - IP en LAN Ethernet (ARP)
  - Direccionamiento (subredes, Proxy-ARP, CIDR)
  - Fragmentación y reensamblado
  - ICMP
- Enrutamiento en Internet
  - Distance-Vector
  - Link-State
  - Path-Vector
- UDP
- TCP
  - Características
  - Gestión de conexiones
  - Control de flujo
  - Retransmisiones
  - Control de congestión

Bibliografía

- [Kurose05] James K. Kurose, Keith W. Ross, *"Computer Networking. A top-Down Approach Featuring the Internet"*, Ed. Addison-Wesley
- [Tanenbaum05] Andrew S. Tanenbaum, *"Computer Networks"*, Ed. Prentice Hall
- [Forouzan03] Behrouz A. Forouzan, *"TCP/IP Protocol Suite"*, Ed. Mc Graw Hill
- [Stevens] W. Richard Stevens, *"TCP/IP Illustrated, Volume 1, The Protocols"*, Ed. Addison-Wesley
- [Perlman] Radia Perlman, *Interconnections Second Edition, Bridges, Routers, Switches and Internetworking Protocols"*, Ed. Addison-Wesley

Contenido

- Internet Protocol
  - **Características**
  - IP en LAN Ethernet (ARP)
  - Direccionamiento (subredes, Proxy-ARP, CIDR)
  - Fragmentación y reensamblado
  - ICMP
- Enrutamiento en Internet
  - Distance-Vector
  - Link-State
  - Path-Vector
- UDP
- TCP
  - Características
  - Gestión de conexiones
  - Control de flujo
  - Retransmisiones
  - Control de congestión

Lecturas recomendadas

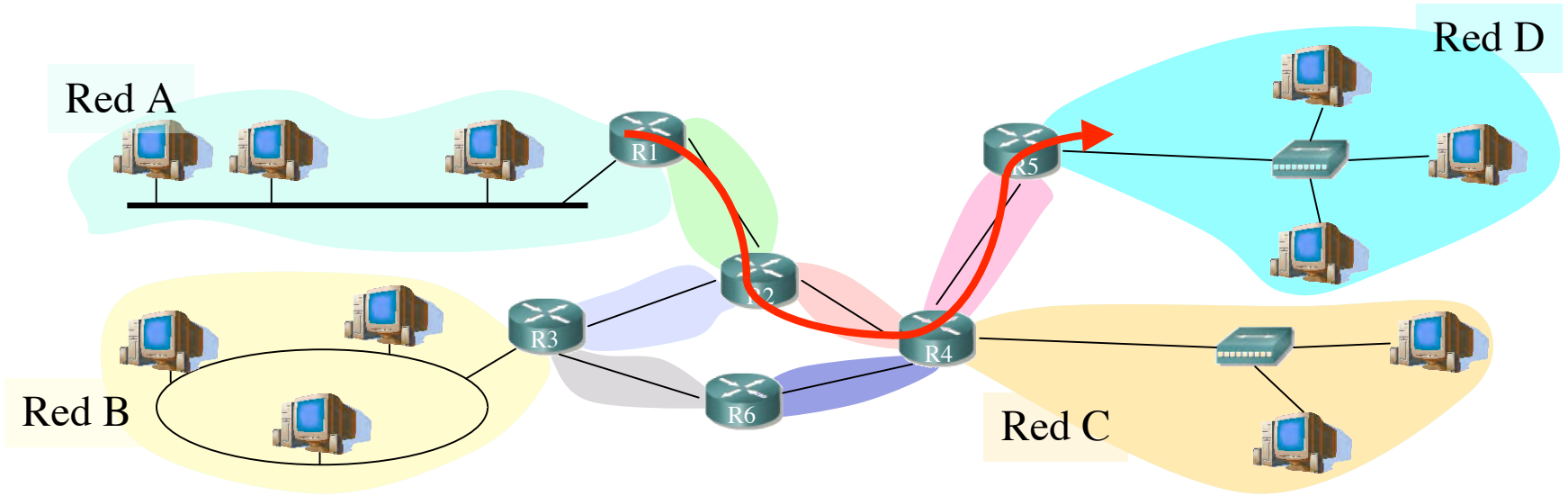
- [Tanenbaum05] 5.5-5.5.4, 5.6-5.6.1

Nivel de red

- Objetivo:
  - Llevar paquetes del origen al destino
  - Usar los enlaces de forma “eficiente”
- Direccionamiento:
  - Que permita identificar a los nodos
  - Tiene una estructura (no es plano)
  - Ésta reduce la información en los routers
- Enrutamiento
  - Elementos de encaminamiento deben “aprender” cómo es la red
  - Deben cacular “buenos” caminos a los destinos
  - Esto se almacena en las “tablas de rutas”

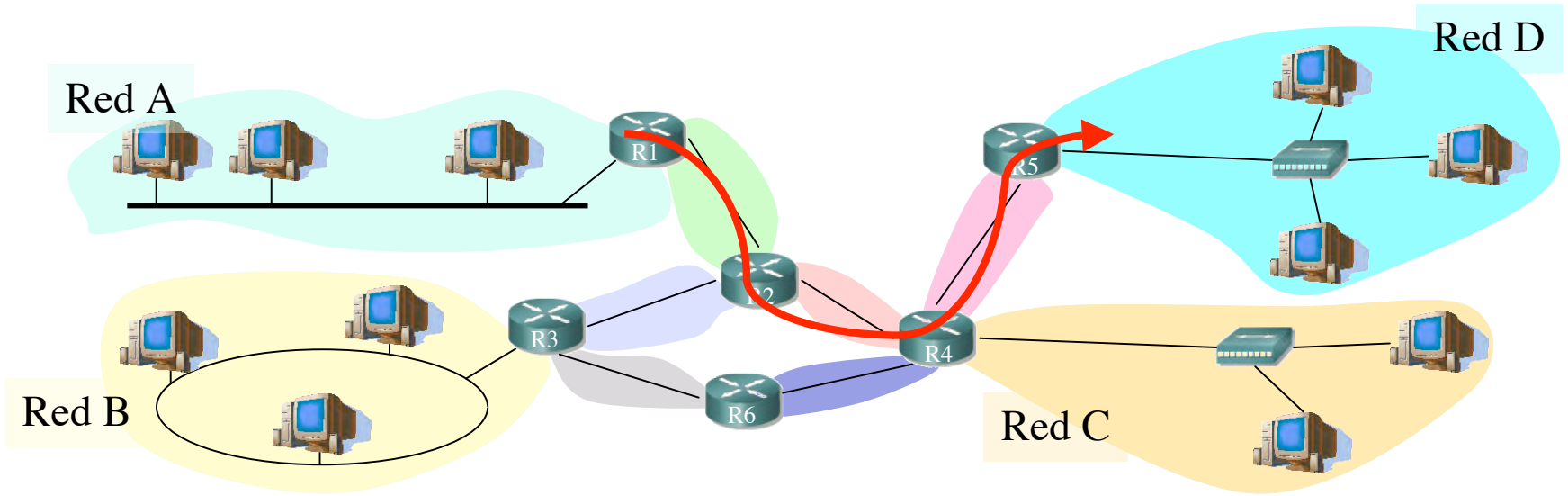
Routing

- "Ruta" es un camino (**path**) ⇒ acíclico …
- "Routing" = proceso de **calcular los caminos** que deben seguir los paquetes
- Se pueden calcular en función de:
  - Flujo
  - Tipo de tráfico
  - (origen, destino)
  - Destino



Conmutación

- Reenviar los bits por el camino
- Servicios posibles
  - Circuitos (telefonía, longitud de onda)
  - Paquetes
    - Circuitos virtuales … → Cada paquete del mismo flujo sigue la misma ruta
    - Datagramas … → Cada paquete es conmutado independientemente



Características de IP

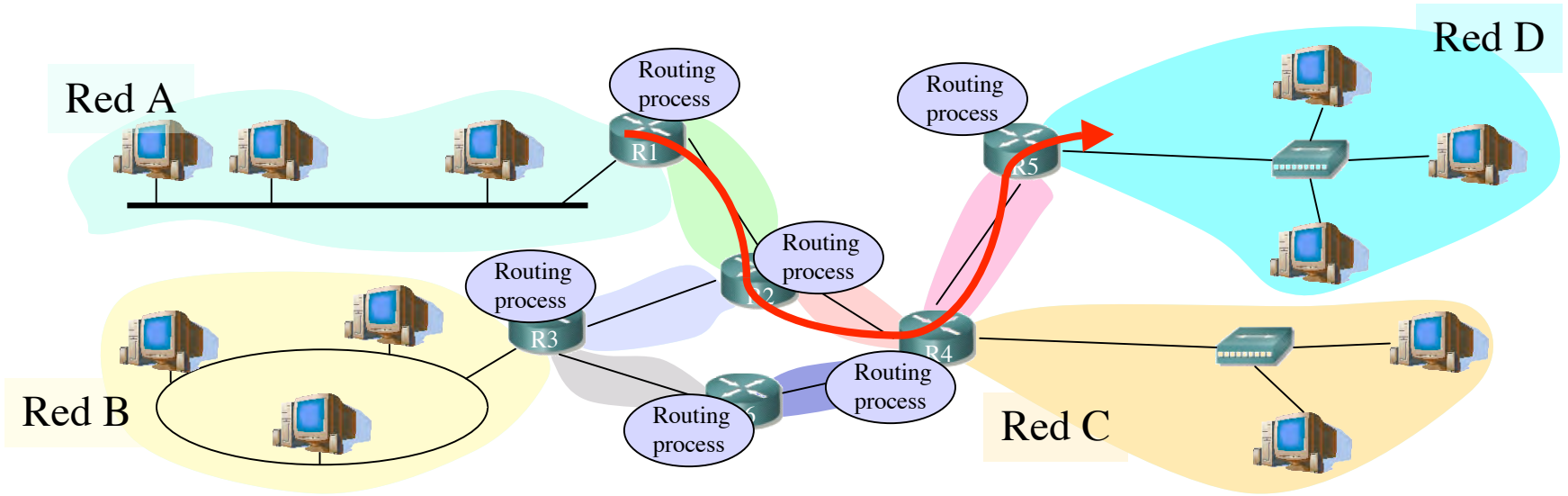
- Nivel de red
- Servicio de datagramas, sin conexión
- Routing en función de la dirección destino
- No fiable
- Best effort
- Provee:
  - Independencia de las tecnologías de cada red
  - Direccionamiento global
  - TOS
  - Fragmentación y reensamblado

Otros aspectos

- Direccionamiento
  - Nivel 2: local, plano ⇒ no escalable
  - Nivel 3: según lugar, jerárquico ⇒ escalable
  - Direcciones temporales
  - Network Address Translation para reducir direcciones
- Routing
  - Basado en la dirección destino
  - La red se descompone en dominios
  - Routing *interdomain* : algoritmo *path-vector*
  - Routing *intradomain* : link state o distance vector
- Más
  - Multicast; Ad-hoc; P2P; Sensores, etc

Routing en IP

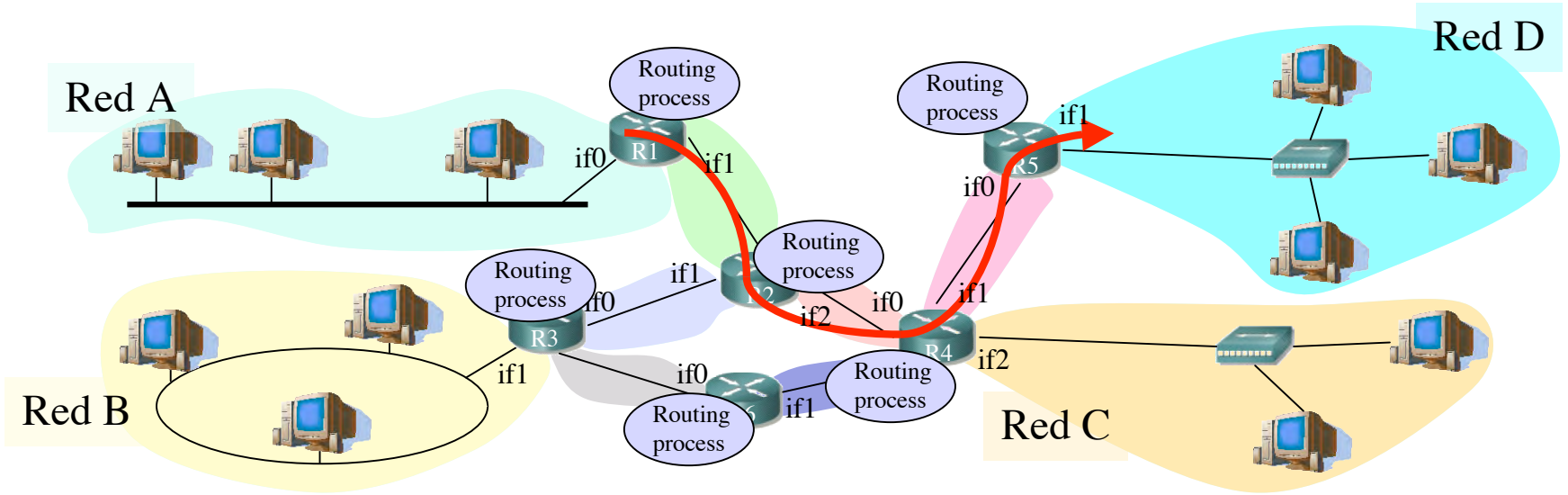
- Llevado a cabo por un **proceso** que se ejecuta en cada router (cálculo distribuido) …
- Resultado: una "**tabla de rutas**" en cada router …



Routing en IP

- Llevado a cabo por un **proceso** que se ejecuta en cada router (cálculo distribuido) …
- Resultado: una "**tabla de rutas**" en cada router …

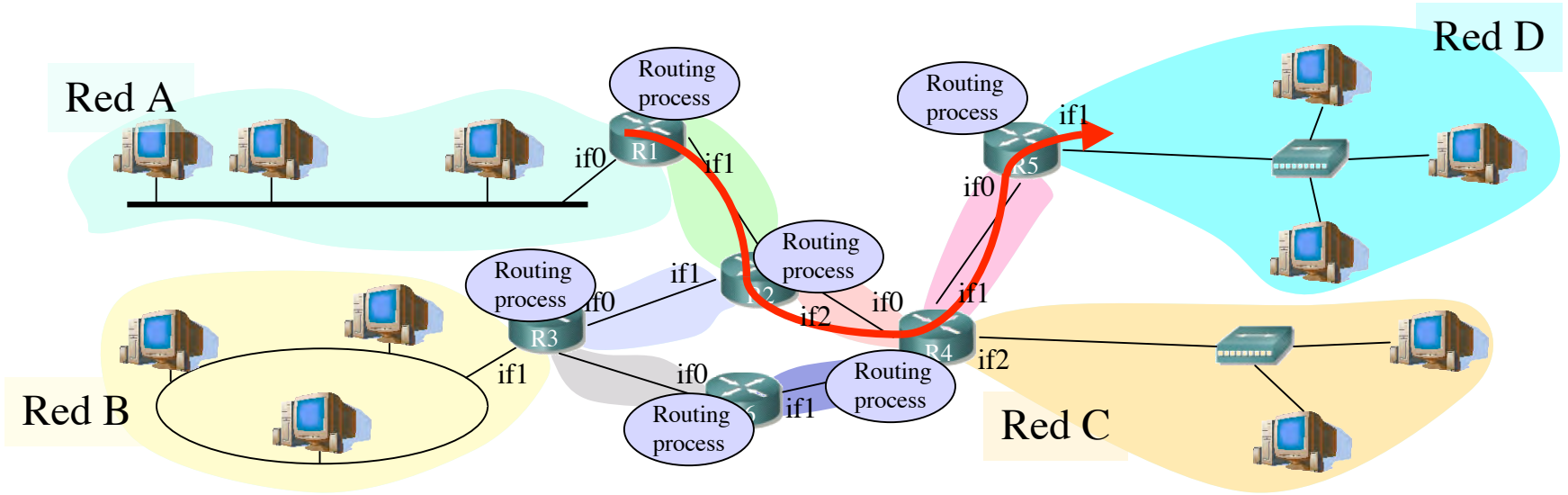
Destino	Next-hop
Red A	IP de if1 de R1
Red B	IP de if0 de R3
Red C	IP de if0 de R4
Red D	IP de if0 de R4
…	…



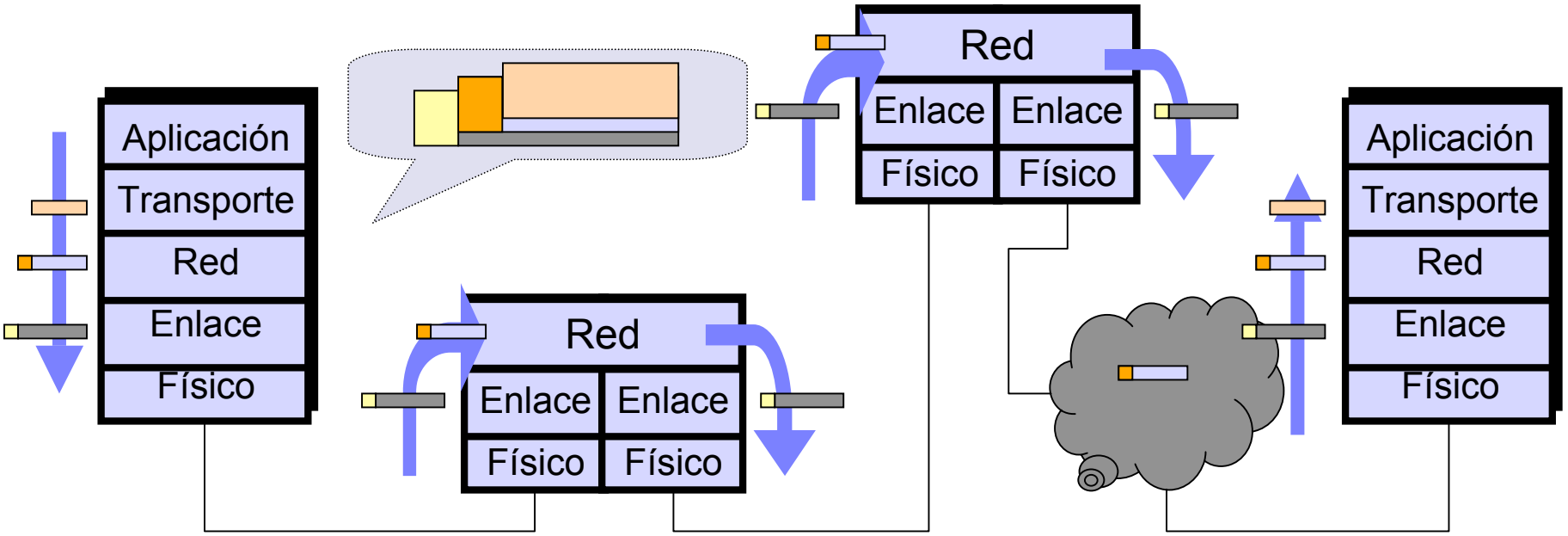
Forwarding en IP

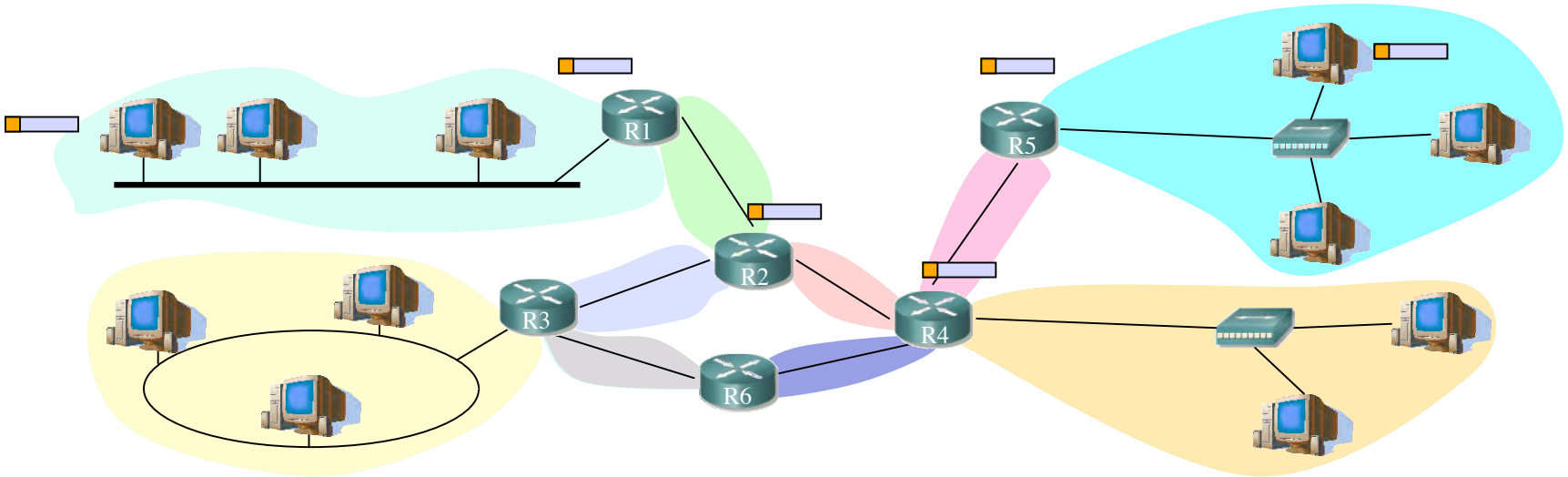
- Tarea de "**reenviar**" por el interfaz adecuado el paquete recibido
- En base a la tabla de rutas del router
- La tabla indica cuál es el siguiente router (**next-hop**) en el camino
- El router tendrá **conectividad a nivel 2** con él

Destino	Next-hop
Red A	IP de if1 de R1
Red B	IP de if0 de R3
Red C	IP de if0 de R4
Red D	IP de if0 de R4
…	…



Encapsulación

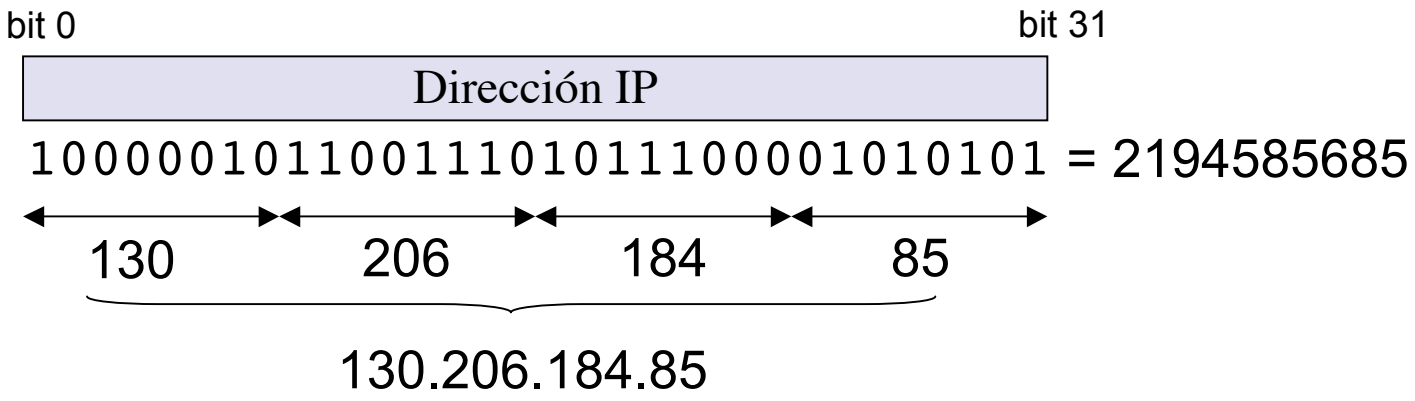




Algunas características de IP

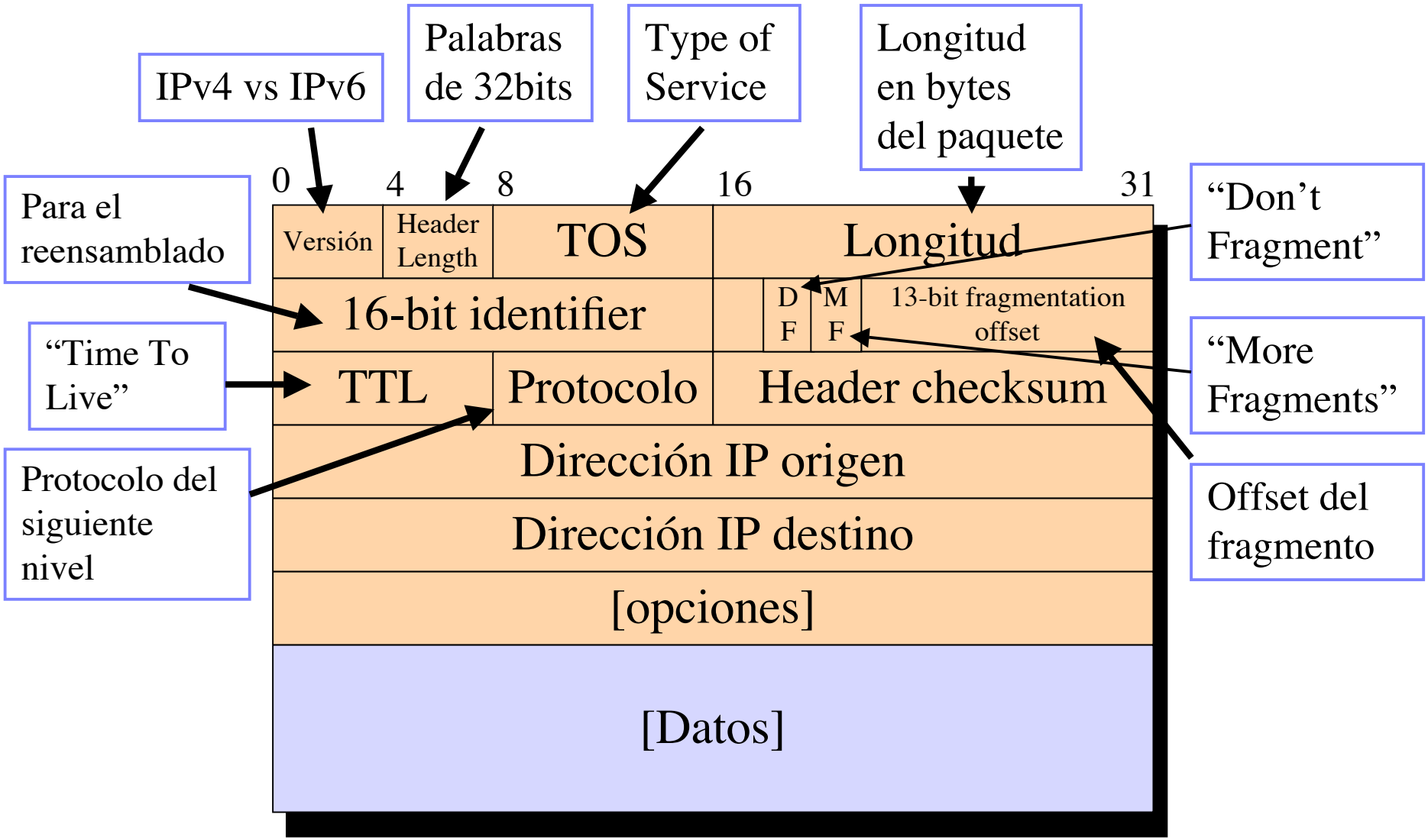
- Muy bueno en escalabilidad
  - Millones de nodos
  - Tablas de rutas deben ser “pequeñas”
  - Actualizaciones deben ser “manejables”
- Bueno ante cambios de topología
  - Los routers calculan nuevas rutas
  - Los cambios no afectan a la mayoría
- Pobre rendimiento
  - Utilización de los enlaces no se balancea
  - Las actualizaciones no son muy rápidas
  - Algunos flujos deberían tener garantías de calidad
  - No detecta errores de configuración
  - No se protege ante ataques

Representación de las direcciones



- Números de 32 bits cómodos para computadoras, no para humanos
- Representación “dotted-decimal”

Formato del datagrama IP



Contenido

- Internet Protocol
  - Características
  - **IP en LAN Ethernet (ARP)**
  - Direccionamiento (subredes, Proxy-ARP, CIDR)
  - Fragmentación y reensamblado
  - ICMP
- Enrutamiento en Internet
  - Distance-Vector
  - Link-State
  - Path-Vector
- UDP
- TCP
  - Características
  - Gestión de conexiones
  - Control de flujo
  - Retransmisiones
  - Control de congestión

Lecturas recomendadas

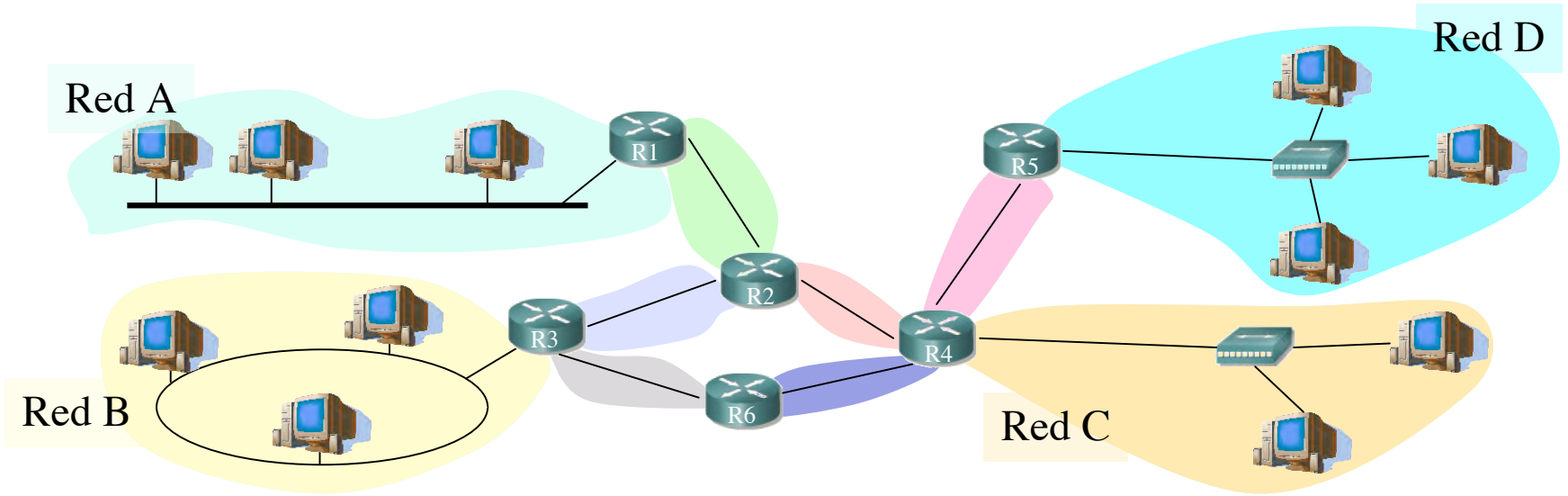
- [Kurose05] 5.4
- [Stevens] 4

Contenido

- Introducción
- ARP
  - Motivación
  - Funcionamiento
  - Ejemplos
- Comunicación entre hosts en distintas LANs

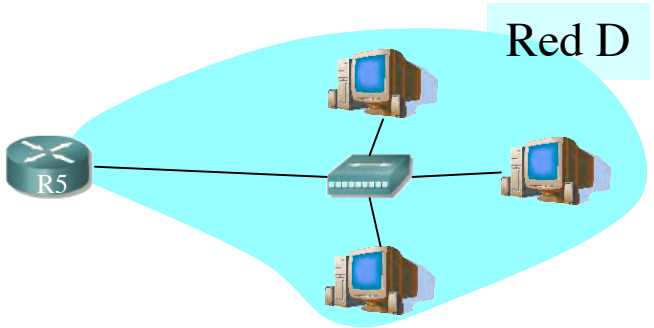
Introducción

- Nivel de red permite que paquetes lleguen de unas redes a otras
- Dentro de cada red depende de la tecnología (nivel *Host a Red*)
- Veamos cómo se realiza la comunicación en el caso de una LAN Ethernet …



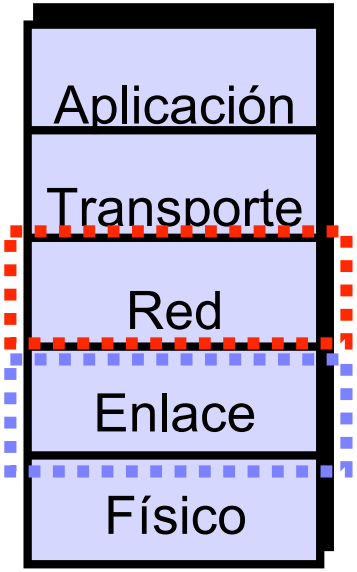
Introducción

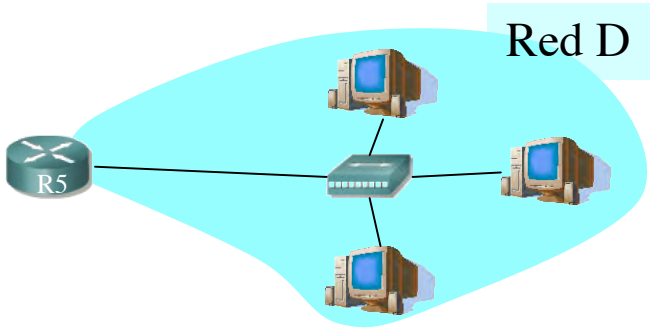
- Nivel de red permite que paquetes lleguen de unas redes a otras
- Dentro de cada red depende de la tecnología (nivel *Host a Red*)
- Veamos cómo se realiza la comunicación en el caso de una LAN Ethernet …



Direcciones IP y MAC

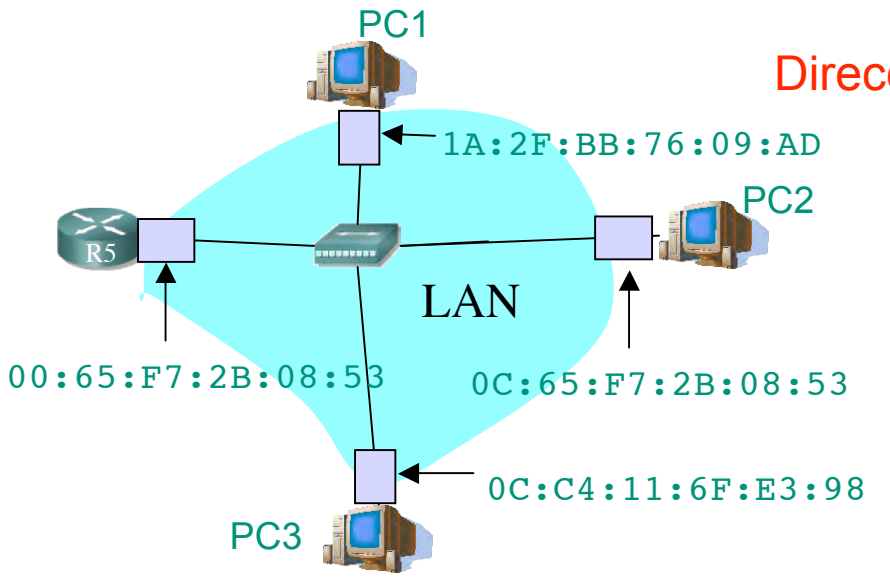
- Direcciones IP:
  - Direcciones del nivel de red, de **32 bits**
  - Empleadas para que el **datagrama** llegue a la red IP destino
  - *Lógicas*
- Direcciones MAC (direcciones LAN o físicas o Ethernet):
  - Para que una **trama** llegue de un interfaz a otro físicamente conectado en la **misma red, de 48 bits** en la mayoría de LANs
  - A fuego en la ROM de la tarjeta
  - *Físicas*





Direcciones MAC y ARP

Cada tarjeta en la LAN tiene una dirección MAC única

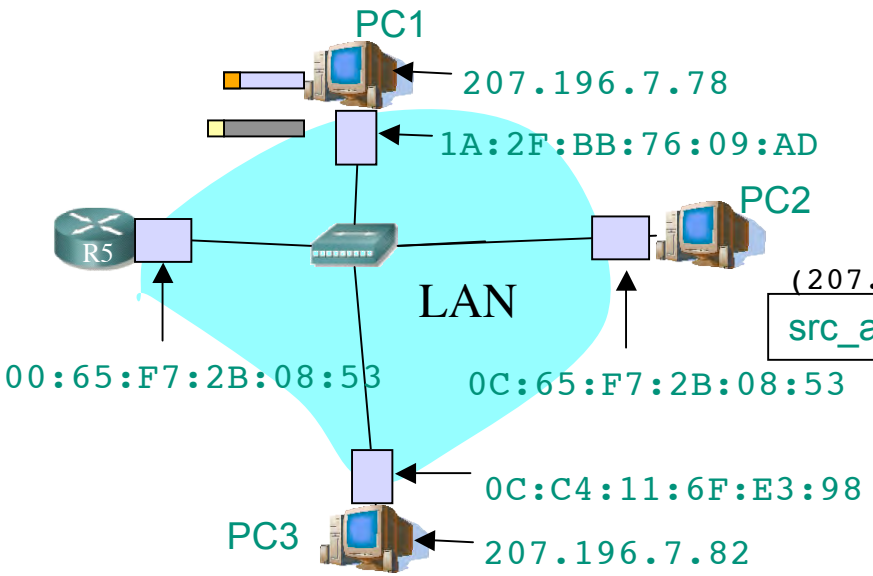


Dirección de broadcast = FF-FF-FF-FF-FF-FF

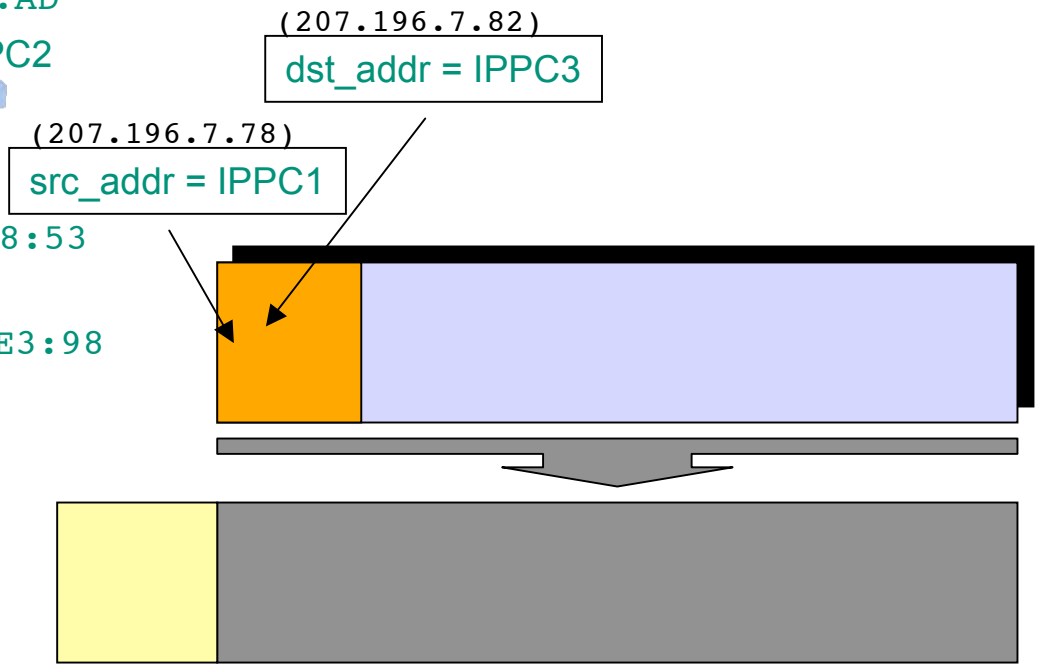
= tarjeta

ARP: Address Resolution Protocol

- **¿Cómo enviar un paquete IP de un nodo a otro de la misma red?**

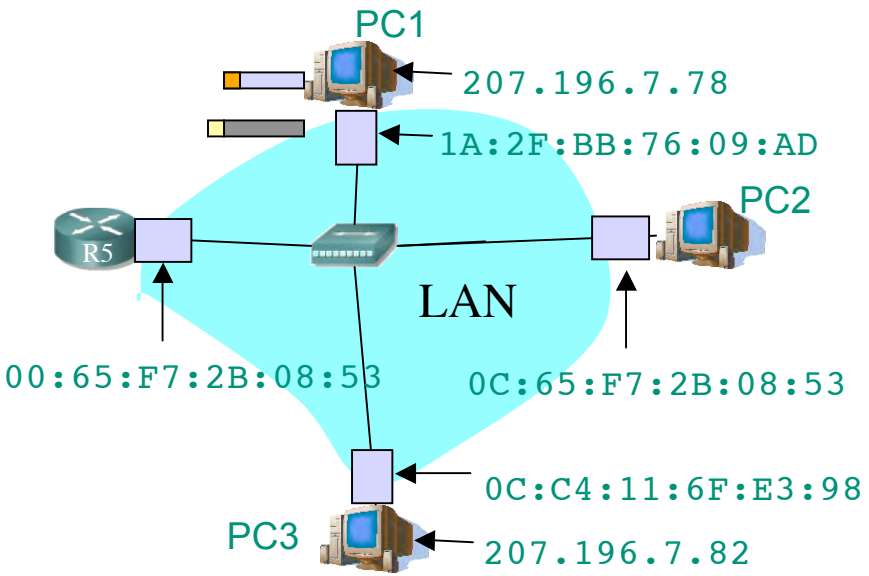


- Ejemplo: Paquete IP de 207.196.7.78 a 207.196.7.88
- Deberá ir en una trama Ethernet…

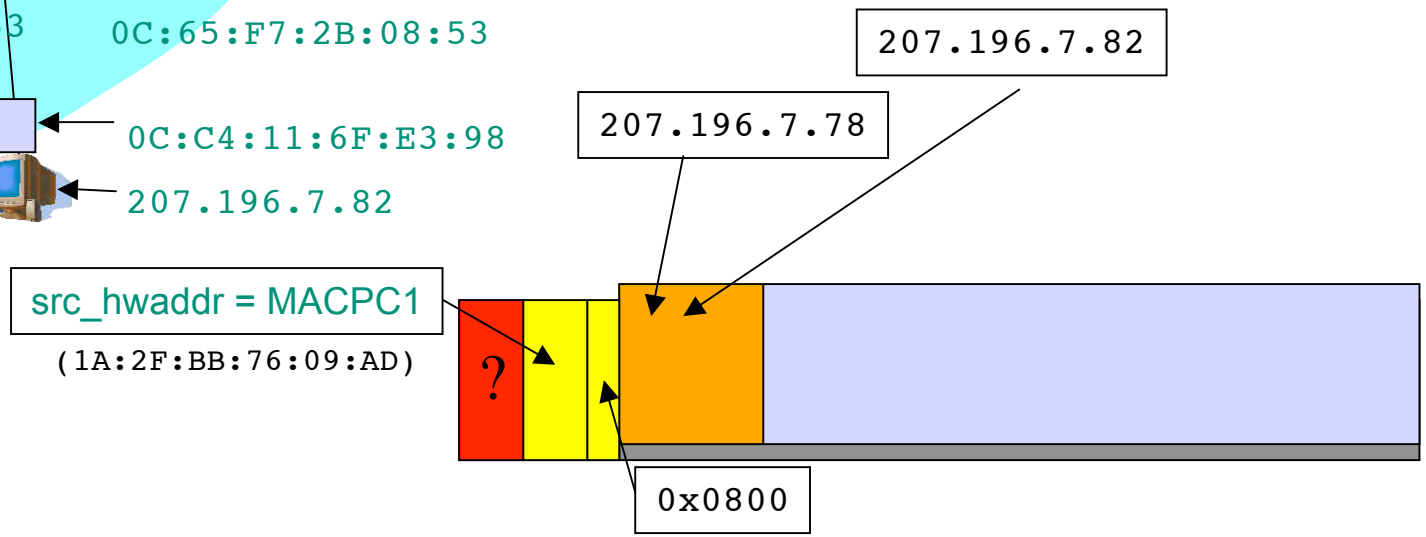


ARP: Address Resolution Protocol

- **¿Cómo enviar un paquete IP de un nodo a otro de la misma red?**

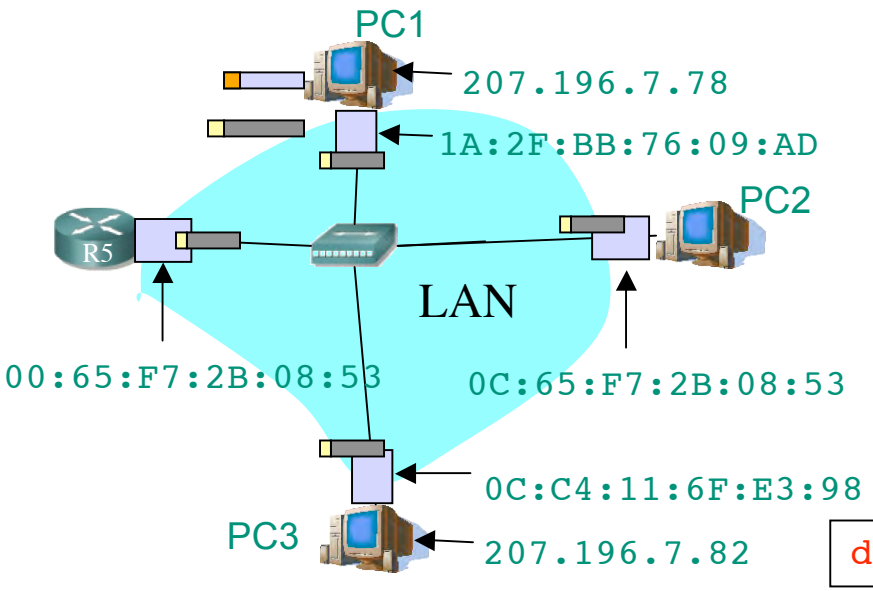


- Ejemplo: Paquete IP de 207.196.7.78 a 207.196.7.88
- Deberá ir en una trama Ethernet…
  - Ethertype: 0x0800 (IP)
  - MAC origen la de la tarjeta que envía
  - MAC destino…



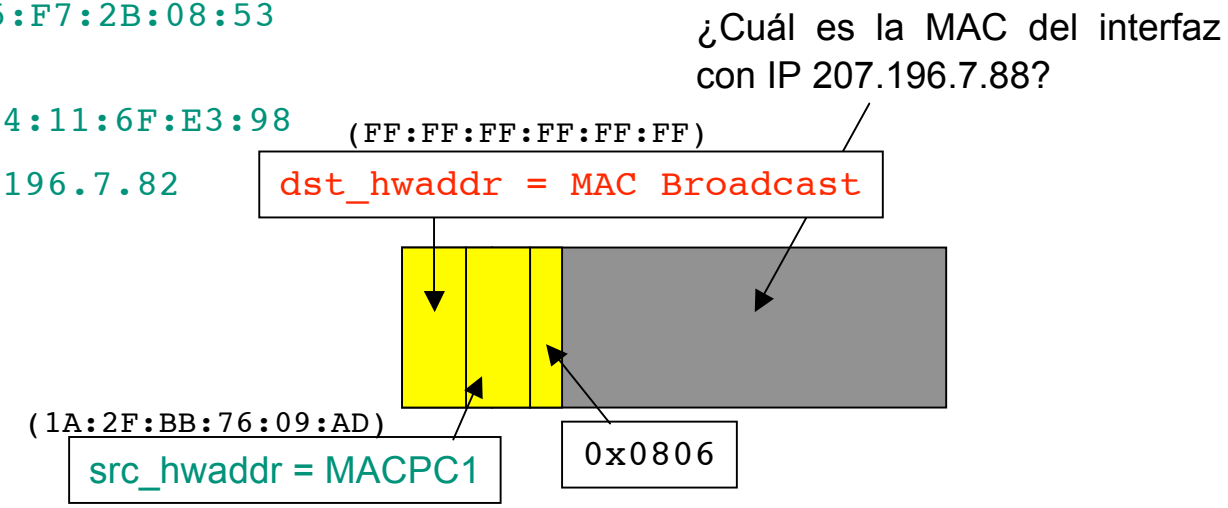
ARP: Address Resolution Protocol

- **¿Cómo enviar un paquete IP de un nodo a otro de la misma red?**



ARP

1. ARP Request: El emisor envía una trama ARP a la dirección MAC de broadcast (FF:FF:FF:FF:FF:FF). Contiene la IP destino
2. Todos los interfaces de la LAN leen esa trama…

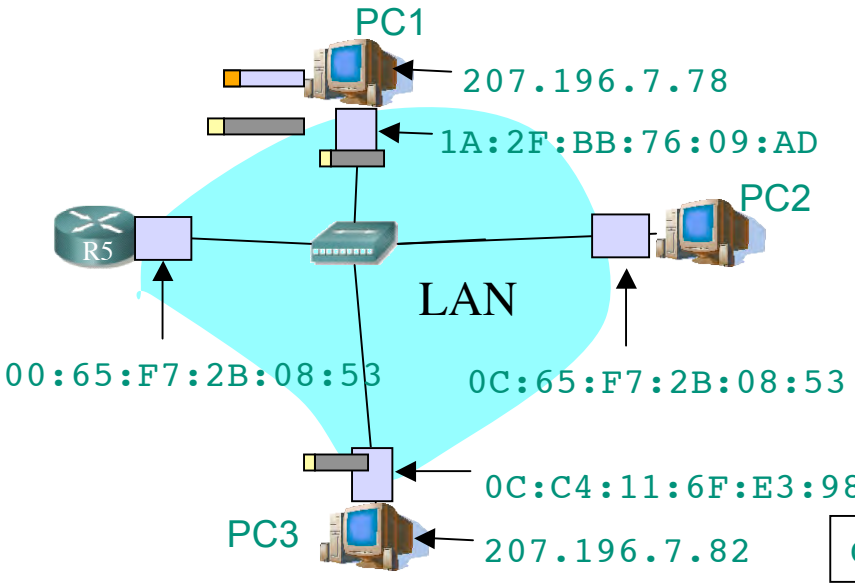


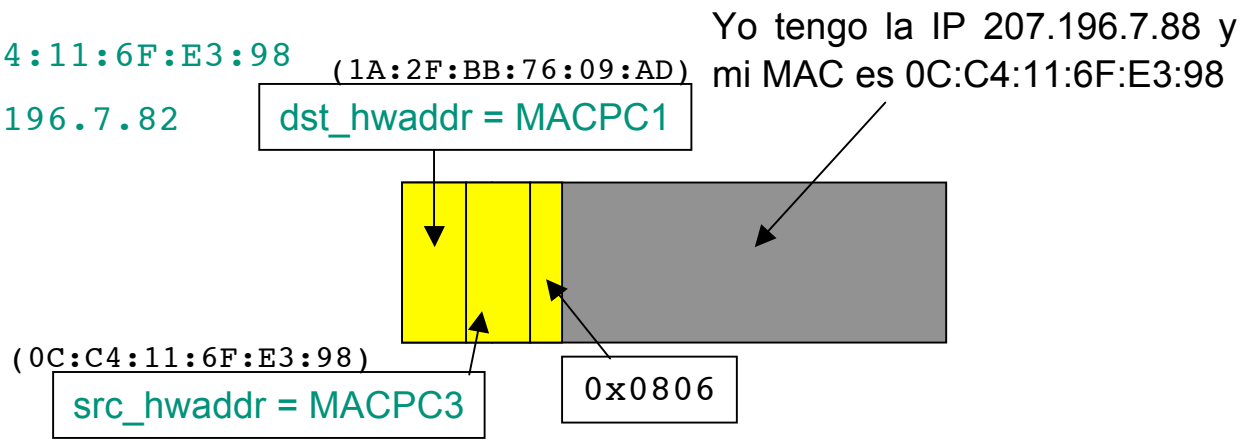
ARP: Address Resolution Protocol

- **¿Cómo enviar un paquete IP de un nodo a otro de la misma red?**

ARP

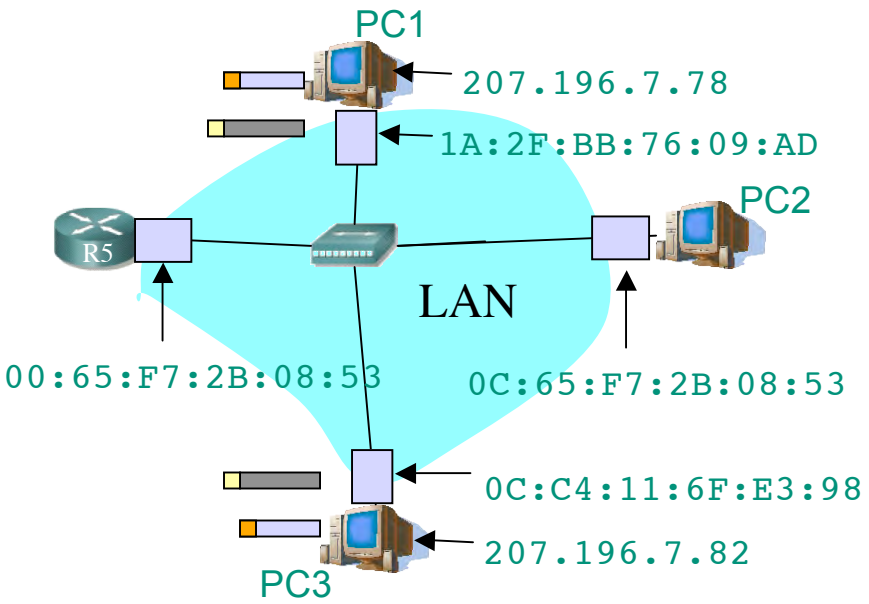
3. ARP Reply: El interfaz con esa IP responde con otra trama ARP…





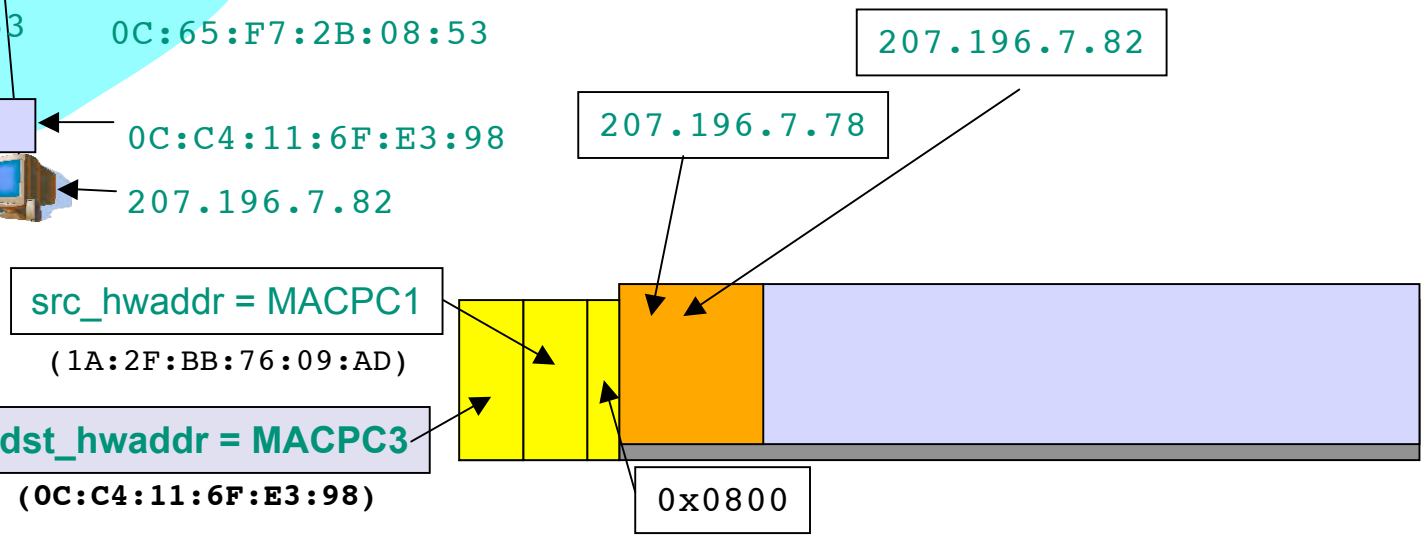
ARP: Address Resolution Protocol

- **¿Cómo enviar un paquete IP de un nodo a otro de la misma red?**

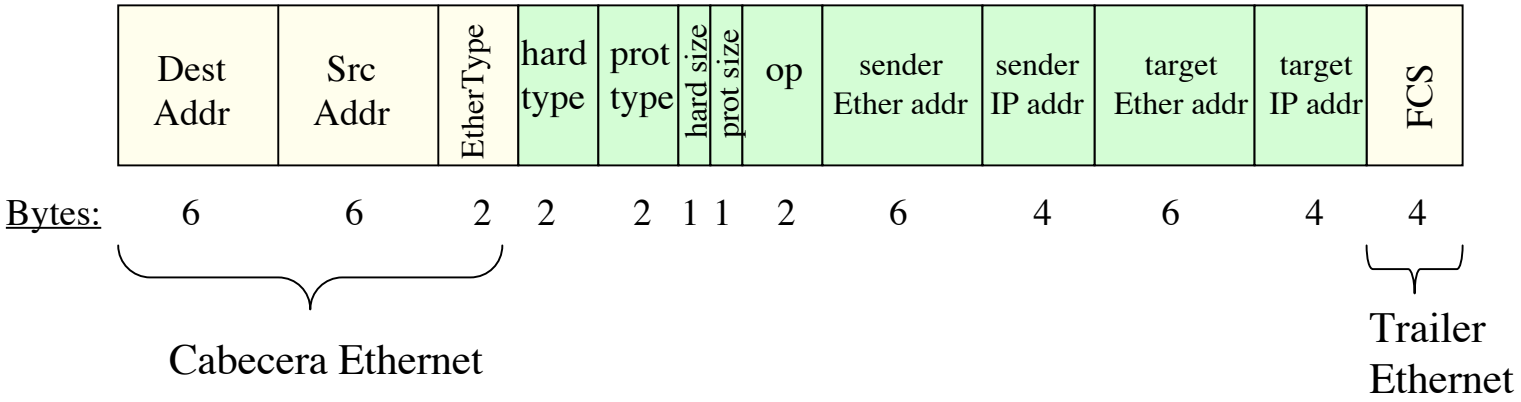


- Ahora puede colocar la MAC destino…
- Y enviarla…

Entrega directa



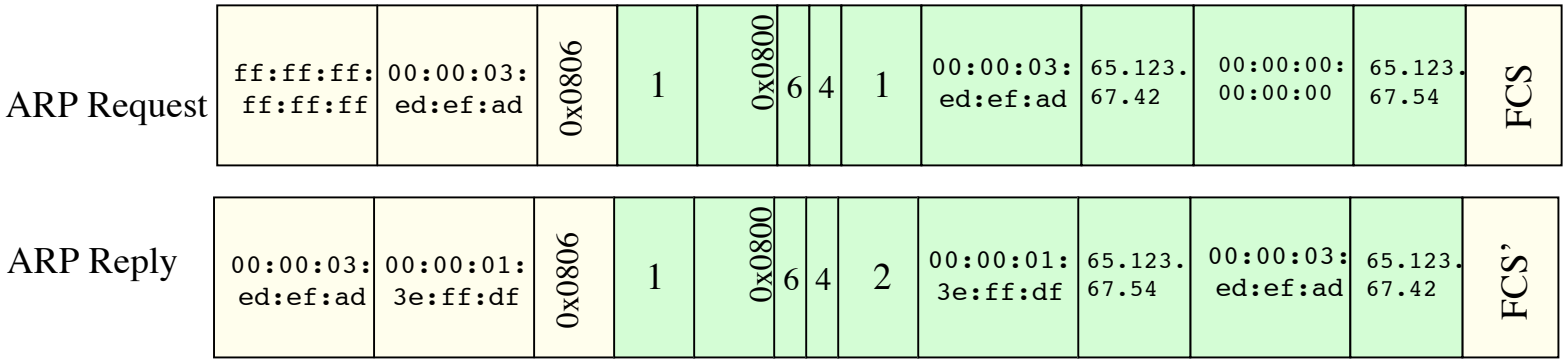
Formato de la PDU de ARP



- *hardware* = nivel de enlace, *protocol* = nivel de red
- *hard type* = tipo de dirección de enlace (1 = Ethernet)
- *prot type* = tipo de dirección de red (0x0800 = IP)
- *hard size* = tamaño en bytes de la dirección de enlace (Ethernet -> 6)
- *prot size* = tamaño en bytes de la dirección de red (IP -> 4)
- *op* = Tipo de operación:
  - 1 = ARP Request
  - 2 = ARP Reply
  - 3 = RARP Request, 4 = RARP Reply

Formato de la PDU de ARP

(Ejemplos)



- *hardware* = nivel de enlace, *protocol* = nivel de red
- *hard type* = tipo de dirección de enlace (1 = Ethernet)
- *prot type* = tipo de dirección de red (0x0800 = IP)
- *hard size* = tamaño en bytes de la dirección de enlace (Ethernet -> 6)
- *prot size* = tamaño en bytes de la dirección de red (IP -> 4)
- *op* = Tipo de operación:
  - 1 = ARP Request
  - 2 = ARP Reply
  - 3 = RARP Request, 4 = RARP Reply

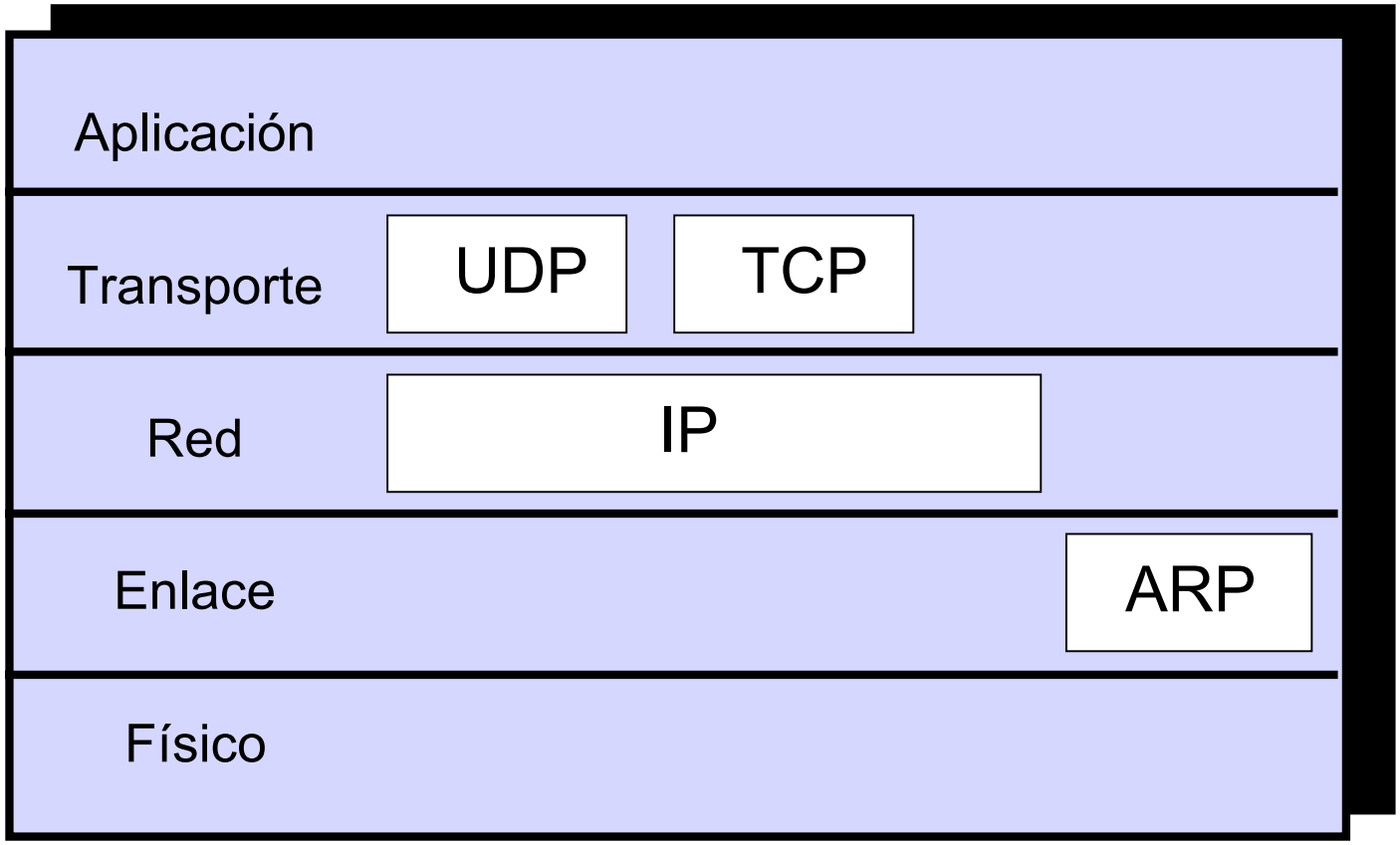
ARP: Address Resolution Protocol

- El host apunta en una **cache** la relación entre IP y MAC
- Para el próximo paquete no necesita hacer ARP
- El receptor del ARP Request aprende con esa trama la pareja (MAC, IP) del emisor
- Las entradas en la cache de ARP **caducan**
- Plug-and-play: no necesita intervención del administrador
- Funciona directamente **sobre el nivel de enlace** (Ethertype 0x0806)

Caché ARP del PC 1 (207.196.7.78)

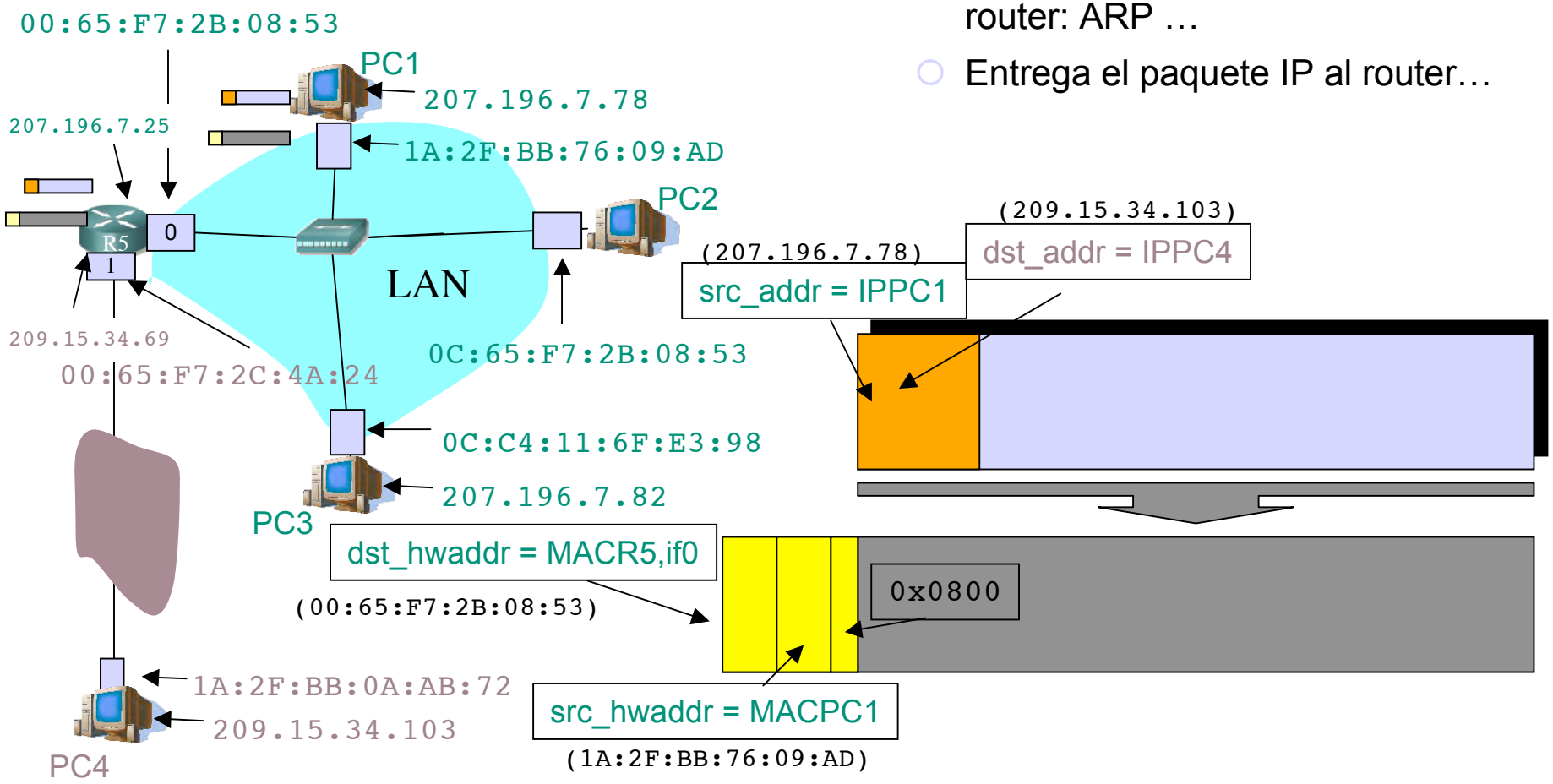
Dirección IP	Dirección MAC	Time
207.196.7.82	0C:C4:11:6F:E3:98	13:45

¿Dónde encaja ARP en la pila TCP/IP?



ARP: Address Resolution Protocol

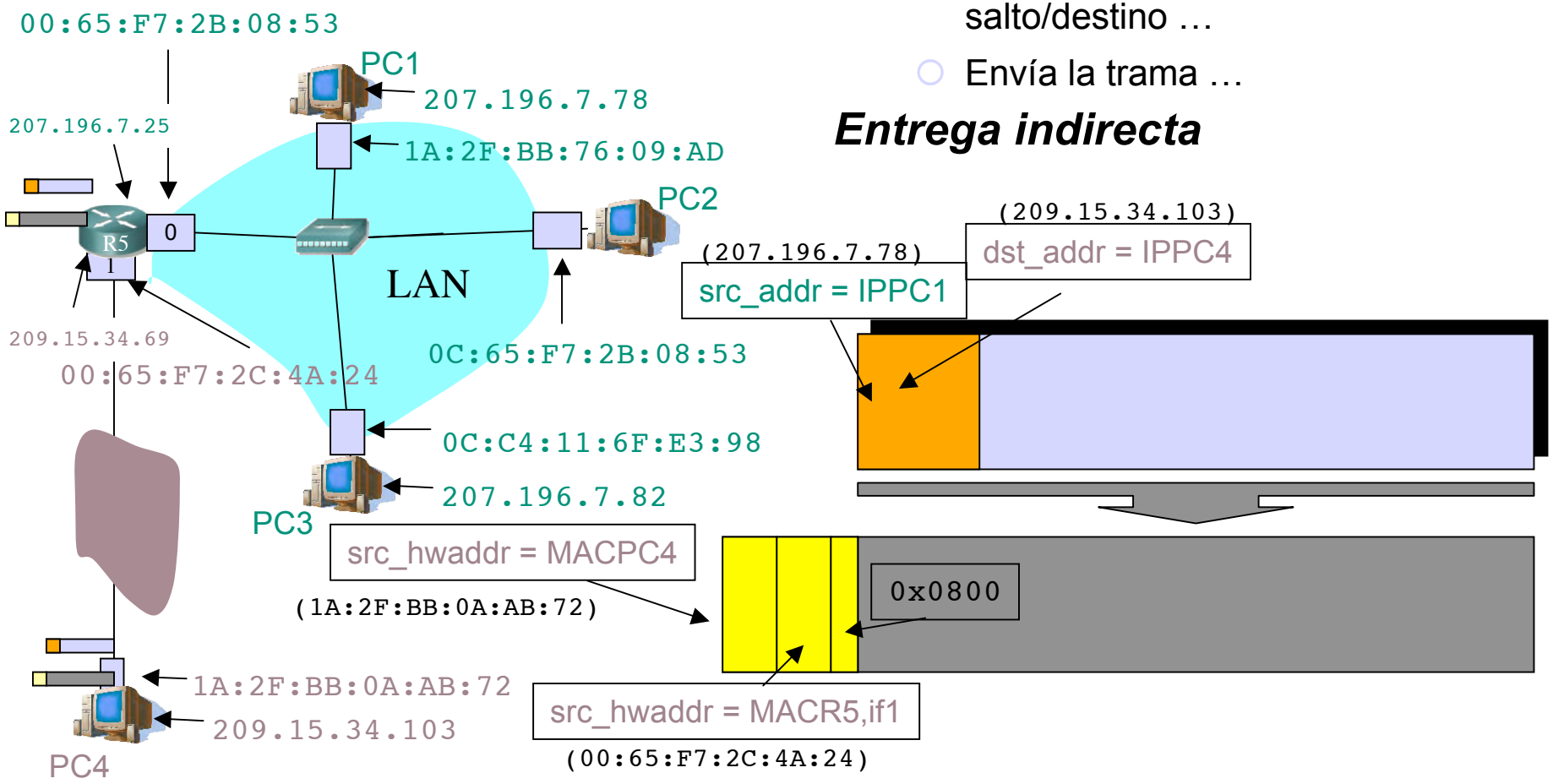
- **¿Y si el destino está en distinta red? …**



- Entregar el paquete a un router en su red:
  - Averiguar la MAC del interfaz del router: ARP …
  - Entrega el paquete IP al router…

ARP: Address Resolution Protocol

- **¿Y si el destino está en distinta red?**



- El router, como un host, repite el proceso:
  - Calcula la MAC del siguiente salto/destino …
  - Envía la trama …

Entrega indirecta

Contenido

- Internet Protocol
  - Características
  - IP en LAN Ethernet (ARP)
  - **Direccionamiento (subredes, Proxy-ARP, CIDR)**
  - Fragmentación y reensamblado
  - ICMP
- Enrutamiento en Internet
  - Distance-Vector
  - Link-State
  - Path-Vector
- UDP
- TCP
  - Características
  - Gestión de conexiones
  - Control de flujo
  - Retransmisiones
  - Control de congestión

Lecturas recomendadas

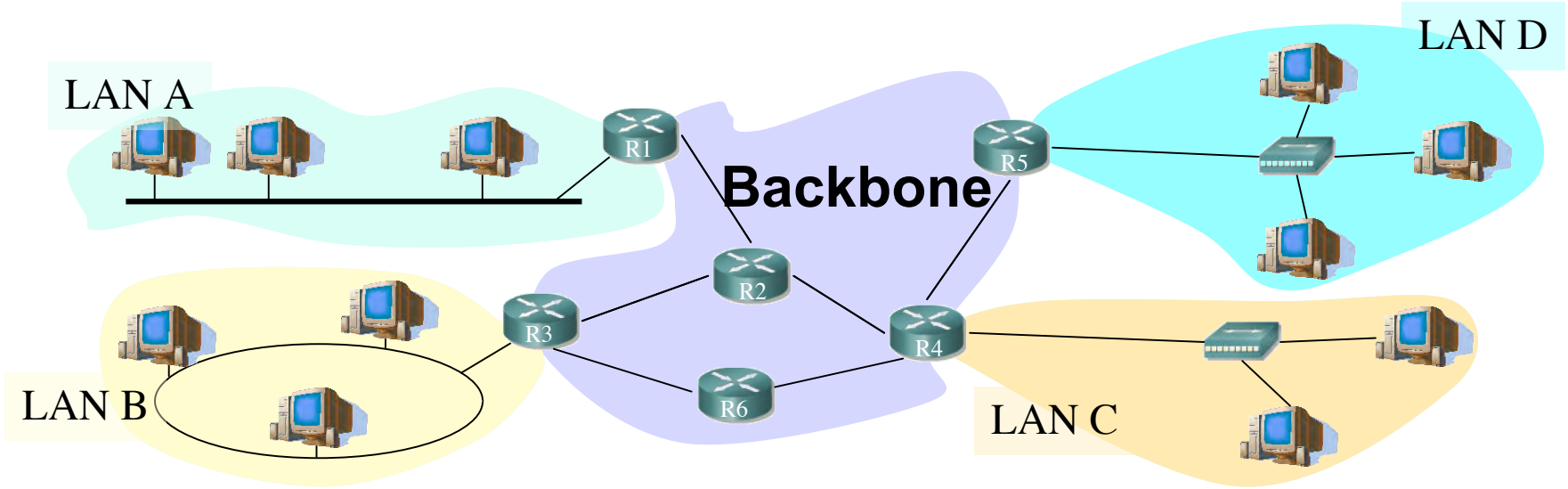
- [Forouzan] 4.2, 5.1
- [Stevens] 1.4, 3.4-3.7

Contenido

- Direccionamiento Classful
  - ¿Cómo es?
  - ¿Por qué así?
  - ¿Cómo funcionan los routers y los hosts?
  - Problemas
- Subredes
  - Proxy-ARP
  - Subnetting
    - ¿Cómo es?
    - ¿Cómo funcionan los routers y los hosts?
    - ¿Problemas?

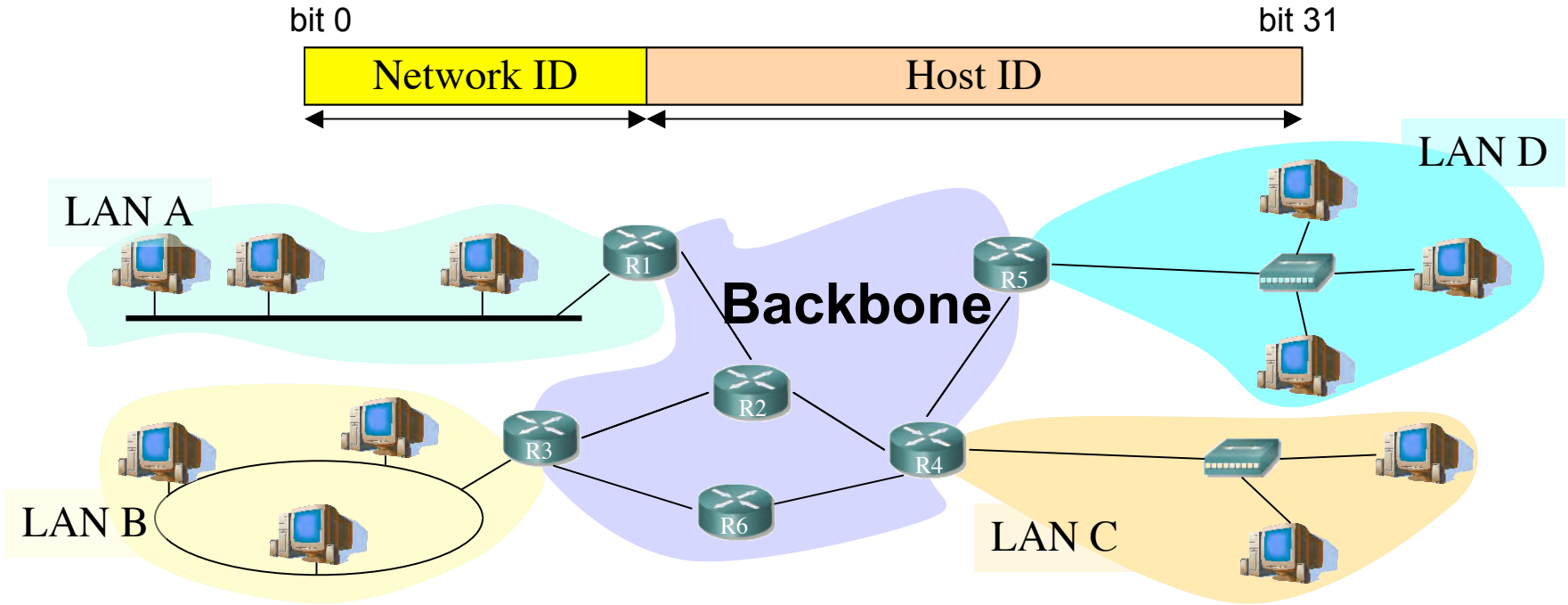
Direccionamiento Classful

- La “abuela” de Internet: ARPANET
- Cada red tiene un router de acceso que la conecta con el backbone de la red y así con las otras redes
- A cada red se le asigna un rango de direcciones IP
- ¿Red? Si origen y destino están en la misma, la tecnología se debe encargar de hacer llegar el paquete



Direccionamiento Classful

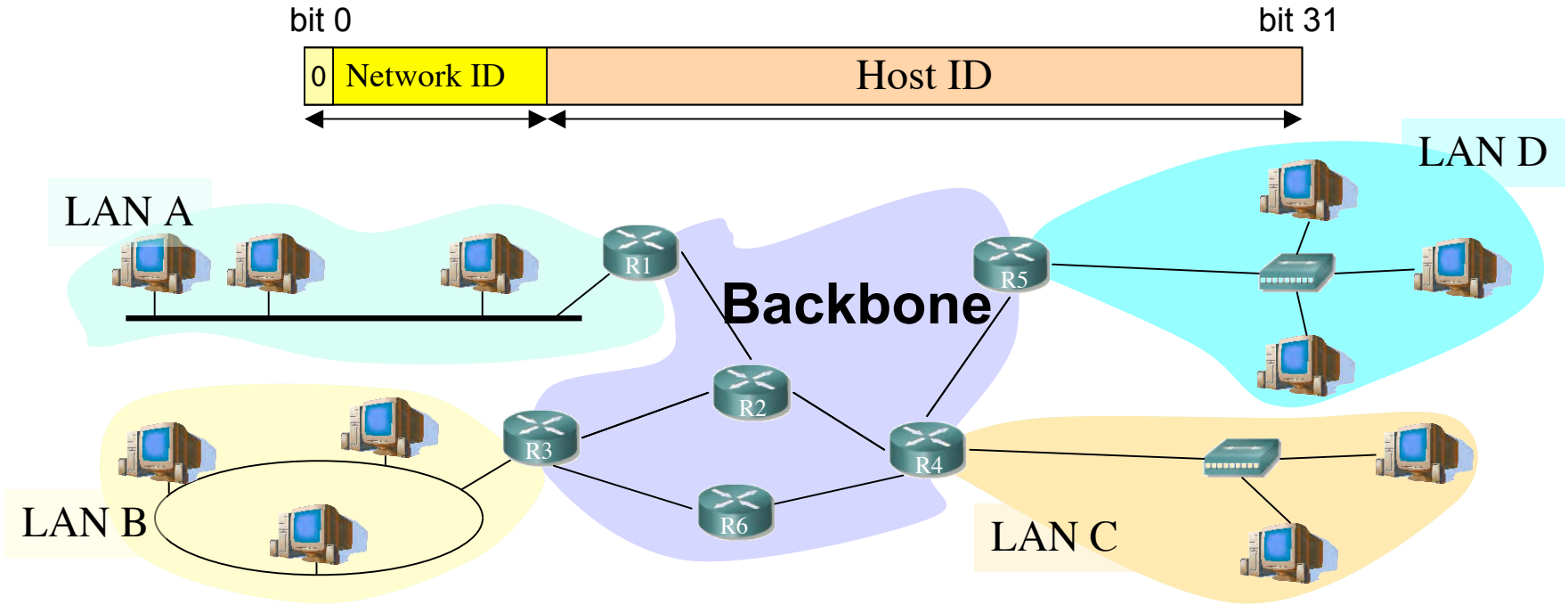
- Se pensó que podría haber redes de diferente tamaño (número de hosts)
- Se crearon 3 “tipos” de redes: clase A, clase B y clase C
- Las direcciones IP tendrán 2 partes:
  - Identificador de la red (network ID) …
  - Identificador del host (host ID) …



Clase A

- Network ID:
  - 8 bits, primero a 0 (…)
  - Primer byte: 0 - 127 (…)
  - 50% de las direcciones
- Host ID:
  - 24 bits (…)
  - Más de 16M direcciones!!

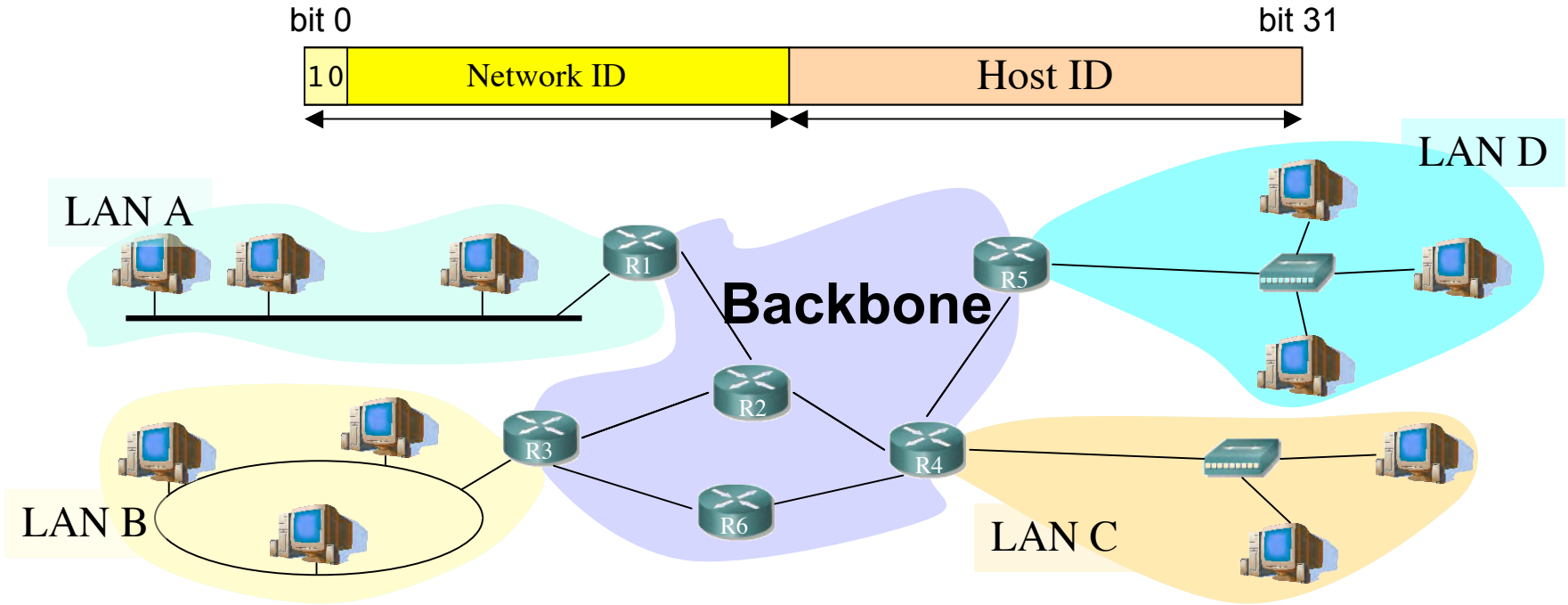
Redes “MUY” grandes



Clase B

- Network ID:
  - 16 bits, primeros a 10 (…)
  - Primer byte: 128 - 191 (…)
  - 16K redes
  - 25% de las direcciones
- Host ID:
  - 16 bits (…)
  - 64K direcciones

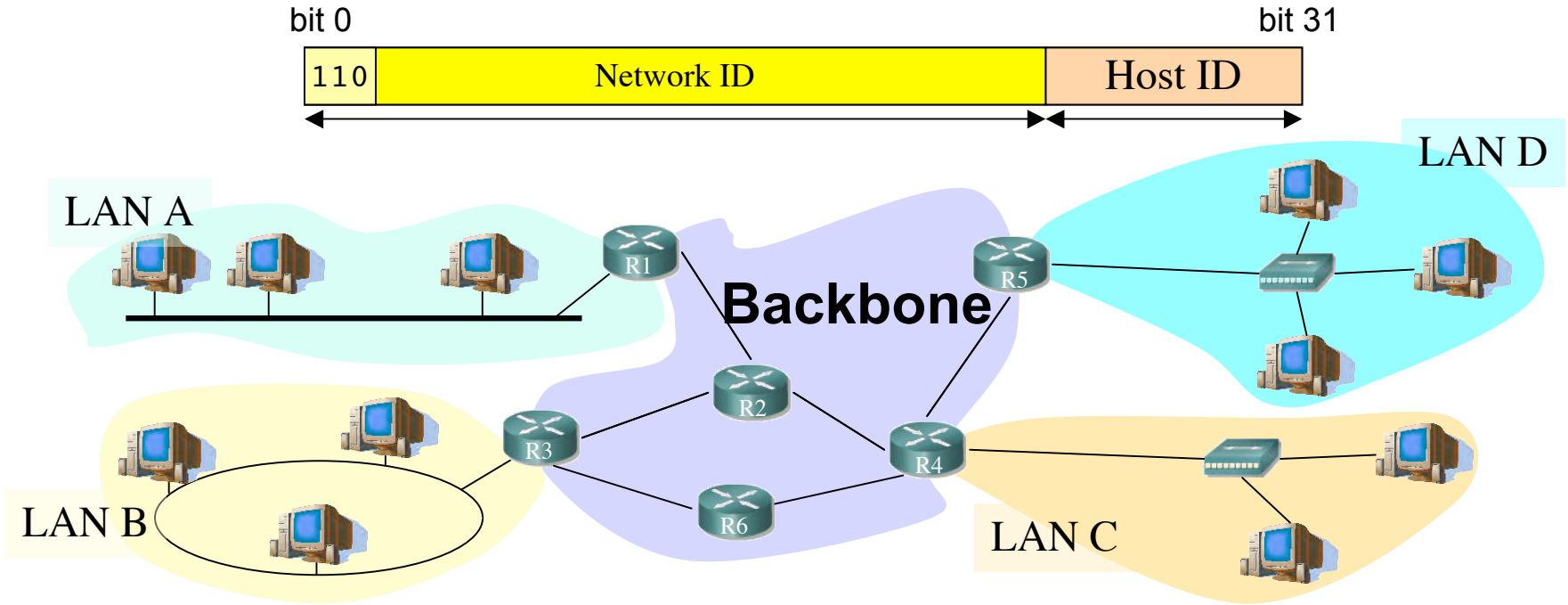
Redes grandes



Clase C

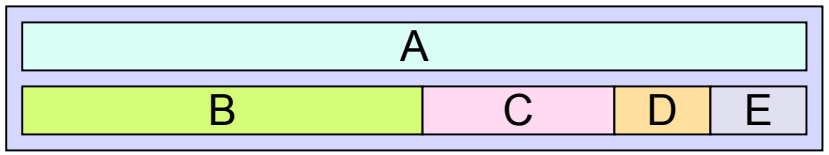
- Network ID:
  - 24 bits, primeros a 110 (…)
  - Primer byte: 192 - 223 (…)
  - 2M redes
  - 12.5% de las direcciones
- Host ID:
  - 8 bits (…)
  - 256 direcciones

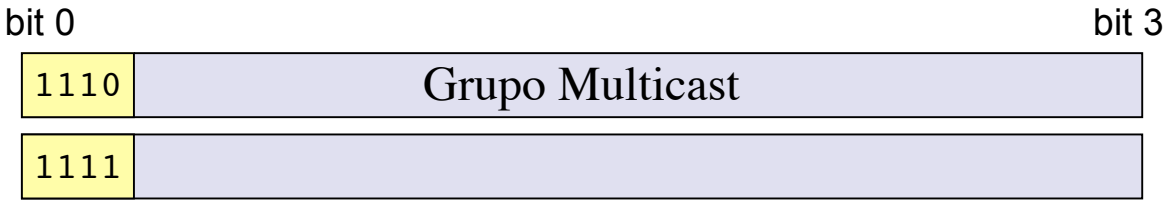
Redes pequeñas

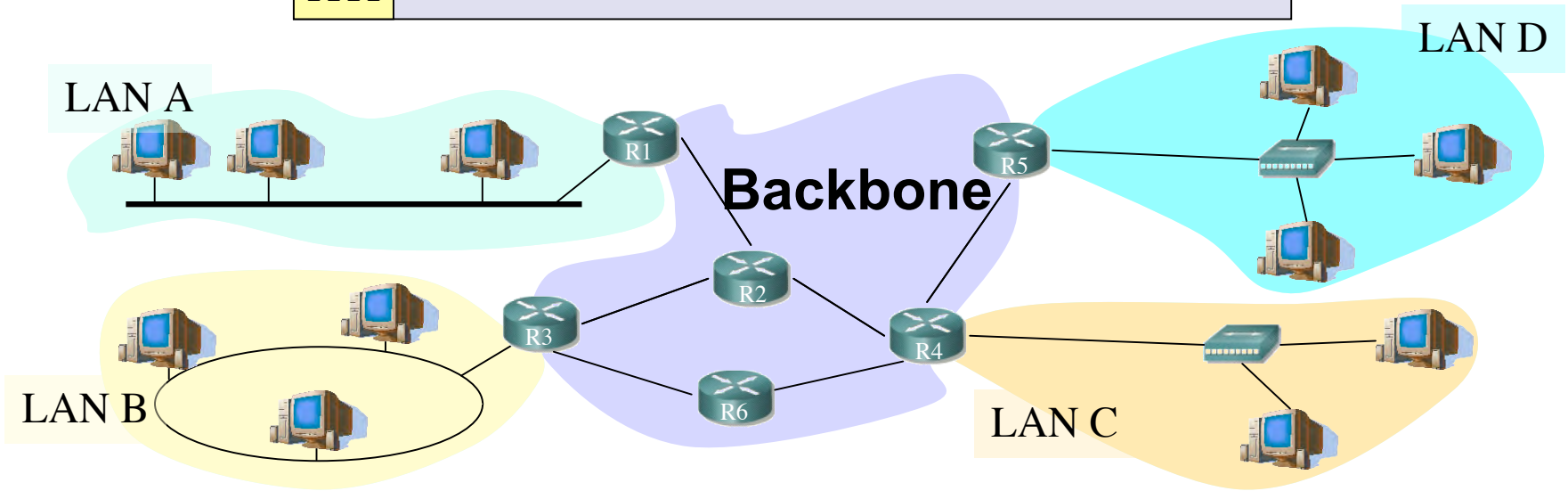


¿Y el resto de direcciones?

- Clase D:
  - Primeros bits a 1110
  - Primer byte: 224 - 239
  - Grupos multicast
- Clase E:
  - Reservadas para futuro uso
- Reparto en clases:

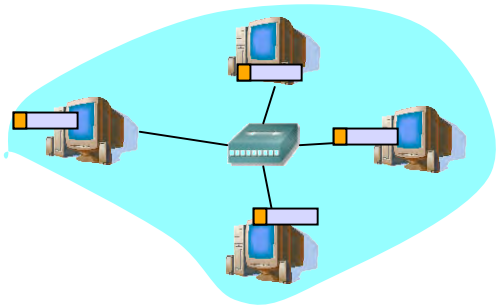




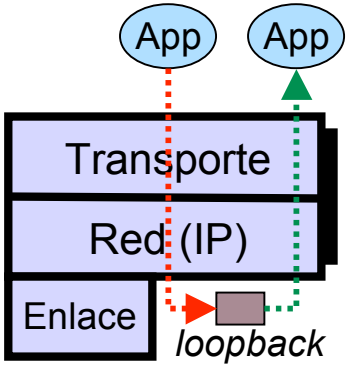


Direcciones especiales

- Dirección de red
  - Host ID = 0s
    - Ej: 130.206.0.0
- Dirección de broadcast de red (…)
  - Host ID = 1s
    - Ej: 130.206.255.255
- Broadcast limitado
  - 255.255.255.255



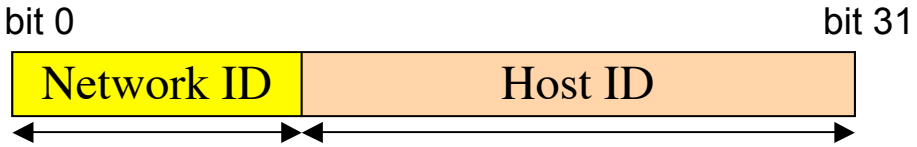
- Redes reservadas:
  - 0
  - 127 (loopback) (…)
  - 10 (privada)
  - 169.254 (no IP)
  - 172.16 a 172.31 (privada)
  - 192.0.2 (TEST-NET)
  - 192.168.0 a 192.168.255 (privada)
  - 192.18.0 a 192.19.255 (pruebas prestaciones)



Direccionamiento Classful

¿Por qué así?

- Routers emplean el Network ID para la decisión de reenvío

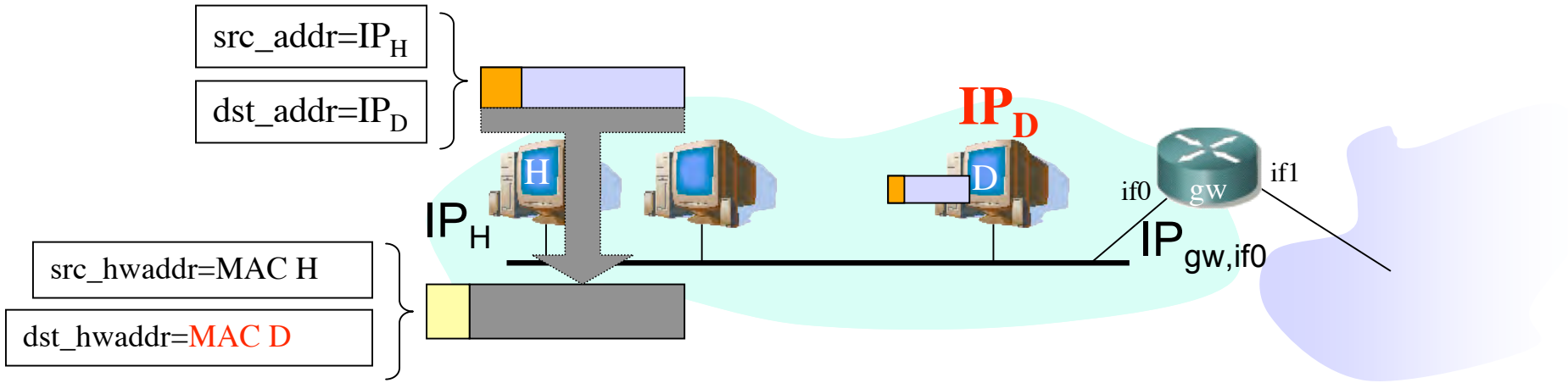


- Deben averiguar rápidamente cuál es el Network ID de la red a la que pertenece el destino (IP_d)
  - primer bit = 0:
    - $IP_d \in$ red de clase A
    - NetID = primeros 8 bits
  - (primer bit = 1)&(segundo bit = 0):
    - $IP_d \in$ red de clase B
    - NetID = primeros 16 bits
  - (primer bit = 1)&(segundo bit = 1)&(tercer bit=0):
    - $IP_d \in$ red de clase C
    - NetID = primeros 24 bits
- En la propia dirección IP está codificado el número de bits del NetID
- Son comprobaciones rápidas de realizar
- Cuanto menos tiempo emplee el router con cada paquete más paquetes podrá procesar por segundo

Direccionamiento Classful

Envío de paquetes desde los hosts

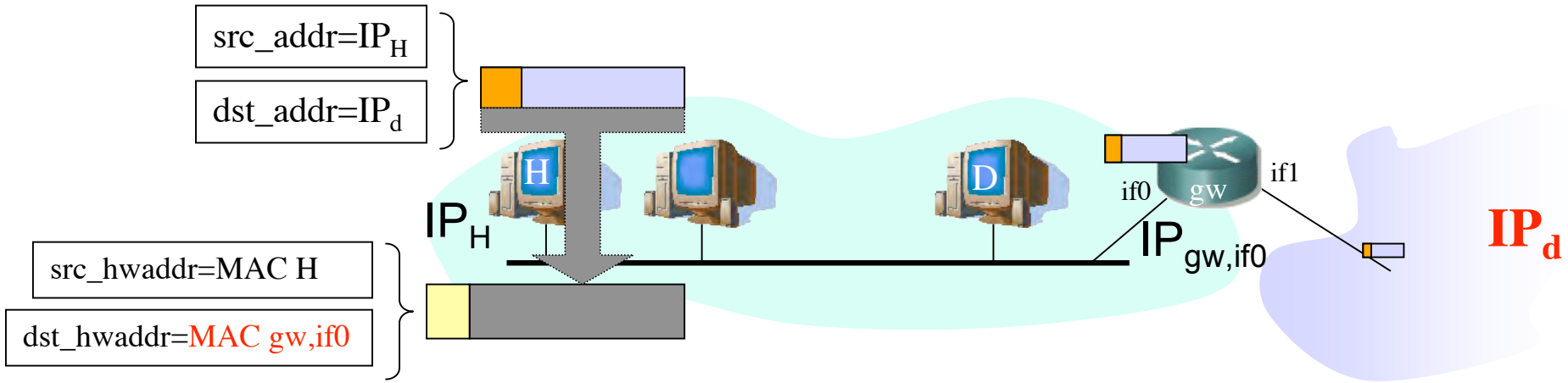
- Tienen configurado:
  - Su dirección IP (IP_H)
  - Dirección IP del router de salida de su LAN en el interfaz conectado a la misma (IP_{gw})
  - Pueden averiguar el NetID de su LAN a partir de su IP
- Dada la IP_D del destino al que desean enviar un paquete :
  - Calculan el NetID
  - ¿Es el mismo que el de mi red?
    - Sí: está en mi red, se lo envío directamente (a su MAC) … …
    - No: está en otra red, se lo envío al router (a la MAC del router) …



Direccionamiento Classful

Envío de paquetes desde los hosts

- Tienen configurado:
  - Su dirección IP (IP_H)
  - Dirección IP del router de salida de su LAN en el interfaz conectado a la misma (IP_{gw})
  - Pueden averiguar el NetID de su LAN a partir de su IP
- Dada la IP_D del destino al que desean enviar un paquete :
  - Calculan el NetID
  - ¿Es el mismo que el de mi red?
    - Sí: está en mi red, se lo envío directamente (a su MAC) … …
    - No: está en otra red, se lo envío al router (a la MAC del router) … …

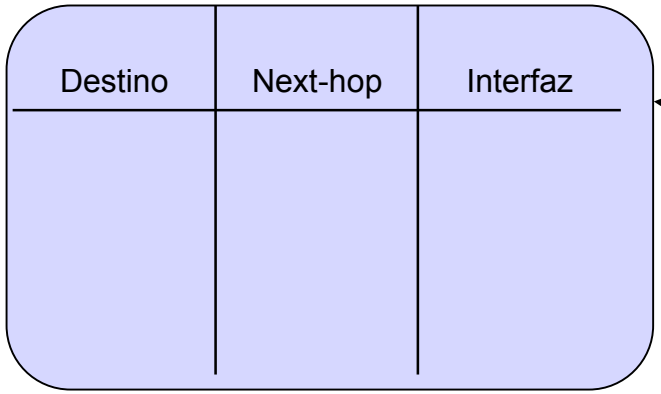


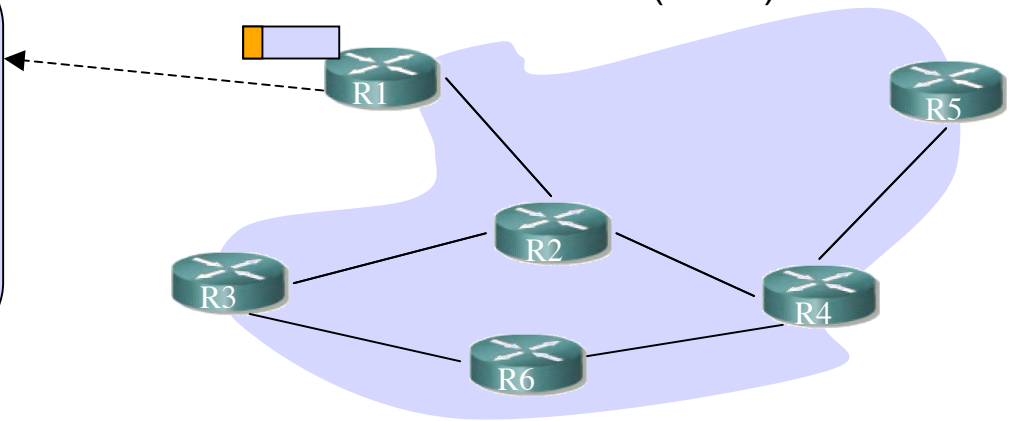
Direccionamiento Classful

Reenvío de paquetes en los routers

- Sin estado. Decisiones paquete a paquete.
- Tienen configurado:
  - IP de cada uno de sus interfaces
  - Tabla de rutas
- Dada IP_D que no es ninguna de sus direcciones IP:
  - Busca en la tabla fila t.q. "Destino" = IP_D
    - Sí: Es una **ruta a ese host**, lo envía según indica la fila
    - No: Calcula el NetID.

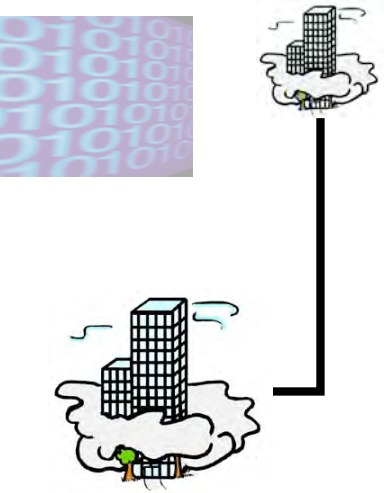
    Busca una ruta a esa red
      - Sí: Es una **ruta a esa red**, lo envía según indica la fila
      - No: Busca en la tabla una **ruta por defecto**. ¿Encuentra una?
        - Sí: Lo envía según indica la fila
        - No: No sabe cómo hacer llegar el paquete al destino. Lo descarta (*lo tira*)

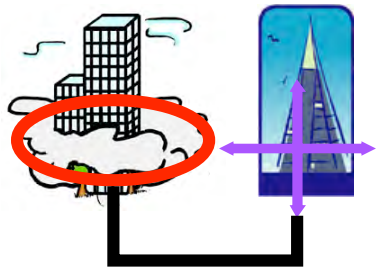


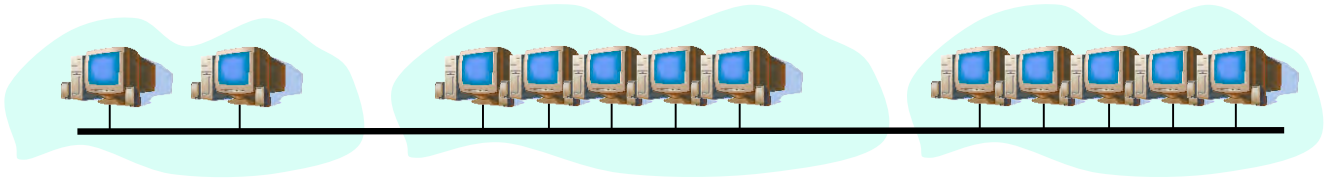


Problemas del esquema Classful

- Las redes pueden llegar a ser muy grandes
- Clase A:
  - Direcciones para millones de hosts
  - Difícil que una tecnología de LAN soporte esa cifra de máquinas conectadas
- Situaciones en que hace falta “partir” la red:
  - LANs en edificios distantes (enlaces punto-a-punto)..
  - LANs de diferentes tecnologías …
  - Exceder límites tecnológicos (número de hosts, distancias, etc)
  - Congestión por comunicación entre ciertos pares de hosts …
  - Excesivo tráfico de broadcast a nivel de enlace



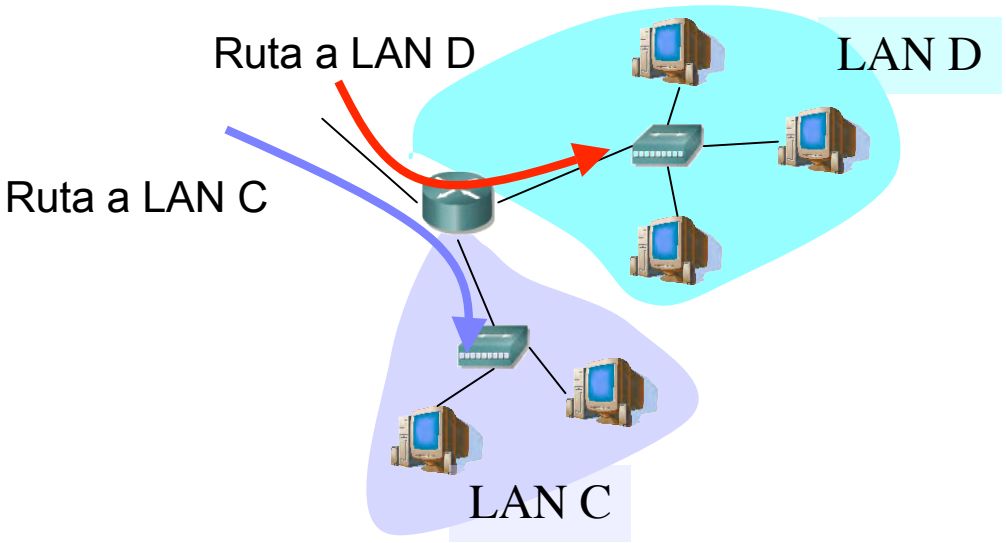




Una organización con más de una LAN

Un *NetworkID* para cada una

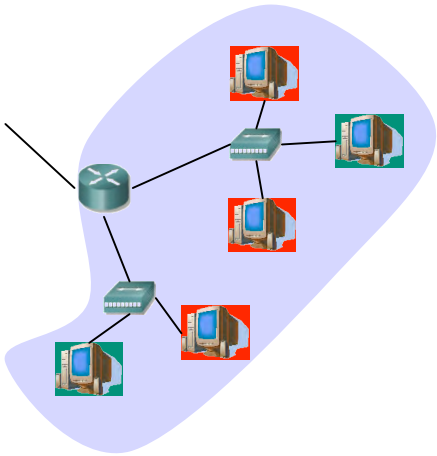
- Pro:
  - No requiere modificaciones
- Cons:
  - Crecen las tablas de rutas
  - Se propaga al exterior información interna

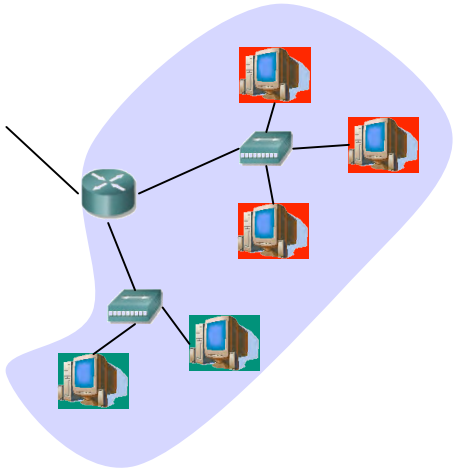


Una organización con más de una LAN

Un solo *NetworkID* y

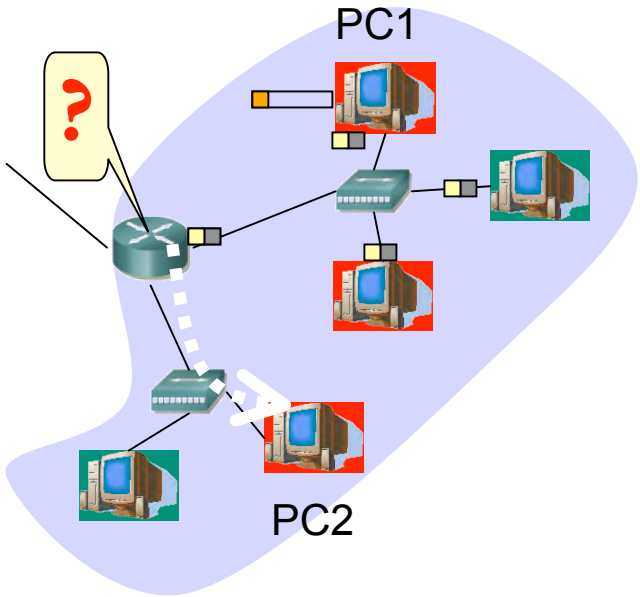
- asignar direcciones a los hosts sin tener en cuenta las diferentes LANs (*"transparent subnets"*)
  - Proxy ARP
- particionar el espacio de direcciones para las diferentes LANs (*"explicit subnets"*)
  - Modificar implementación de IP





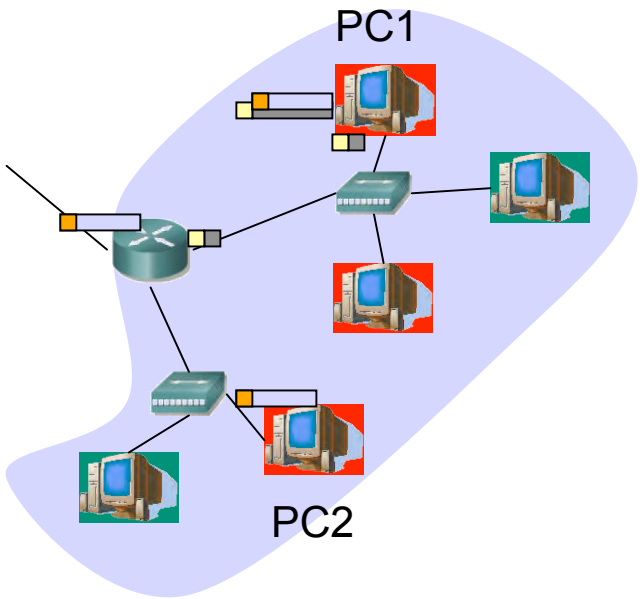
Proxy ARP

- También llamado el *"ARP Hack"*
- PC1 desea enviar un paquete IP a PC2
- Para PC1 ambos están en la misma LAN
- Manda un *ARP Request… …*
- Router sabe que PC2 está en otro segmento …
- Router *responde al ARP* con su MAC…
- PC1 envía la trama al router pensando que es PC2 …
- El router reenvía el paquete IP …



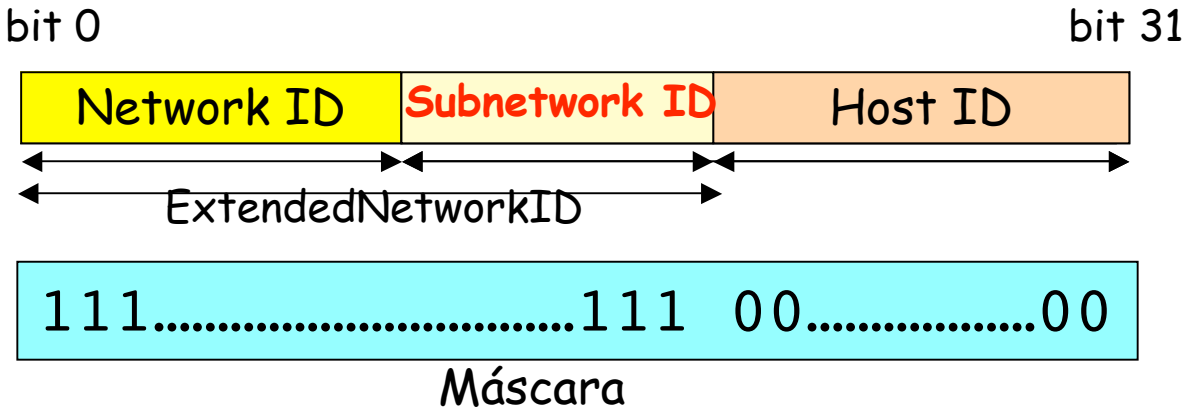
Proxy ARP

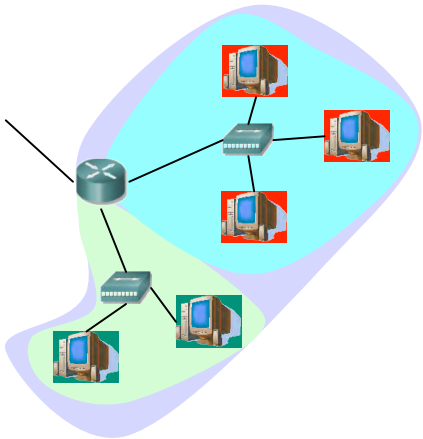
- También llamado el *"ARP Hack"*
- PC1 desea enviar un paquete IP a PC2
- Para PC1 ambos están en la misma LAN
- Manda un *ARP Request… …*
- Router sabe que PC2 está en otro segmento …
- Router *responde al ARP* con su MAC…
- PC1 envía la trama al router pensando que es PC2 …
- El router reenvía el paquete IP …



Subnetting

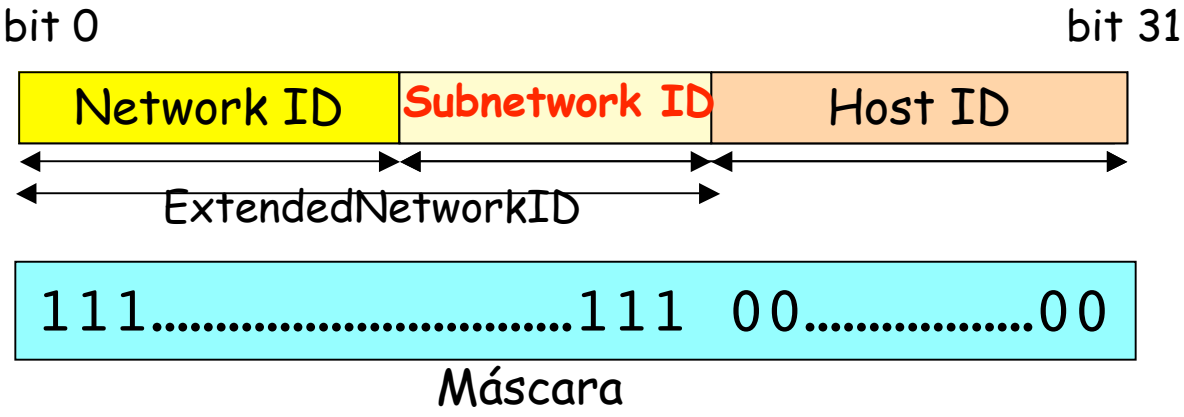
- También llamado FLSM (Fixed Length Subnet Masks)
- Parte del *Host ID* se emplea para diferenciar la *subred*…
- *NetworkID*+*SubnetworkID* = ***ExtendedNetworkID*** …
- Determinado por la *máscara de subred …*
- Se empleó en redes Clase B
  - Muy pocas redes Clase A
  - Clase C muy pequeñas

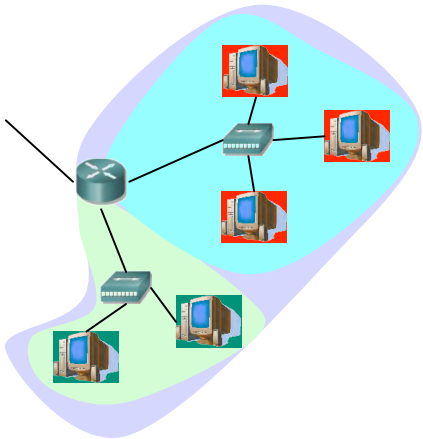




Subnetting

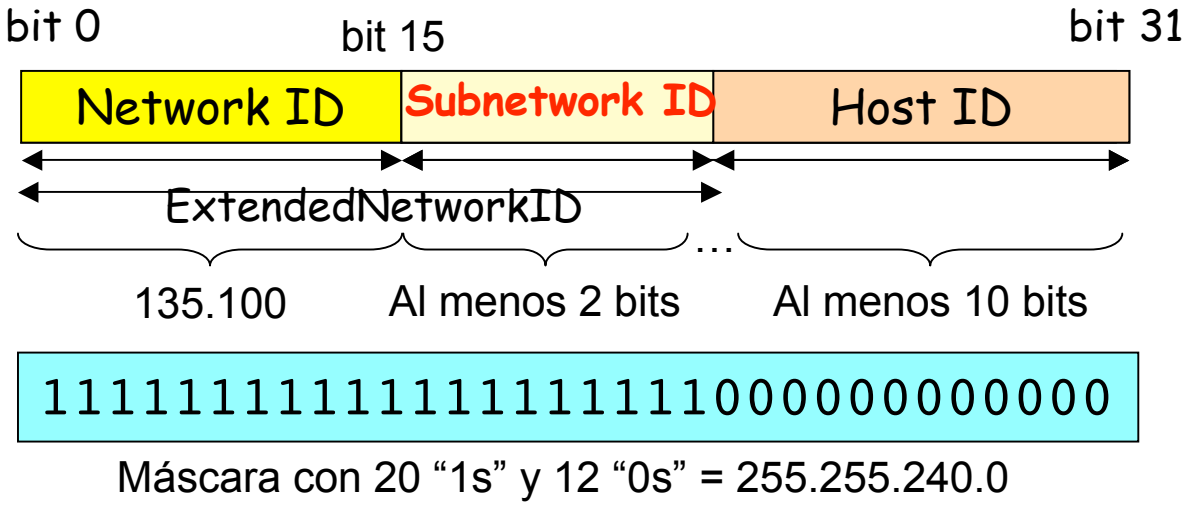
- Algunas restricciones:
  - SubnetworkID ≠ 0s (“this” network)
  - SubnetworkID ≠ 1s (“all” subnetworks)
  - ⇒ Al menos 2 bits
  - Misma máscara en todas las subredes de la misma red (FLSM)
- En cada subred:
  - Dirección de la subred (HostID=0s)
  - Dirección de *broadcast* de la subred (HostID=1s)

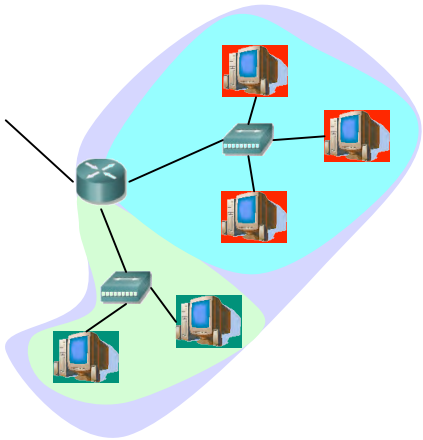




Subnetting: Ejemplo

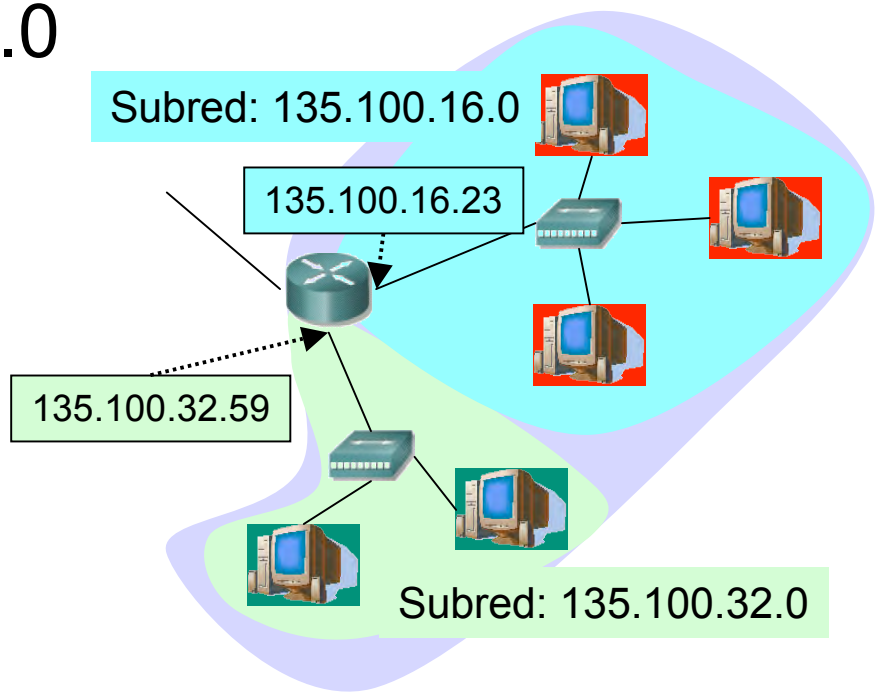
- LAN: 135.100.0.0 (clase ?)
- Queremos al menos poder conectar 1000 máquinas en cada subred
- 2 subredes ⇒ mínimo número de bits? …
- 1000 máquinas ⇒ mínimo número de bits? …
- ¿Y si hay bits “sobrantes”?…
- Por ejemplo 4 bits para el Subnetwork ID …





Subnetting: Ejemplo

- Direcciones de subred:
  - 10000111011001000001 0000000000
    - Hosts: 135.100.16.1 a 135.100.31.254
  - 10000111011001000010 0000000000
    - Hosts: 135.100.32.1 a 135.100.47.254
- Máscara: 255.255.240.0

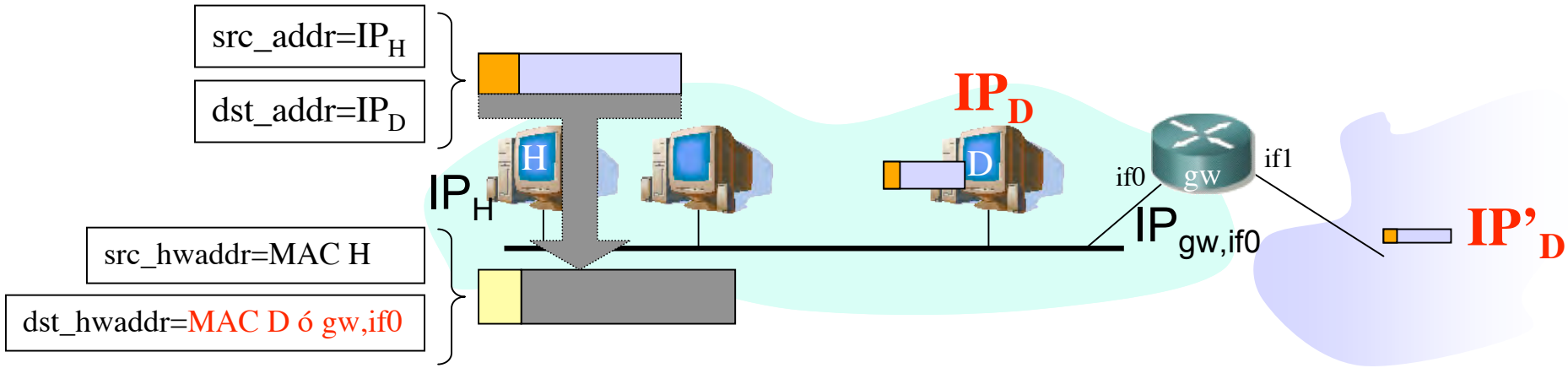


Subnetting
Envío de paquetes desde los hosts

- Tienen configurado:
  - Su dirección IP (IP_H)
  - La máscara de subred
  - Dirección IP del router de salida de su LAN en el interfaz conectado a la misma (IP_{gw})
  - Pueden averiguar el Extended Network ID de su LAN a partir de su IP:

```
135.100.35.67 = 10000111011001000010001101000011
255.255.240.0 = 11111111111111111111000000000000 (AND)
ExtendedNetID = 10000111011001000010000000000000 = 135.100.32.0
```

- Dada la IP_D del destino al que desean enviar un paquete :
  - Aplica (AND) la máscara de subred
  - ¿El resultado es el ExtendedNetworkID de mi subred?
    - Sí: se lo envío directamente (a su MAC)
    - No: está en otra subred o en otra red,
      se lo envío al router (a la MAC del router)



Subnetting

Reenvío de paquetes en los routers

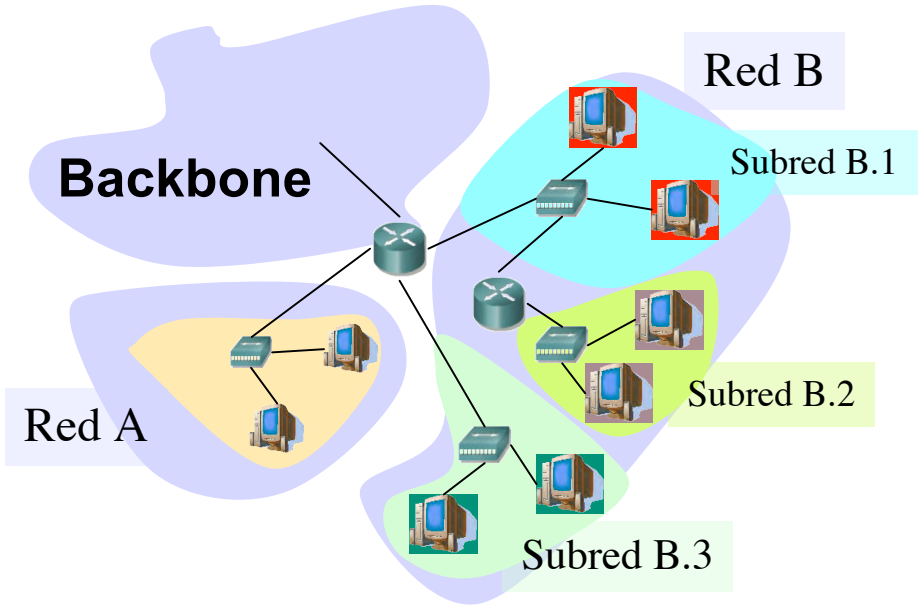
- Tienen configurado:
  - IP en cada uno de sus interfaces
  - Máscara en cada uno
  - Tabla de rutas
- IP_D que no es ninguna de sus direcciones IP
- Calcula el NetworkID de la red a la que pertenece (classful)
- ¿Tiene un interfaz en esa red?
  - No: Red destino identificada
  - Sí: Toma la máscara del interfaz que tiene en esa red

    Calcula el ExtendedNetworkID

- ¿Encuentra ese identificador de red/subred en su tabla de rutas?
  - Sí: lo envía según indica la ruta
  - No: Busca en la tabla una ruta por defecto

    ¿Encuentra una?
    - Sí: Lo envía según indica la ruta
    - No: Descarta el paquete

Destino	Next-hop	Interfaz

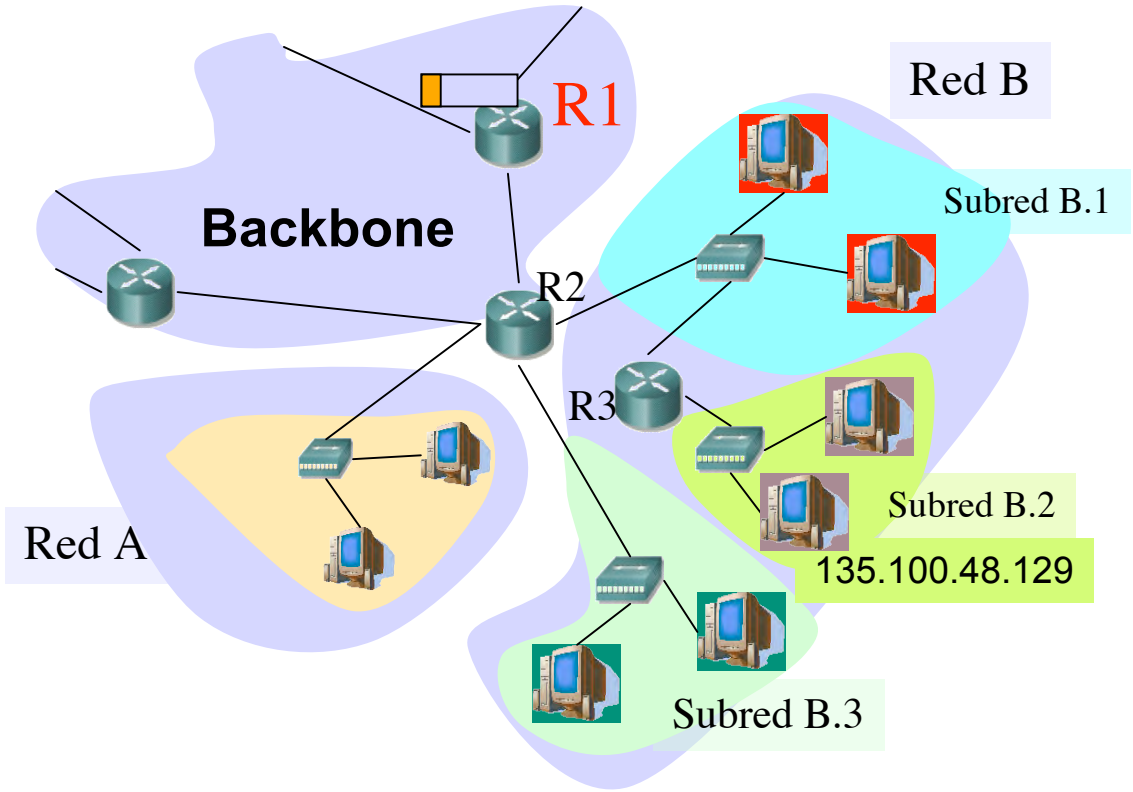


Subnetting

Ejemplo: IP_d=135.100.48.129

Destino	Next-hop	if
135.100.0.0 *(B)*	10.50.43.12 *(R2)*	1
45.0.0.0	(otro)	0
64.0.0.0	(otro)	0
130.206.0.0	(otro)	2
…	…	…

R1



Subnetting

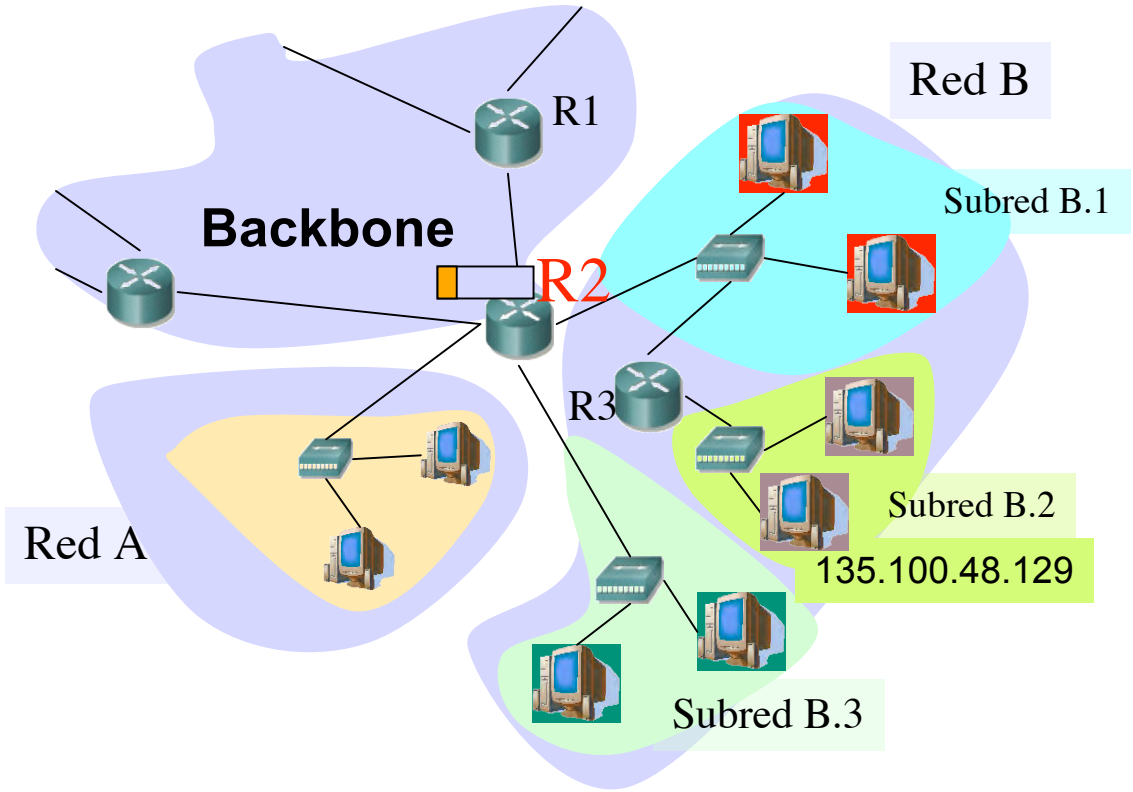
Ejemplo: IP_d=135.100.48.129

R1

Destino	Next-hop	if
135.100.0.0 *(B)*	10.50.43.12 *(R2)*	1
45.0.0.0	(otro)	0
64.0.0.0	(otro)	0
130.206.0.0	(otro)	2
…	…	…

R2

Destino	Next-hop	if
135.100.16.0 (B.1)	-	1
135.100.32.0 (B.2)	135.100.16.1 (R3)	1
135.100.48.0 (B.3)	-	2
180.40.0.0 (A)	-	3
…	…	…



Subnetting

Ejemplo: IP_d=135.100.48.129

R1

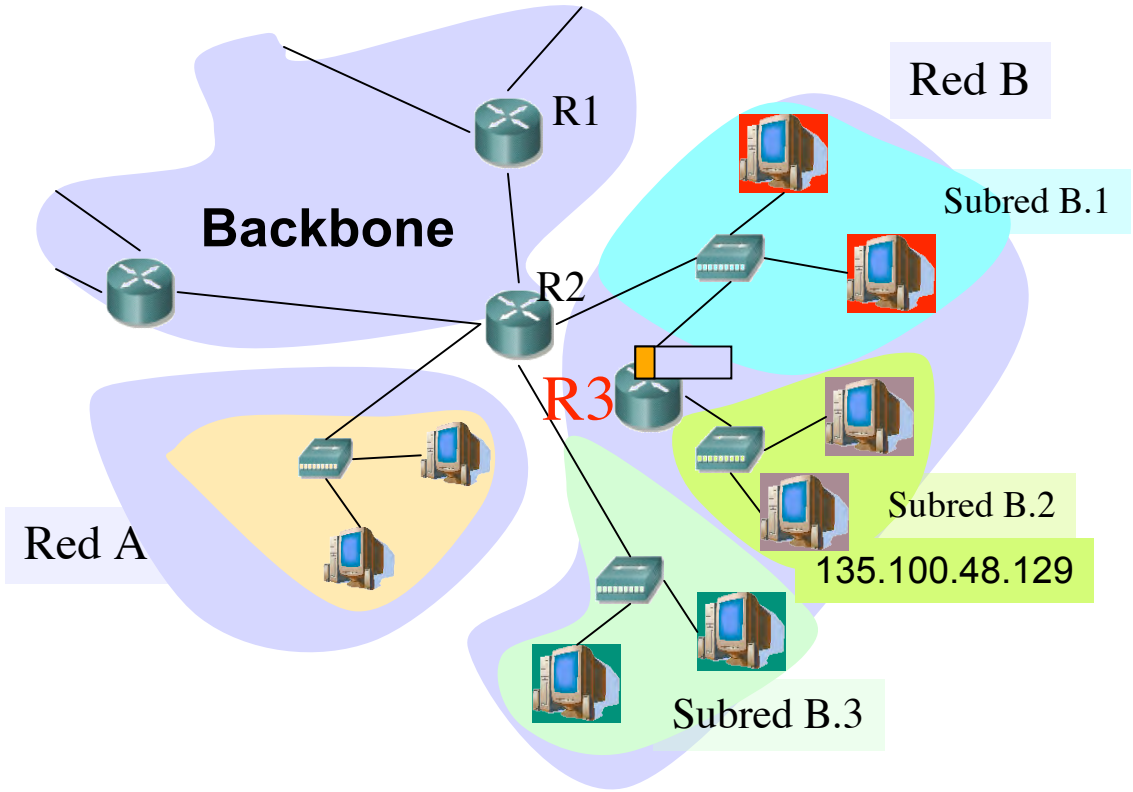
Destino	Next-hop	if
135.100.0.0 *(B)*	10.50.43.12 *(R2)*	1
45.0.0.0	(otro)	0
64.0.0.0	(otro)	0
130.206.0.0	(otro)	2
…	…	…

R2

Destino	Next-hop	if
135.100.16.0 (B.1)	-	1
135.100.32.0 (B.2)	135.100.16.1 (R3)	1
135.100.48.0 (B.3)	-	2
180.40.0.0 (A)	-	3
…	…	…

R3

Destino	Next-hop	if
135.100.16.0 (B.1)	-	0
135.100.32.0 (B.2)	-	1
135.100.48.0 (B.3)	135.100.16.2 (R2)	0
default	135.100.16.2 (R2)	0



Lecturas recomendadas

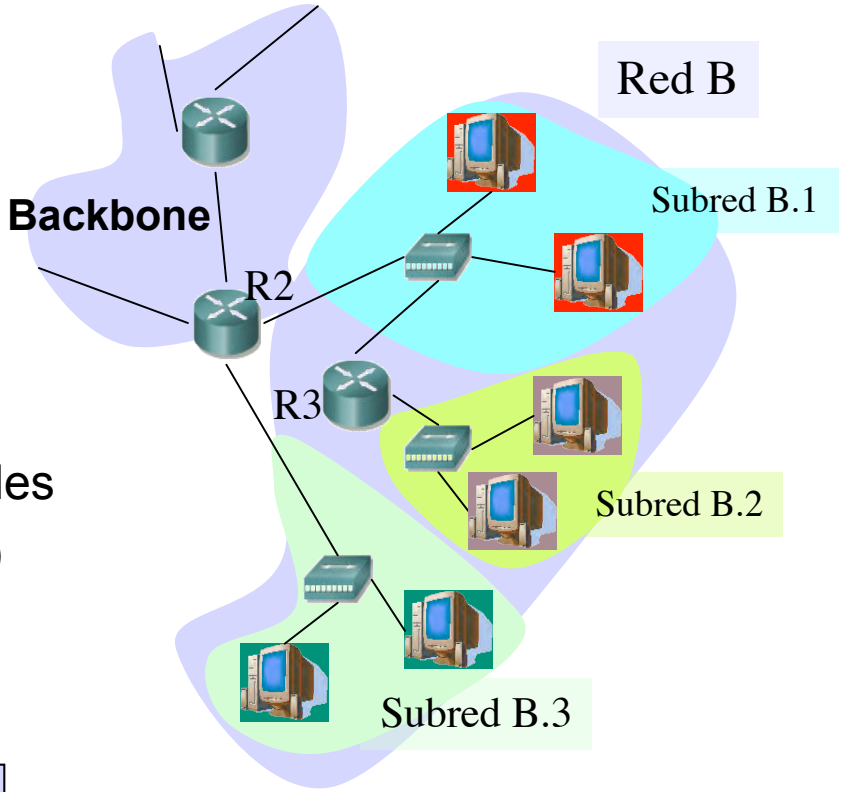
- [Forouzan03] 5.1-5.3
- RFC 1519

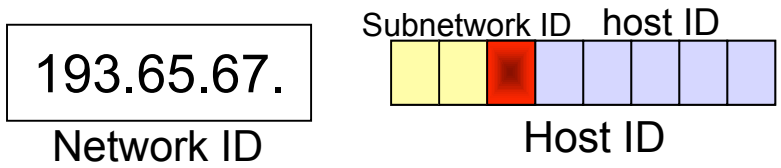
Contenido

- Evolución de los esquemas de direccionamiento
  - VLSM
  - Supernetting
  - CIDR

Problemas con Subnetting

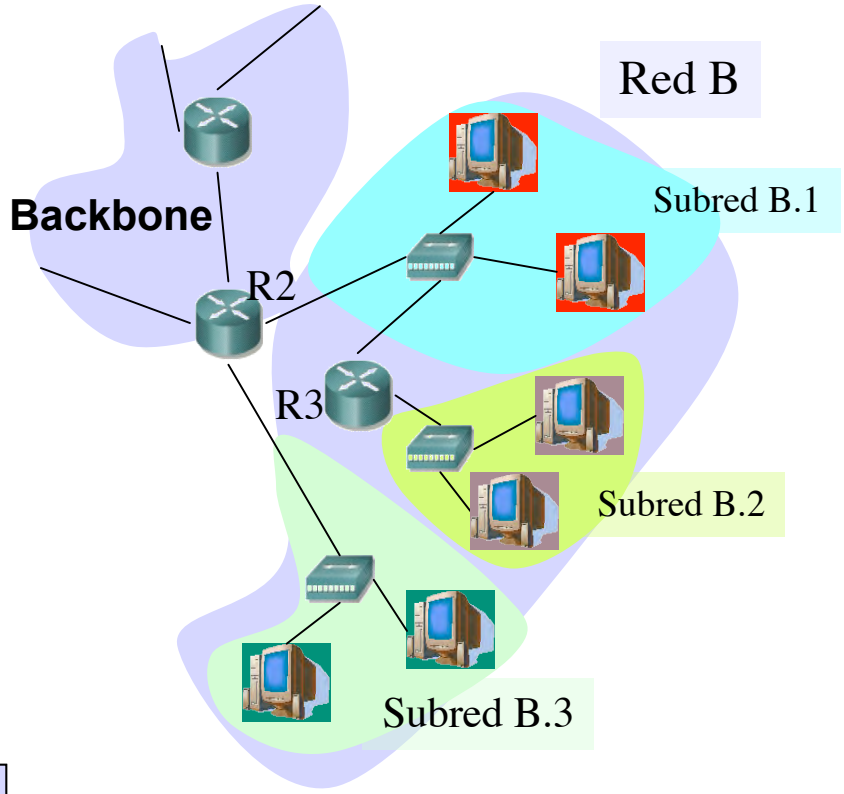
- Todas las subredes deben emplear la misma máscara
- Subredes de tamaño heterogénero ⇒ desaprovechar direcciones
- Ejemplo:
  - Red 193.65.67.0
  - Se crean 3 subredes
  - B.1: Al menos 50 hosts
  - B.2: Al menos 20 hosts
  - B.3: Al menos 20 hosts
  - Total: 90 hosts
  - Clase C ⇒ 256 direcciones disponibles
  - 3 subredes ⇒ SubNetID > 2 bits (…)
  - B.1 50 hosts ⇒ HostID > 5 bits (…)

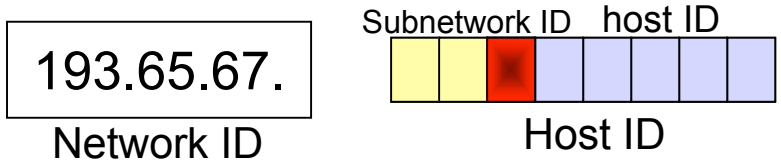




Problemas con Subnetting

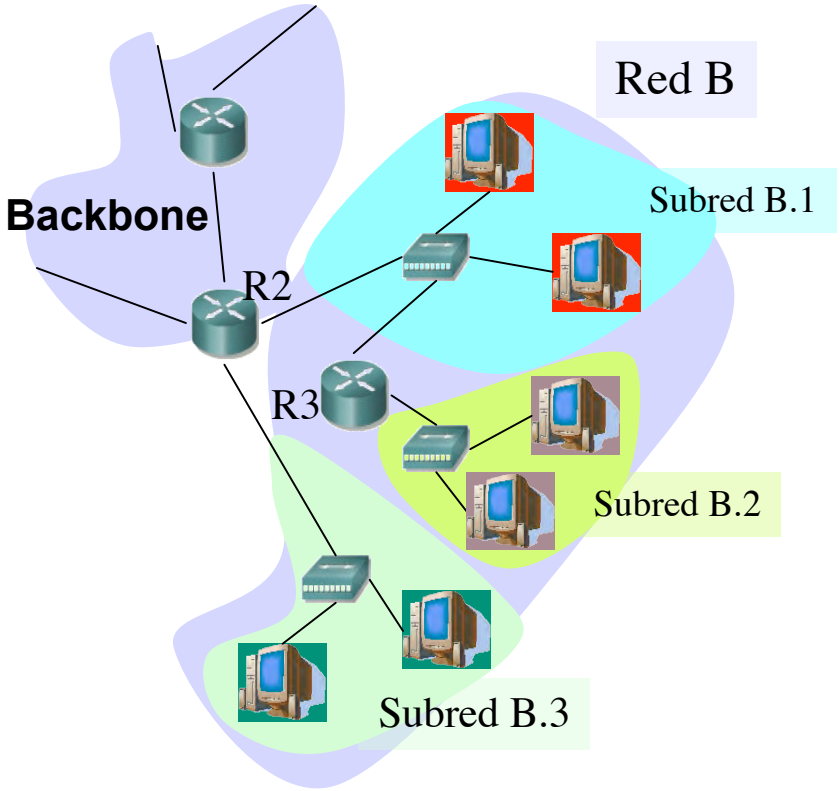
- ¿Dónde se han perdido las direcciones?
- Las 3 subredes dimensionadas con el tamaño de la mayor (máscara fija)
- No se usan dos subredes
- ¡Esas dos son del mismo tamaño que la mayor!





VLSM

- Subnetting = FLSM (Fixed Length Subnet Masks)
- FLSM es “one-size-fits-all”
- ¿Cómo ajustar mejor el tamaño de cada subred?
  - VLSM = Variable Length Subnet Masks
- Ejemplo:
  - B.1 50 hosts ⇒ HostID = 6 bits
    193.65.67. [**00 XXXXXX**]
  - B.2 20 hosts ⇒ HostID = 5 bits
    193.65.67. [**01 0 XXXXX**]
  - B.3 20 hosts ⇒ HostID = 5 bits
    193.65.67. [**01 1 XXXXX**]
  - Quedan sin asignar:
    - 193.65.67. [**1X XXXXXX**]



VLSM (Ejemplo)

- B.1 50 hosts ⇒ HostID = 6 bits

  193.65.67. [**00 XXXXXX**]

  Dir. Red = 193.65.67.0

  Máscara = 255.255.255.192

- B.2 20 hosts ⇒ HostID = 5 bits

  193.65.67. [**01 0 XXXXX**]

  Dir. Red = 193.65.67.64

  Máscara = 255.255.255.224

- B.3 20 hosts ⇒ HostID = 5 bits

  193.65.67. [**01 1 XXXXX**]

  Dir. Red = 193.65.67.96

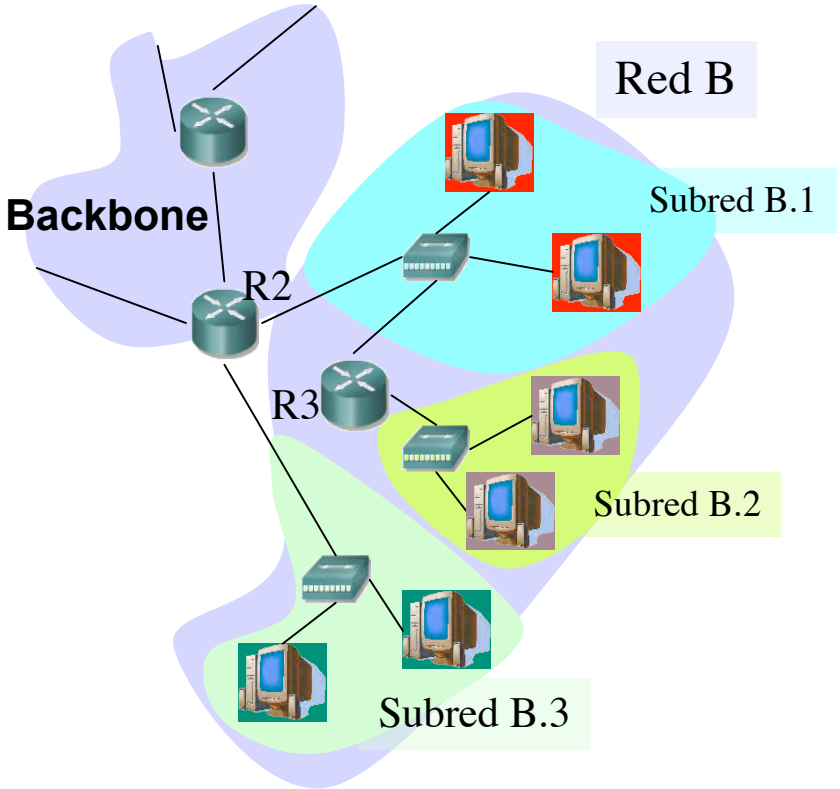
  Máscara = 255.255.255.224

- Quedan sin asignar:

  193.65.67. [**1X XXXXXX**]

  Dir. Red = 193.65.67.128

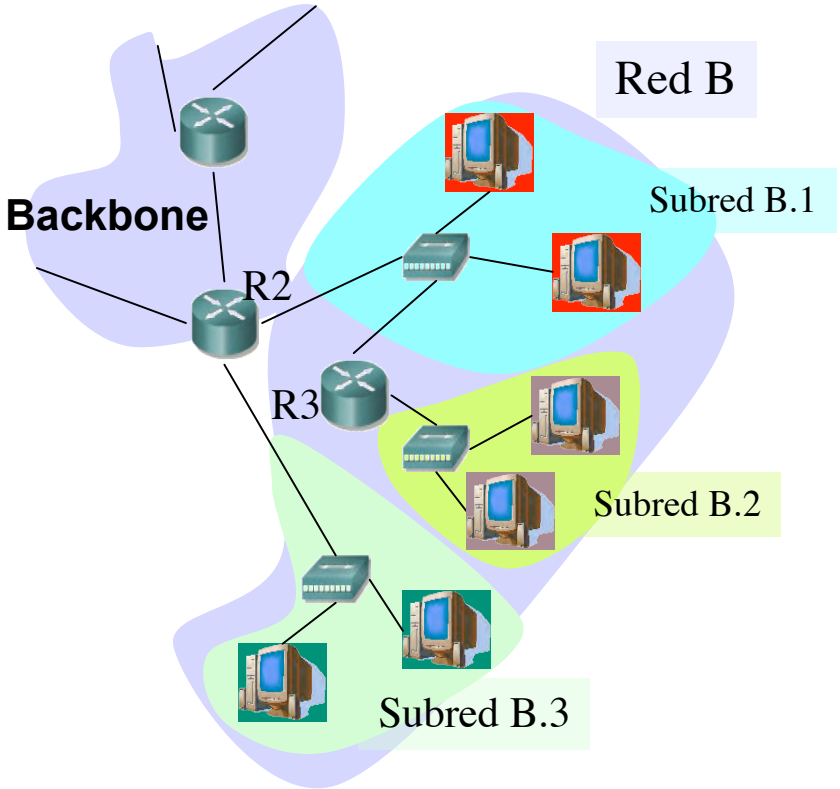
  Máscara = 255.255.255.128



VLSM

- Cada subred puede tener una máscara diferente
- Las rutas en la tabla de rutas deben incluir la máscara

Destino	Máscara	Next-hop	Interfaz



Supernetting

El problema

- Clases A y B casi agotadas
- Muchas redes clase C pero pequeñas (256 direcciones)
- Ejemplo:
  - Red para 1000 hosts
  - Clase C: insuficiente
  - Clase B: ¡ desperdicia más de 60.000 direcciones (98%) !
- Solución: Asignar varias redes de Clase C
- Una ruta para cada Clase C: Explosión de rutas
- ¿ Cómo evitarlo ?

Supernetting

¿Cómo?

- Asignar las redes formando un bloque
- Redes consecutivas
- Sin “huecos”
- Ejemplo
  - 1000 hosts ⇒ 4 redes clase C ⇒ 4 rutas (…)

    200.45.64.0 = 11001000 00101101 01000000 00000000

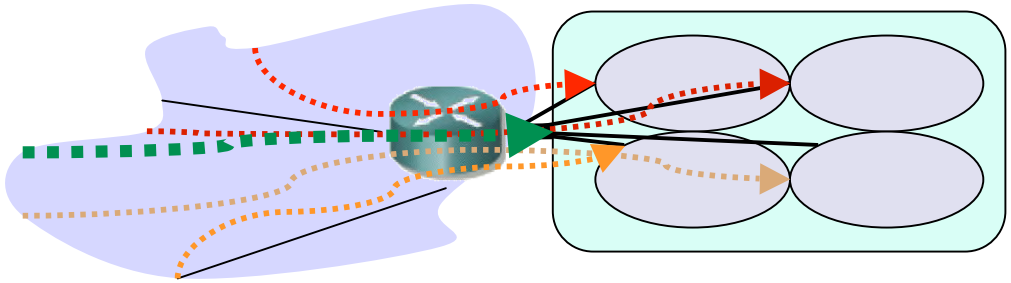
    200.45.65.0 = 11001000 00101101 01000001 00000000

    200.45.66.0 = 11001000 00101101 01000010 00000000

    200.45.67.0 = 11001000 00101101 01000011 00000000
  - Resultado (…):

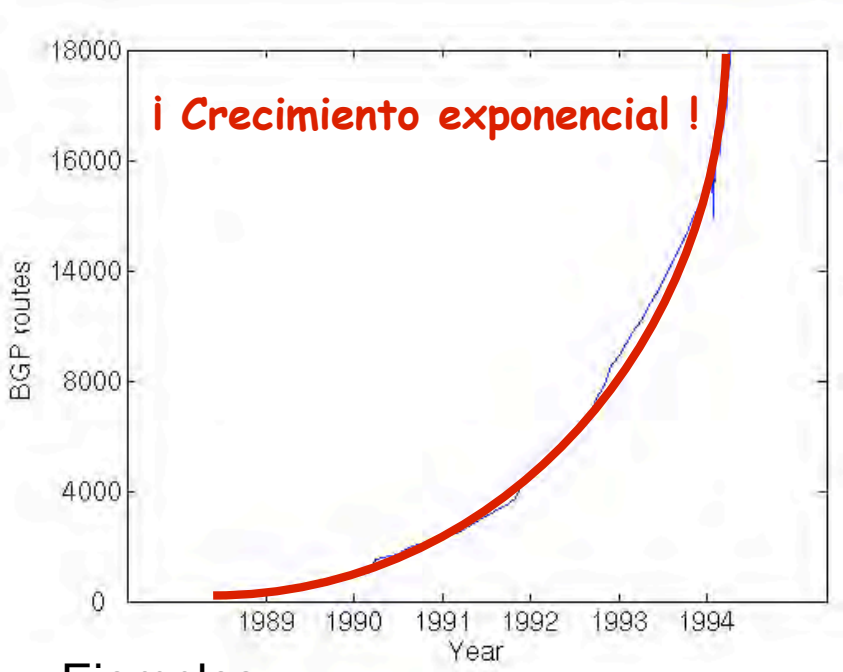
    Red 200.45.64.0, Máscara de Superred:

    255.255.252.0 = 11111111 11111111 11111100 00000000
- Una sola ruta (…)
- Máscaras en las tablas de rutas



CIDR

- Classless InterDomain Routing
- Respuesta a los problemas de:
  - Agotamiento de direcciones
  - Crecimiento de tablas de rutas (…)
- Junta VLSM y Supernetting
- Las clases (A, B y C) dejan de tener significado
- Un bloque de direcciones viene dado por:
  - Dirección de red
  - Máscara
- Slash notation = CIDR notation:
  - A.B.C.D/n
  - A.B.C.D = dirección de red (prefix)
  - n = prefix length ⇒ máscara con n bits a 1
- Evolución de las rutas (… …)



Ejemplos:

- **11001011 01100001 00000010** 00000000
  203.97.2.0/24
- **11001011 01100001 00000010 11**000000
  203.97.2.192/26
- **11001011 01100001 00**000000 00000000
  203.97.0.0/18

CIDR

Permite:

- Asignar **redes más ajustadas** al tamaño necesario
- Bloque puede estar en cualquier rango disponible (**ignora clases**)
- "**Resumir**" (*summarization*) varias rutas en una (≈Supernetting)
- Ya no existe un "Subnetwork ID"
- Ya no hay que eliminar subred 0's
- Broadcast a subredes obsoleto ⇒ se puede usar la subred 1's

- Redes privadas:
  - 10.0.0.0/8
  - 172.16.0.0/12
  - 192.168.0.0/16

Necesita:

- Rutas deben llevar máscara
- El protocolo de enrutamiento debe transportar las máscaras
- Debería hacerse un reparto manteniendo jerarquía

Regional Internet Registries (RIR):

- RIPE NCC (www.ripe.net)

  Europa, Oriente Medio, Asia Central, África norecuatorial
- ARIN (www.arin.net)

  América, parte del Caribe y África subecuatorial
- APNIC (www.apnic.net)

  Asia y Pacífico
- LACNIC (www.lacnic.net)

  América Latina y Caribe

CIDR

¿Cómo actúan los hosts y los routers?

- Tienen configurado:
  - IP en cada uno de sus interfaces
  - Máscara en cada uno
  - Tabla de rutas

Destino	Máscara	Next-hop	Interfaz
Dir.Red …	Máscara …	IP_next …	If X …

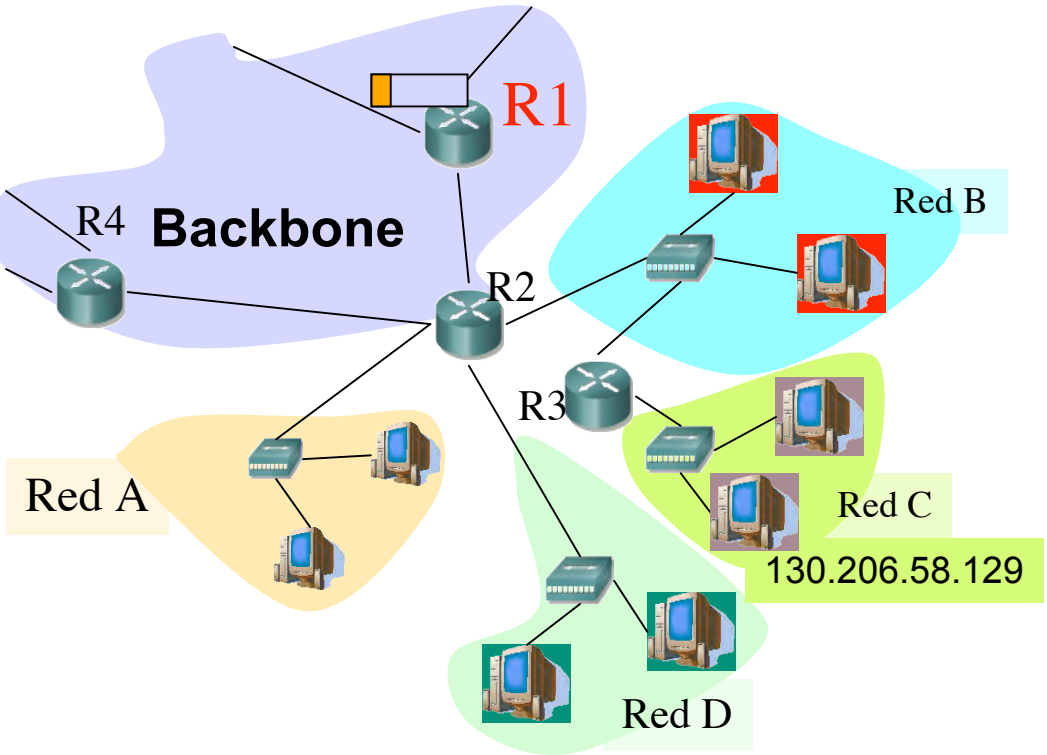
- IP_D que no es ninguna de sus direcciones IP
- La máscara no tiene por qué ser la de una red final (summaries)
- Comprueba con cada ruta si lleva hacia IP_D:
  - ((IP_D AND Máscara) == Dir.Red) ? válida : no válida
- ¿ Ninguna ruta es válida ? ⇒ descarta paquete
- Escoge la ruta válida con ***prefijo más largo*** (máscara con más 1's)
- **Longest Match**

CIDR

Ejemplo: IP_d=130.206.58.129

Destino	Next-hop	if
130.206.0.0/17	10.50.43.12 *(R2)*	1
131.57.0.0/18	(otro)	0
131.58.0.0/18	(otro)	2
…	…	…

R1



CIDR

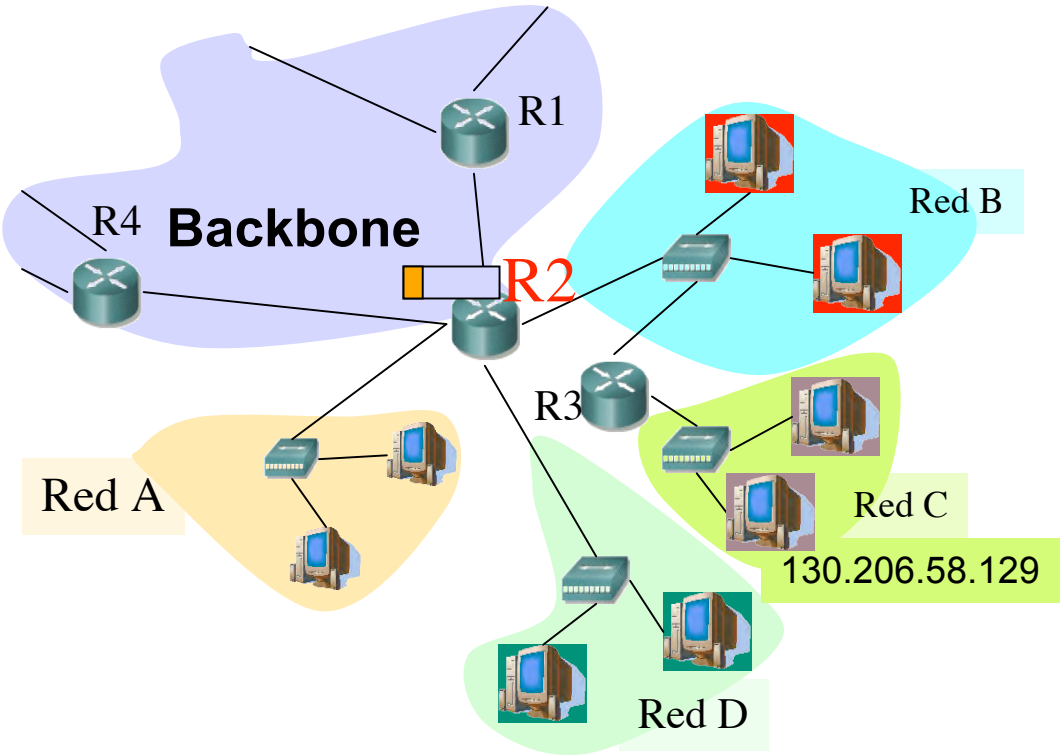
Ejemplo: IP_d=130.206.58.129

R1

Destino	Next-hop	if
130.206.0.0/17	10.50.43.12 *(R2)*	1
131.57.0.0/18	(otro)	0
131.58.0.0/18	(otro)	2
…	…	…

R2

Destino	Next-hop	if
130.206.16.0/20	-	1
130.206.56.0/21	130.206.16.1 (R3)	1
130.206.64.0/18	-	2
201.24.16.0/23	-	3
201.0.0.0/10	10.50.44.1 (R4)	4
0.0.0.0/0	10.50.43.13 (R1)	0



CIDR

Ejemplo: IP_d=130.206.58.129

R1

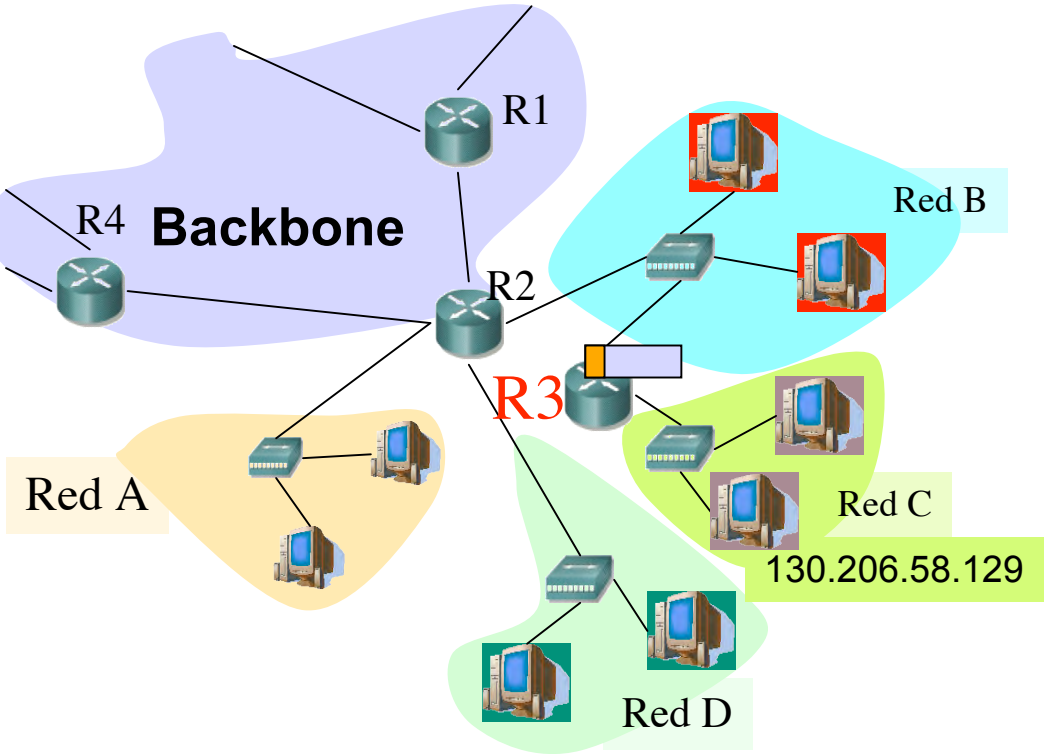
Destino	Next-hop	if
130.206.0.0/17	10.50.43.12 *(R2)*	1
131.57.0.0/18	(otro)	0
131.58.0.0/18	(otro)	2
…	…	…

R2

Destino	Next-hop	if
130.206.16.0/20	-	1
130.206.56.0/21	130.206.16.1 (R3)	1
130.206.64.0/18	-	2
201.24.16.0/23	-	3
201.0.0.0/10	10.50.44.1 (R4)	4
0.0.0.0/0	10.50.43.13 (R1)	0

R3

Destino	Next-hop	if
130.206.16.0/20	-	0
130.206.56.0/21	-	1
0.0.0.0/0	130.206.16.2 (R2)	0



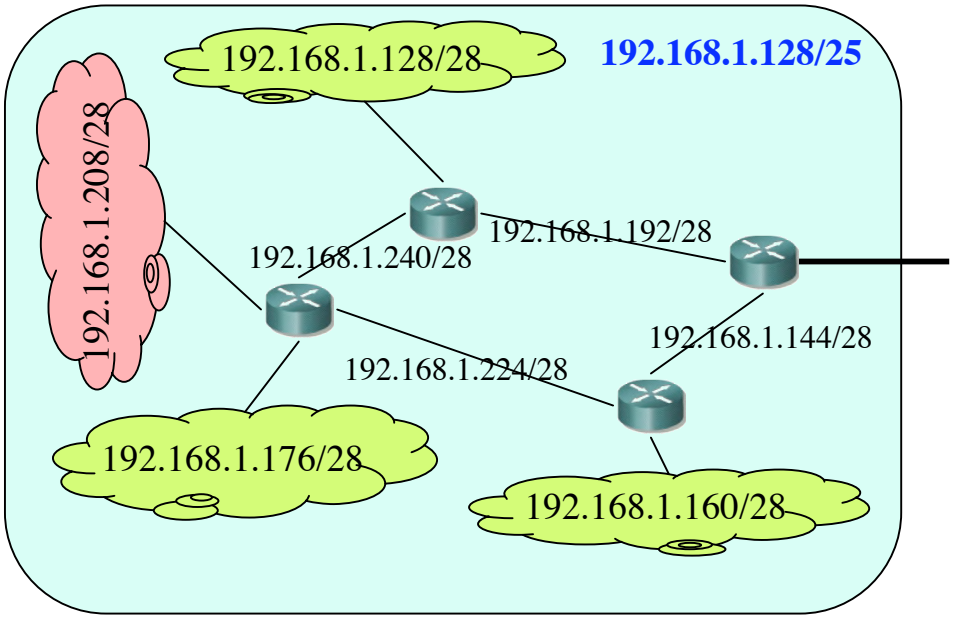
Un par de detalles sobre los *resúmenes*

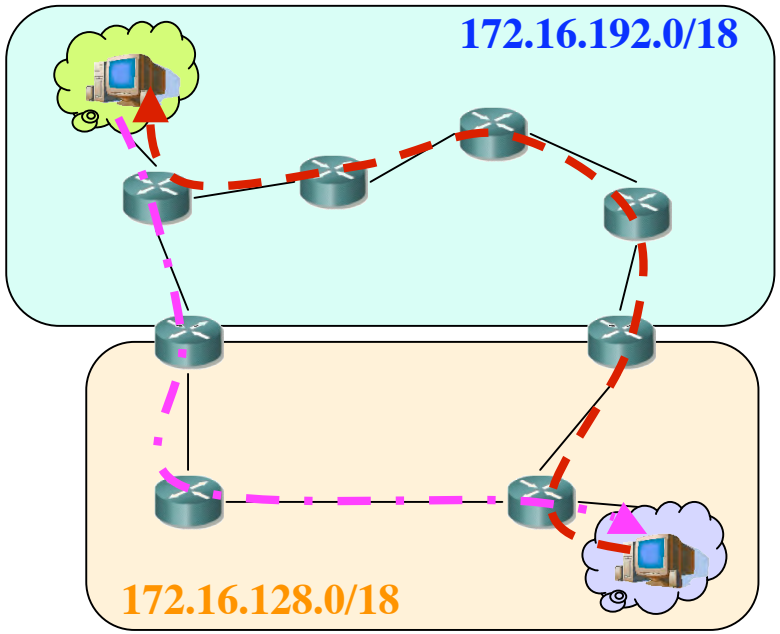
Ventajas

- Menos memoria
- Menos CPU
- Menos BW en updates
- Esconde inestabilidades (…)

Desventajas

- Menor precisión
- Puede crear asimetrías (…)





Contenido

- Internet Protocol
  - Características
  - IP en LAN Ethernet (ARP)
  - Direccionamiento (subredes, Proxy-ARP, CIDR)
  - **Fragmentación y reensamblado**
  - **ICMP**
- Enrutamiento en Internet
  - Distance-Vector
  - Link-State
  - Path-Vector
- UDP
- TCP
  - Características
  - Gestión de conexiones
  - Control de flujo
  - Retransmisiones
  - Control de congestión

Lecturas recomendadas

- [Forouzan03] 8.2, 9
- [Stevens] 8

Contenido

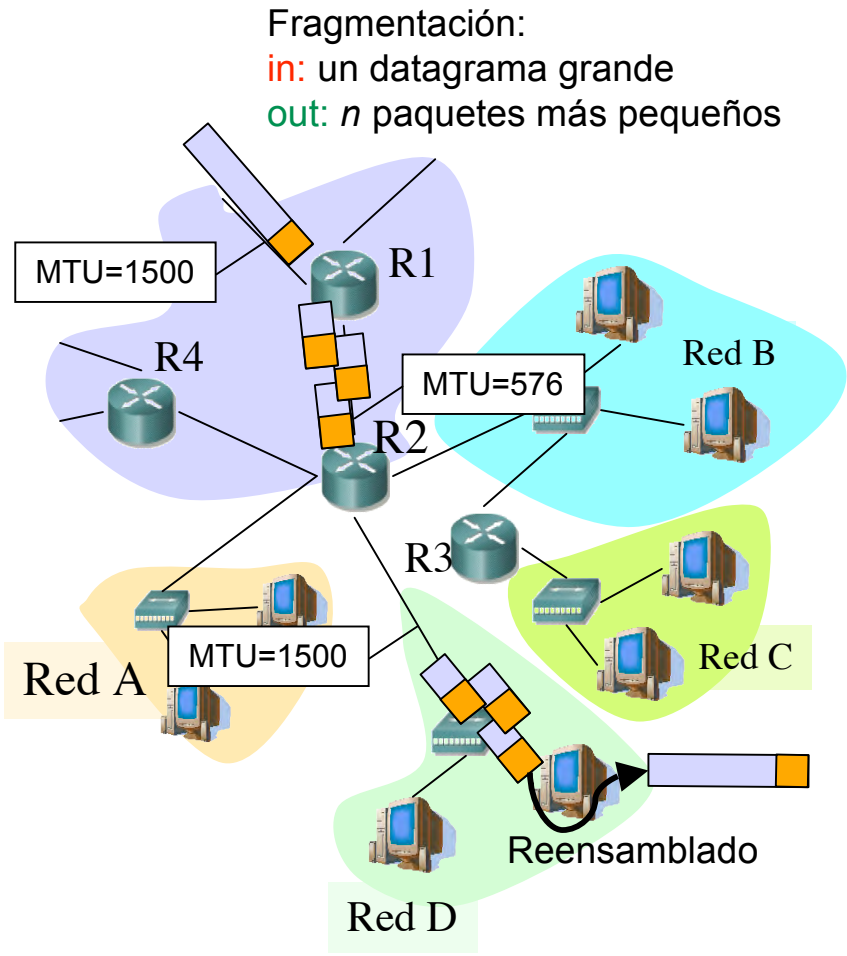
- Fragmentación y reensamblado
  - Necesidad
  - Implementación
  - Problemas
- ICMP
  - Características generales
  - Condiciones generales de envío
  - Mensajes
- Traceroute

Fragmentación y Reensamblado

Necesidad

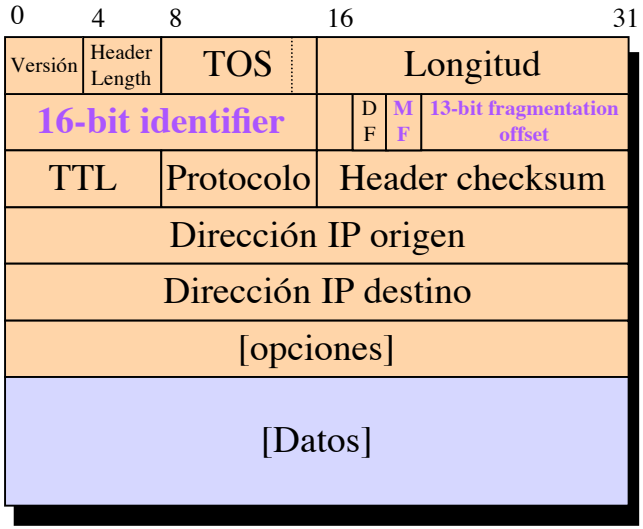
- El nivel de enlace impone unos límites al tamaño
- MTU = Maximum Transfer Unit
- Un datagrama IP es dividido dentro de la red (…)
- Un datagrama se convierte en varios paquetes
- Hosts y routers fragmentan
- *Los routers NO reensamblan (…)*
- Solo el host receptor final reensambla (…)

Red (RFC 1191)	**MTU**
16Mbps Token Ring	17914
IEEE 802.4	8166
FDDI	4352
Ethernet	1500
IEEE 802.3	1492
X.25	576



Fragmentación y Reensamblado

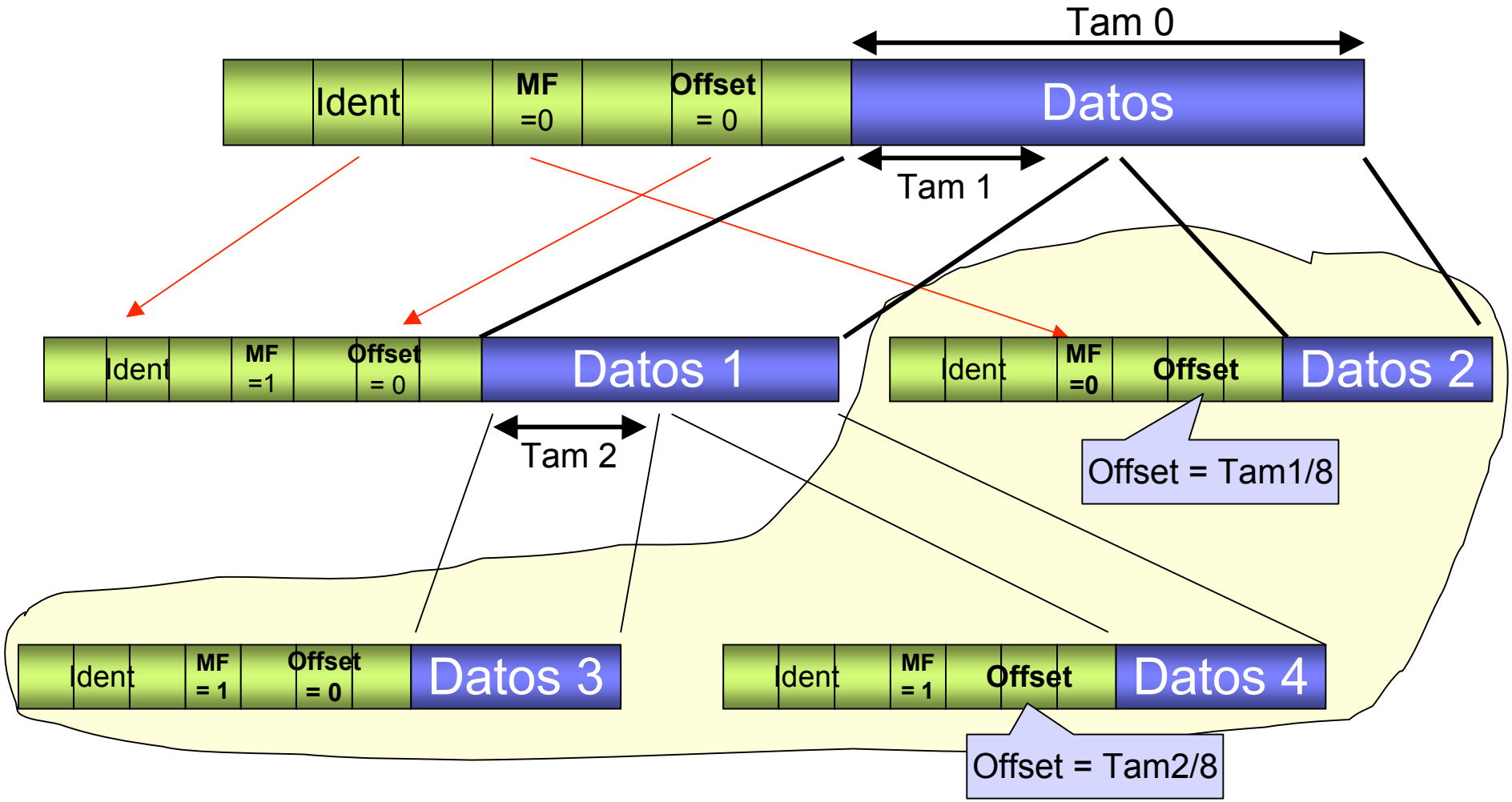
Codificación de la información



- Campos empleados:
  - Identificación
  - Bit MF
  - Fragment offset
- Fragmentos del datagrama:
  - Igual identificación, IP origen, IP destino y protocolo
- "*Longitud*" es la del paquete, no del datagrama
- Ante un primer fragmento ⇒ reservar zona de memoria donde reensamblar
- Debe reservar suficiente para reensamblar al menos datagramas de 576 Bytes

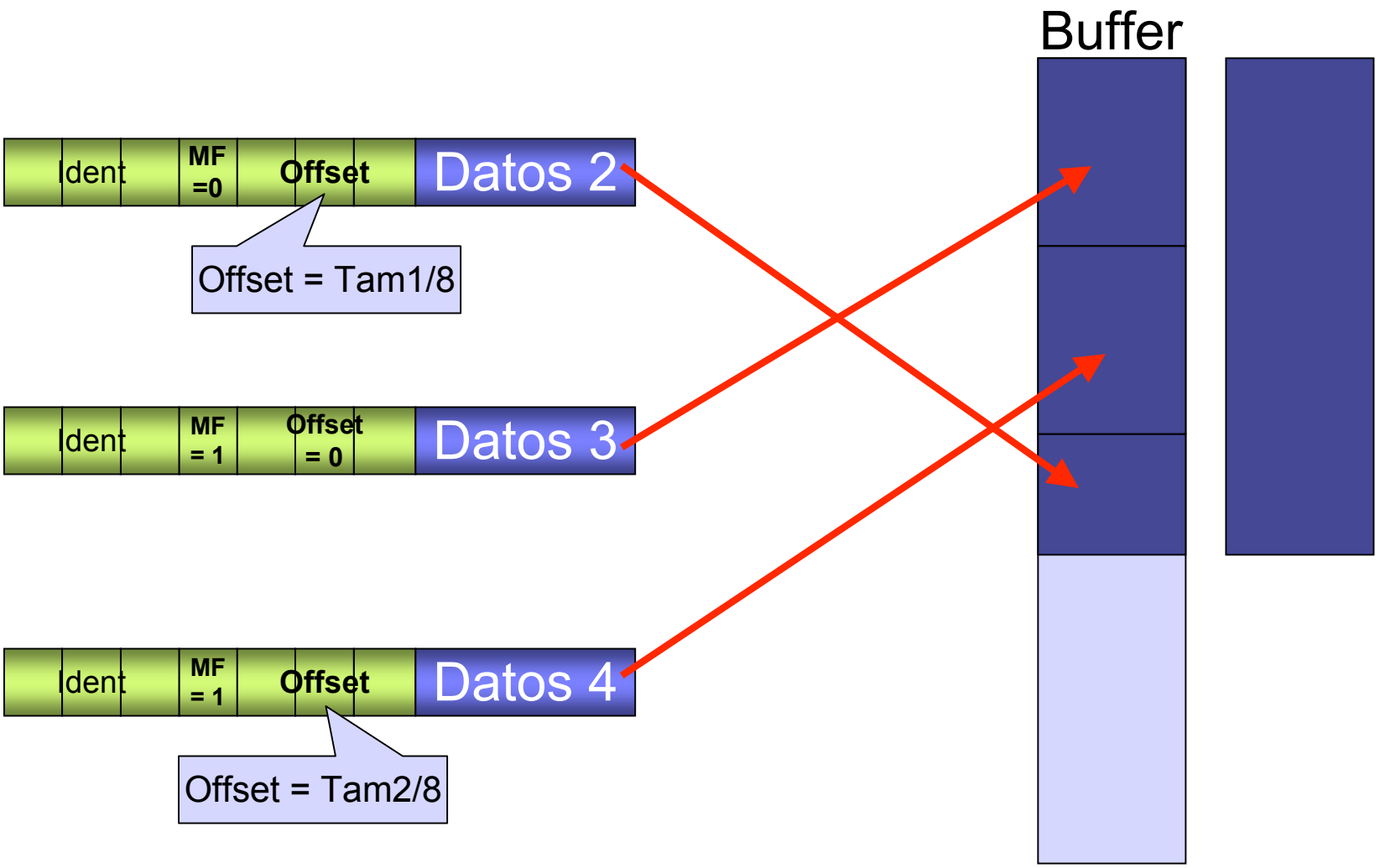
Fragmentación

Implementación



Reensamblado

Implementación



Situaciones de “error”

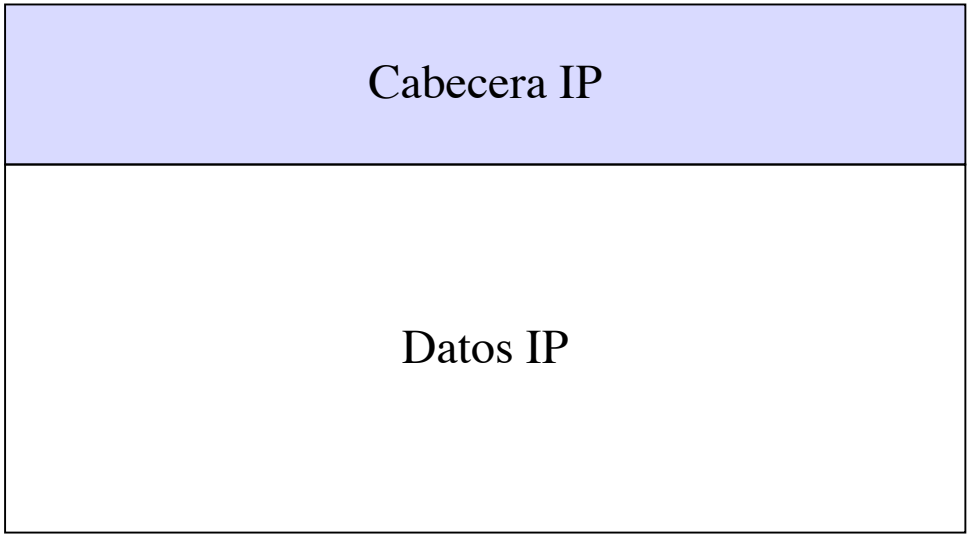
- Bit DF:
  - En la cabecera IP
  - DF==1 ⇒ routers no pueden fragmentar el paquete
  - (Tam>MTU)&&(DF==1) ⇒ lo descarta y devuelve al host origen un paquete indicando el error (ICMP)
- Reensamblado:
  - Inicia un *timer* con el primer fragmento que recibe
  - Si caduca el *timer* sin tener todos los fragmentos descarta todo lo recibido y devuelve al origen un paquete indicando el error (ICMP)

Problemas de la fragmentación

- Menor cociente Datos/Cabeceras
- Añade más carga a los routers (IPv6 la elimina)
- Si se pierde un fragmento:
  - El receptor no puede recomponer el datagrama
  - Tira todos los fragmentos recibidos
- Hasta que no se reciba todo el datagrama no se pueden pasar los datos al nivel de transporte (mayor retardo)

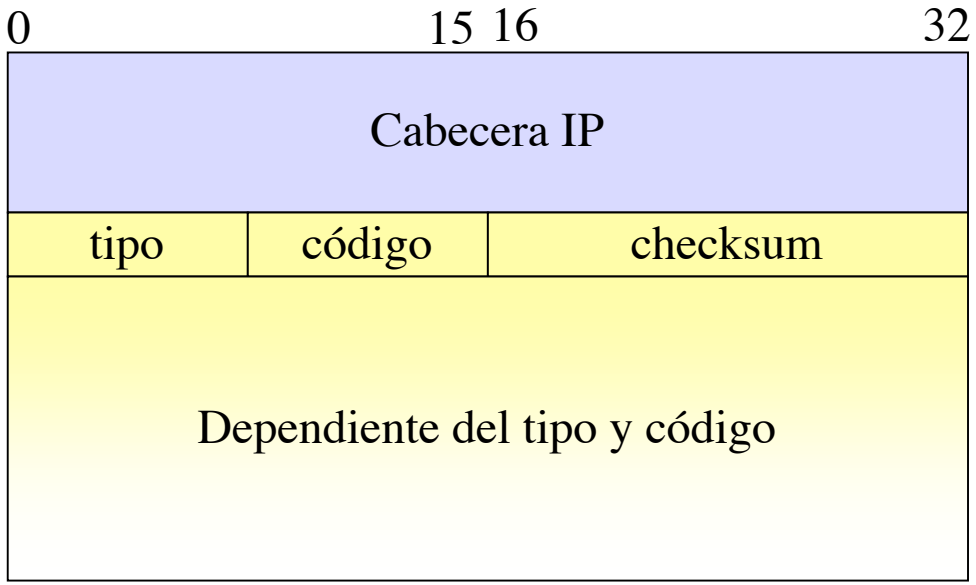
Características generales

- *Internet Control Message Protocol* (RFC 792)
- Para comunicar mensajes de error y otra información del nivel de red
- Mensajes transportados dentro de datagramas IP
- El destino es la dirección del paquete IP que generó el error
- Parte del nivel IP
- Estructura general del mensaje (…):

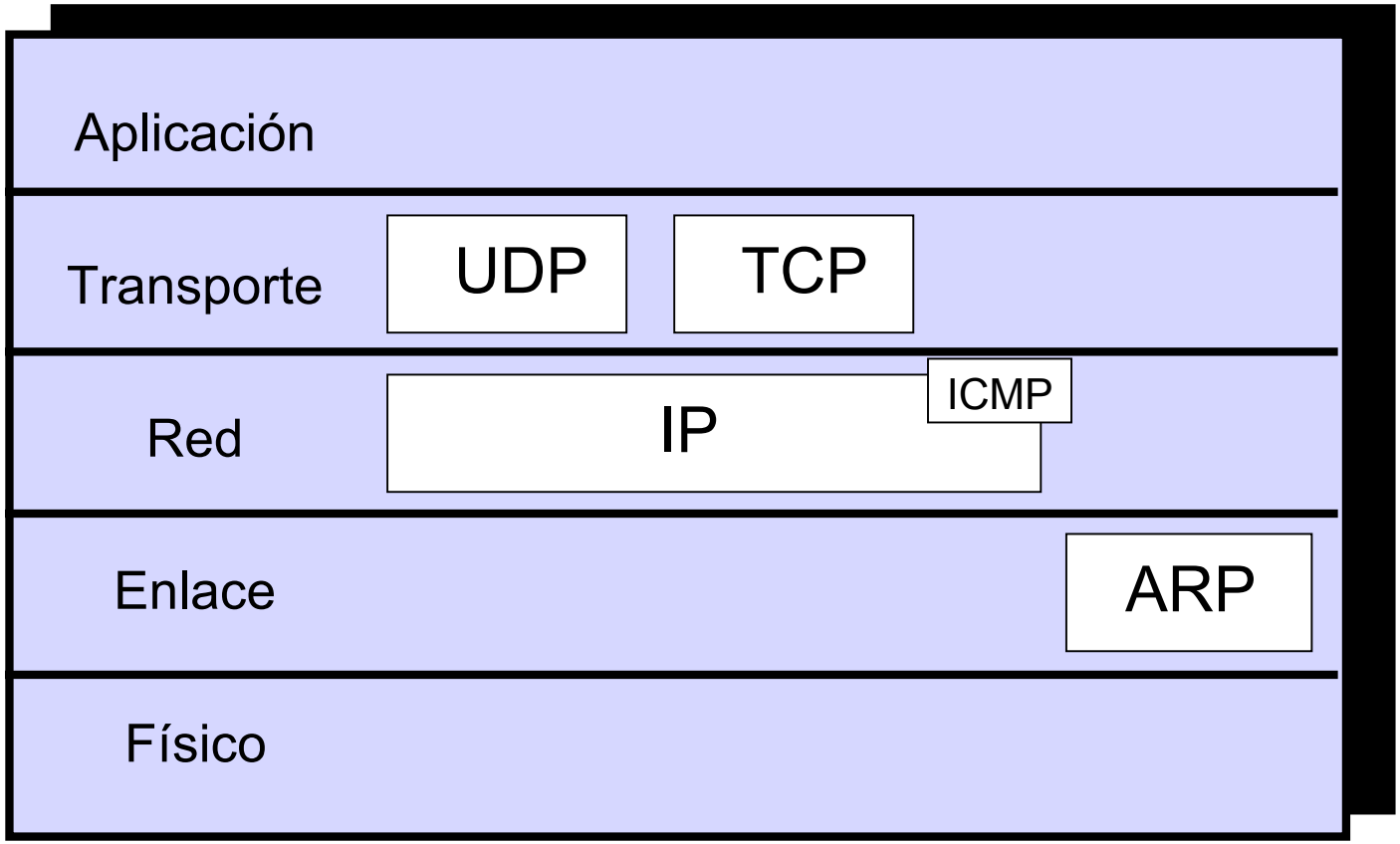


Características generales

- *Internet Control Message Protocol* (RFC 792)
- Para comunicar mensajes de error y otra información del nivel de red
- Mensajes transportados dentro de datagramas IP
- El destino es la dirección del paquete IP que generó el error
- Parte del nivel IP
- Estructura general del mensaje (…):



¿Dónde encaja ICMP en la pila TCP/IP?



Clases de mensajes ICMP

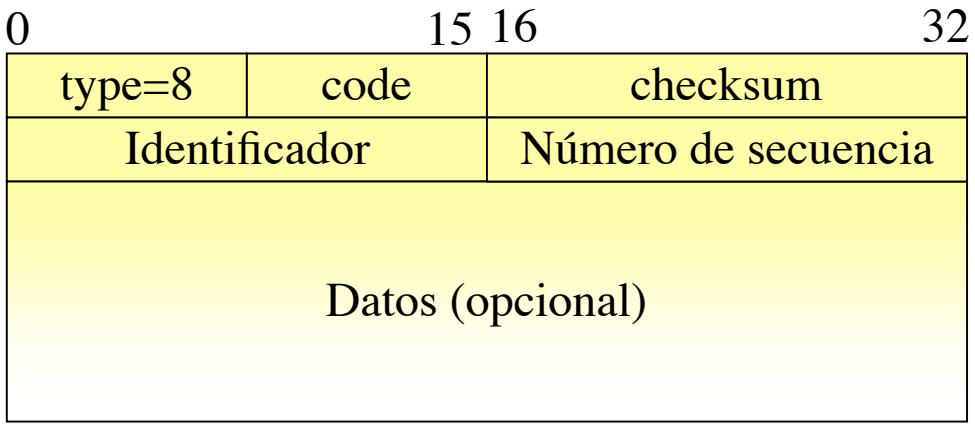
- Mensajes de Error:
  - Destino inalcanzable
  - *Redirect*
  - Tiempo excedido
  - *Source Quench*
  - Problema de parámetros
- Mensajes de pregunta (*query*):
  - Echo
  - *Router Advertisement*
  - *Timestamp*
  - Información
  - *Address Mask*

Condiciones generales de envío

- Para evitar tormentas de errores
- Nunca se generan ICMPs *de error* en respuesta a:
  - Un ICMP de error
  - Un datagrama destinado a una IP de broadcast o multicast
  - Un broadcast (o multicast) a nivel de enlace
  - Un fragmento que no sea el primero
  - Un datagrama cuya IP origen no sea *single-host*: *loopback, broadcast, multicast*

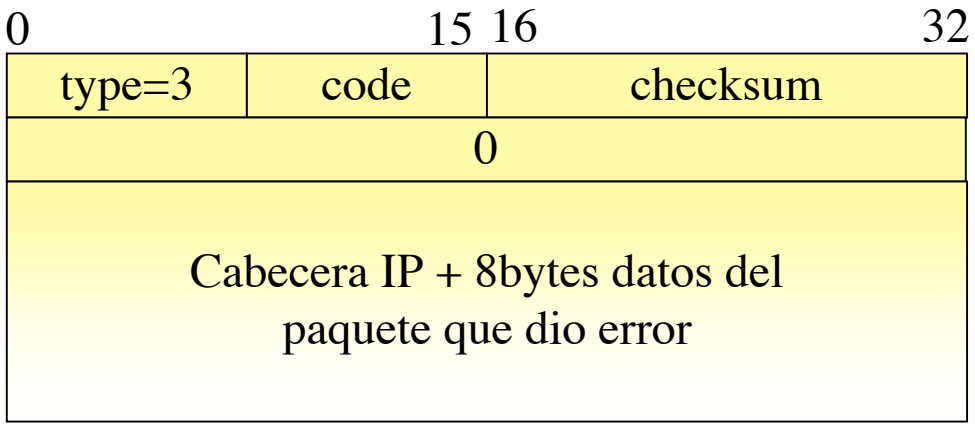
Mensajes ICMP

- Echo request/reply (*query*) (PING)
  - tipo = 8 (request) o 0 (reply), código = 0
  - Servidor debe hacer *echo* del paquete (incluidos los datos)
  - Obligatorio de implementar (generalmente en el kernel)



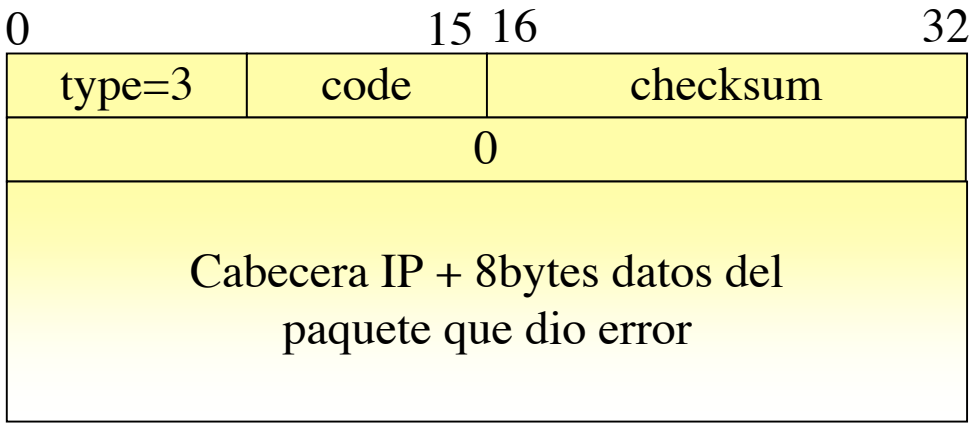
Mensajes ICMP

- Destino inalcanzable (*error*)
  - tipo = 3
  - Si según la tabla de rutas no se puede llegar al destino, host/router debe enviarlo (…)



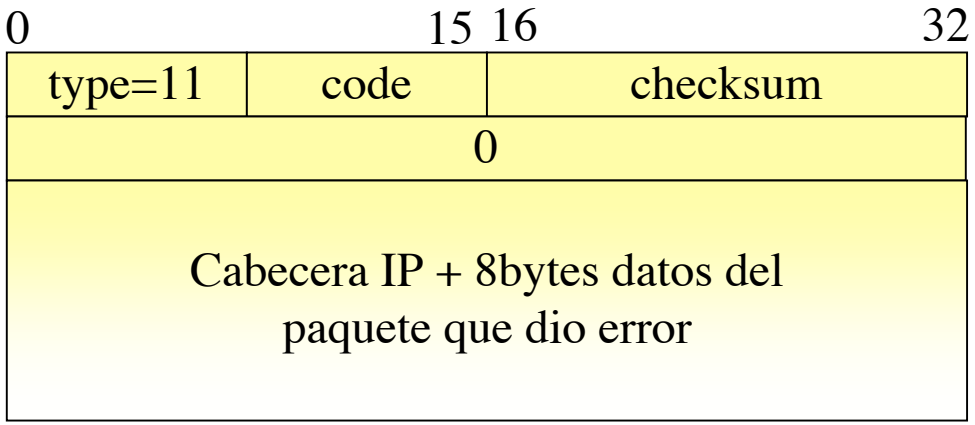
Mensajes ICMP
(Destino inalcanzable)

- Código:
  - 0 = Red destino inalcanzable
  - 1 = Host destino inalcanzable
  - 2 = Protocolo destino inalcanzable
  - 3 = Puerto destino inalcanzable
  - 4 = Fragmentación necesaria y DF activo
  - 5 = Source route failed



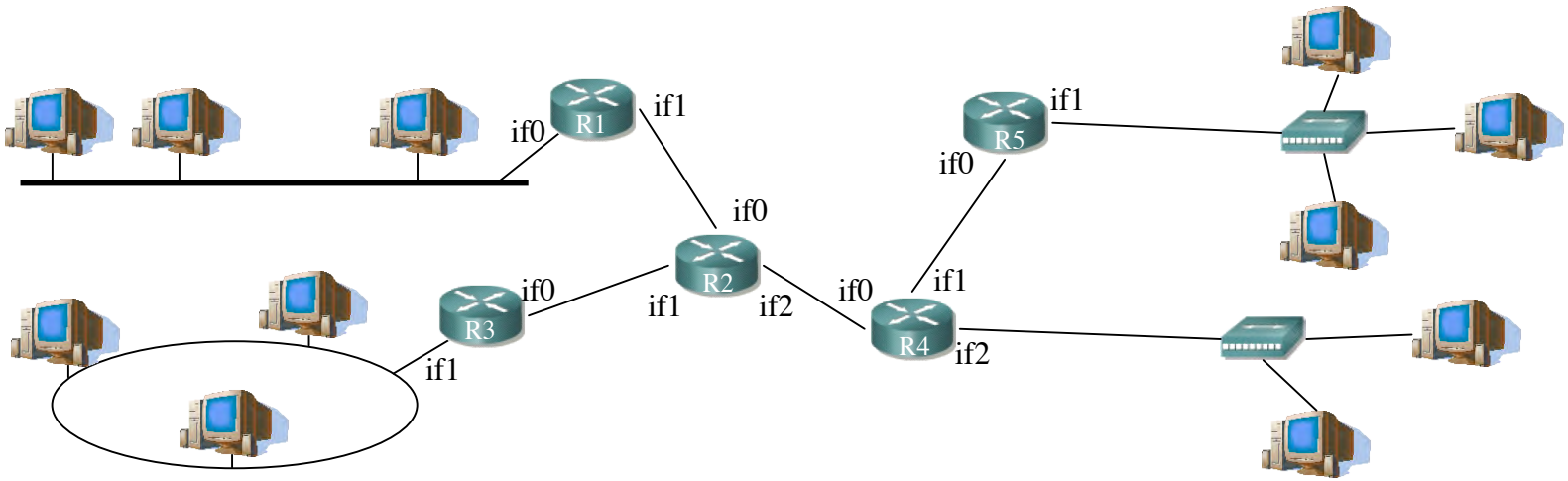
Mensajes ICMP

- Tiempo excedido (*error*)
  - tipo = 11
  - código = 0 (TTL=0 en tránsito), 1 (timeout durante reensamblado, necesita primer paquete)



Traceroute

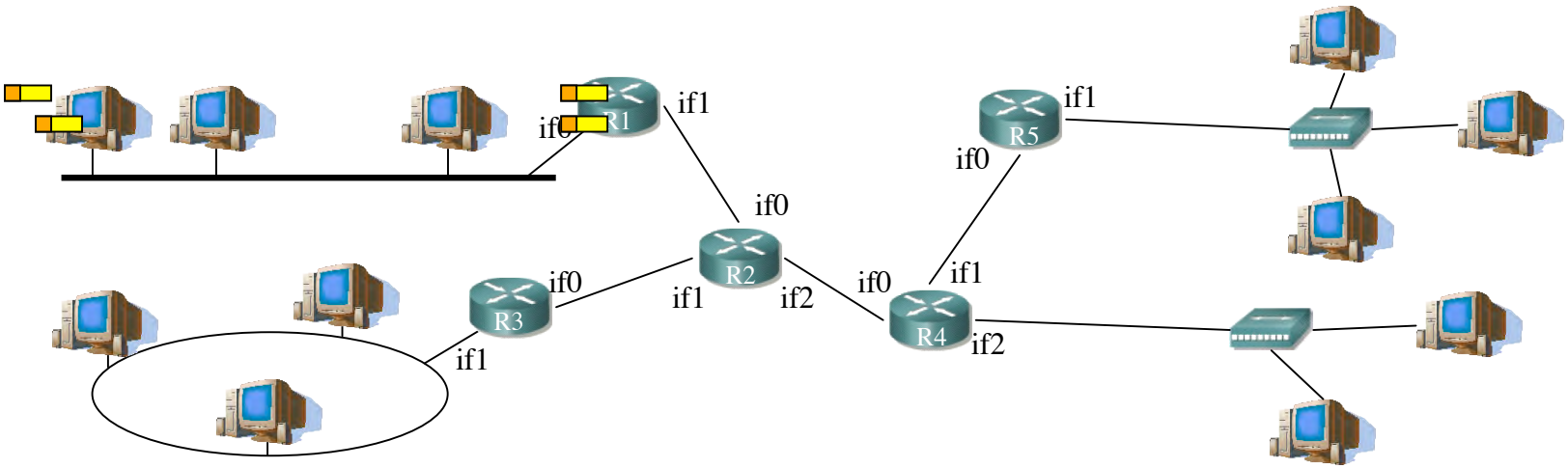
- Permite averiguar *el camino* entre dos hosts
- Suponiendo que el camino se mantiene entre diferentes paquetes
- Requiere que el destino final soporte UDP
- Requiere que se generen ciertos mensajes ICMP
- *Implemented by **Van Jacobson** from a suggestion by Steve Deering. Debugged by a cast of thousands with particularly cogent suggestions or fixes from C. Philip Wood, Tim Seaver, and Ken Adelman.*



Traceroute

- El host inicial envía un datagrama UDP (…)
  - Dirigido al host final
  - Con TTL = 1
- El primer router decrementa el TTL a 0 (…)
  - Tira el paquete
  - Devuelve al origen un ICMP de Error *Tiempo excedido en tránsito*
  - Este es un paquete IP con dirección origen la del interfaz de R1 en la red del host (… …)

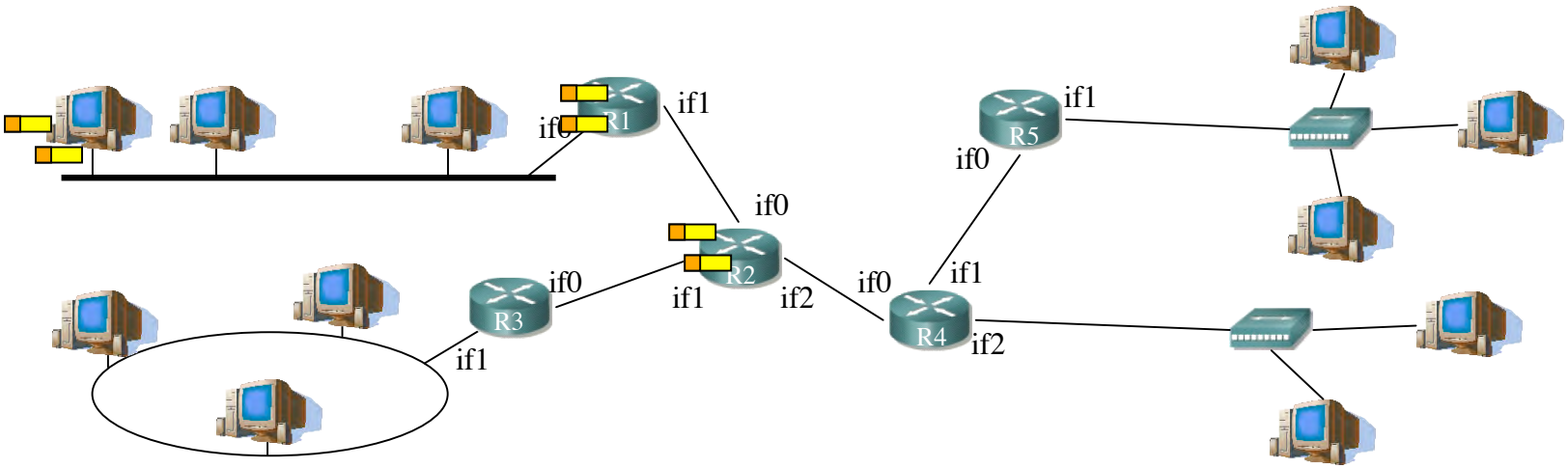
TTL	IP
1	$IPR1_{if0}$



Traceroute

- El host inicial envía un datagrama UDP (…)
  - Dirigido al host final
  - Con TTL = 2
- El primer router decrementa el TTL a 1 y lo reenvía (…)
- El segundo router decrementa el TTL a 0 (…)
  - Tira el paquete
  - Devuelve al origen un ICMP de Error *Tiempo excedido en tránsito*
  - Este es un paquete IP con dirección origen la del interfaz de R2 en dirección hacia el host origen (…)

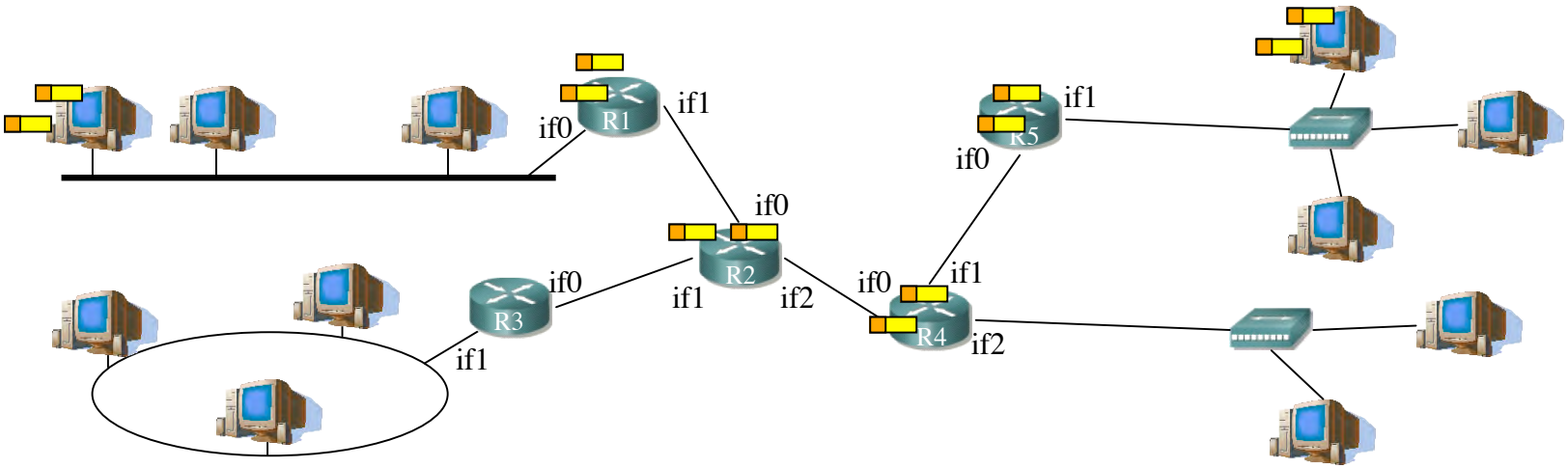
TTL	IP
1	$IPR1_{if0}$
2	$IPR2_{if0}$



Traceroute

- Idem con TTL=3 y TTL=4 (…)
- Con TTL suficientemente grande el paquete llega hasta el destino final (… …)
- En el destino no hay aplicación esperando paquetes UDP en ese puerto:
  - Lo tira
  - Devuelve al origen un ICMP de Error *Puerto destino inalcanzable (… …)*

TTL	IP
1	$IPR1_{if0}$
2	$IPR2_{if0}$
3	$IPR4_{if0}$
4	$IPR5_{if0}$
5	IPhost



Traceroute (Ejemplo)

```
daniel% traceroute www.berkeley.edu
traceroute to arachne.berkeley.edu (169.229.131.109), 30 hops max, 40 byte packets
 1  arce-un.red.unavarra.es (130.206.160.1)  1.691 ms  0.438 ms  0.417 ms
 2  ss-in (130.206.158.25)  1.015 ms  3.091 ms  0.658 ms
 3  unavarra-router.red.unavarra.es (130.206.158.1)  1.587 ms  1.87 ms  1.506 ms
 4  fe0-1-2.eb-pamplona0.red.rediris.es (130.206.209.13)  1.49 ms  1.741 ms  1.25 ms
 5  nav.so2-3-0.eb-bilbao0.red.rediris.es (130.206.240.61)  5.279 ms  4.402 ms  4.398 ms
 6  pav.so2-0-0.eb-iris2.red.rediris.es (130.206.240.29)  50.039 ms  16.511 ms  16.35 ms
 7  so0-0-0.eb-iris4.red.rediris.es (130.206.240.2)  16.341 ms  17.982 ms  16.405 ms
 8  rediris.es1.es.geant.net (62.40.103.61)  118.998 ms  16.741 ms  16.755 ms
 9  es.it1.it.geant.net (62.40.96.186)  96.679 ms  39.288 ms  39.513 ms
10  it.de2.de.geant.net (62.40.96.61)  91.118 ms  48.088 ms  49.83 ms
11  abilene-gw.de2.de.geant.net (62.40.103.254)  141.935 ms  141.812 ms  141.716 ms
12  atlang-washng.abilene.ucaid.edu (198.32.8.65)  157.505 ms  157.692 ms  164.648 ms
13  hstnng-atlang.abilene.ucaid.edu (198.32.8.33)  177.182 ms  177.144 ms  177.201 ms
14  losang-hstnng.abilene.ucaid.edu (198.32.8.21)  199.049 ms  198.489 ms *
15  hpr-lax-gsr1--abilene-la-10ge.cenic.net (137.164.25.2)  199.004 ms  198.621 ms  284.873 ms
16  svl-hpr--lax-hpr-10ge.cenic.net (137.164.25.13)  215.55 ms  218.166 ms  206.364 ms
17  hpr-ucb-ge--svl-hpr.cenic.net (137.164.27.134)  210.841 ms  207.409 ms  207.479 ms
18  vlan187.inr-201-eva.berkeley.edu (128.32.0.33)  283.445 ms  207.842 ms  207.318 ms
19  g5-1.inr-210-srb.berkeley.edu (128.32.255.65)  211.052 ms  207.341 ms  207.408 ms
20  arachne.berkeley.edu (169.229.131.109)  207.431 ms  207.451 ms  207.4 ms
```

Probarlo y ver los paquetes con tcpdump

Contenido

- Internet Protocol
  - Características
  - IP en LAN Ethernet (ARP)
  - Direccionamiento (subredes, Proxy-ARP, CIDR)
  - Fragmentación y reensamblado
  - ICMP
- **Enrutamiento en Internet**
  - Distance-Vector
  - Link-State
  - Path-Vector
- UDP
- TCP
  - Características
  - Gestión de conexiones
  - Control de flujo
  - Retransmisiones
  - Control de congestión

Lecturas recomendadas

- [Kurose05] 1.5, 4.5.3

Contenido

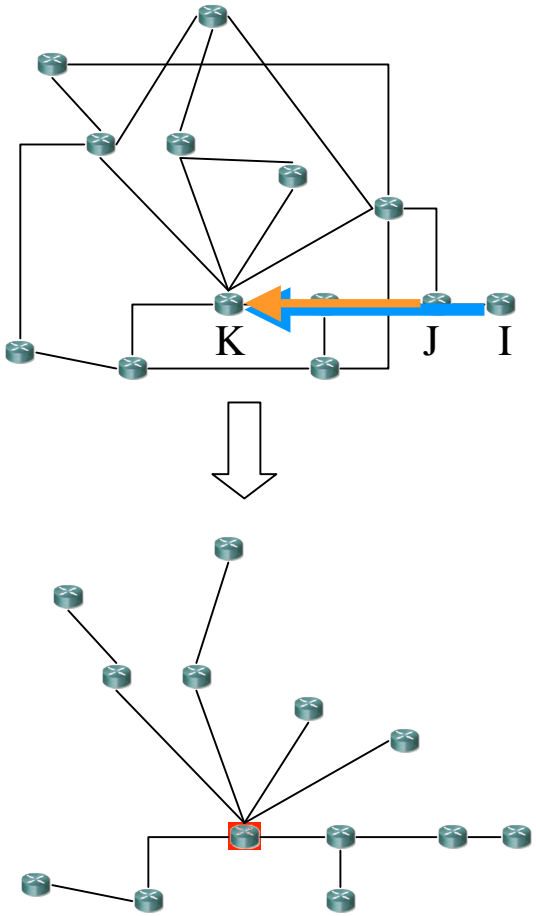
- Introducción
- Enrutamiento jerárquico
  - IGPs
  - EGPs
- Estructura de Internet

Funciones del nivel de red

- Forwarding (data plane)
- Routing (control plane)

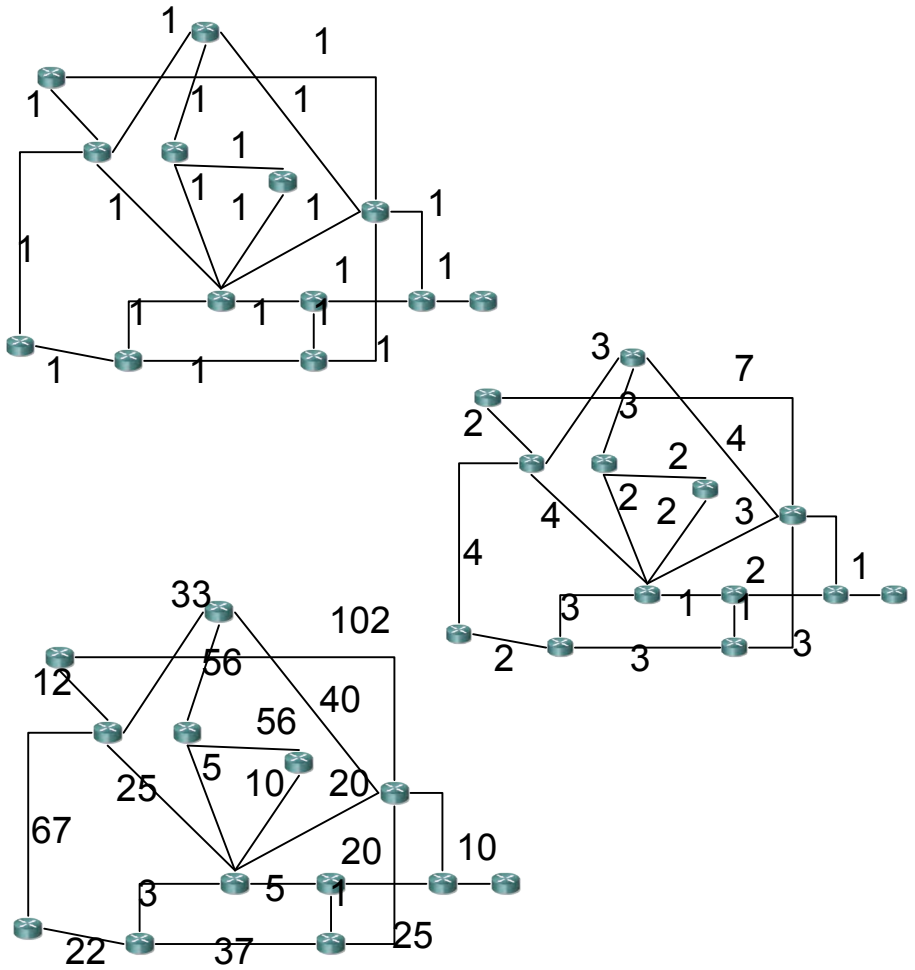
Principio de optimalidad

- *Si router J está en el camino óptimo desde I a K entonces el camino óptimo de J a K está en la misma ruta (…)*
- Si existiera una ruta mejor de J a K se podría concatenar con el de I a J
- El conjunto de rutas óptimas a un destino es un árbol = ***sink tree*** (…)
- Árbol ⇒ sin lazos (*loops*)



¿Camino óptimo?

- ***Shortest path***
- ¿Cómo medirlo?
  - Número de *saltos*
  - Distancia geográfica
  - Retardo
- Peso de cada vértice:
  - BW
  - Tráfico medio
  - Coste (€€)
  - Longitud media de cola
  - Combinación



Construcción de las tablas de rutas

¿Estática o dinámica?

Estática:

- Configuración manual
- Cambios lentos

Dinámica:

- Mediante un protocolo de enrutamiento
  - Escalabilidad
  - Adaptabilidad
  - Complejidad

¿Información global o descentralizada?

Global:

- Todos los routers tienen información completa de la topología y los costes de los enlaces
- Algoritmos “link state”

Descentralizada:

- El router conoce solo a sus vecinos
- Mediante un proceso iterativo intercambia esa información con sus vecinos
- Algoritmos “distance vector”

Enrutamiento jerárquico

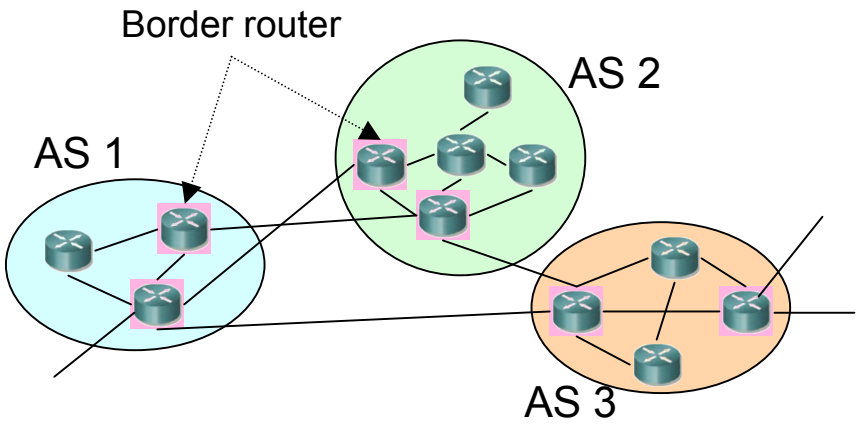
Escala: con centenares de millones de destinos

- ¡No se pueden tener todos los destinos en las tablas de rutas!
  - Memoria
  - CPU
  - BW para informar de rutas

- Autonomía administrativa
- Cada administrador de red quiere controlar el enrutamiento dentro de su red

Enrutamiento jerárquico

- Agrupar routers en regiones: "Autonomous Systems" (AS)
- Routers de un AS un solo administrador
- Normalemente los routers en el mismo AS emplean el mismo protocolo de enturamiento
  - IGP = Interior Gateway Protocol
  - Routers en diferentes AS pueden emplear diferente IGP
  - Interior *oculto*
- Comunicar información de enrutamiento entre los AS
  - EGP = Exterior Gateway Protocol
  - Entre los border routers o routers frontera de los AS

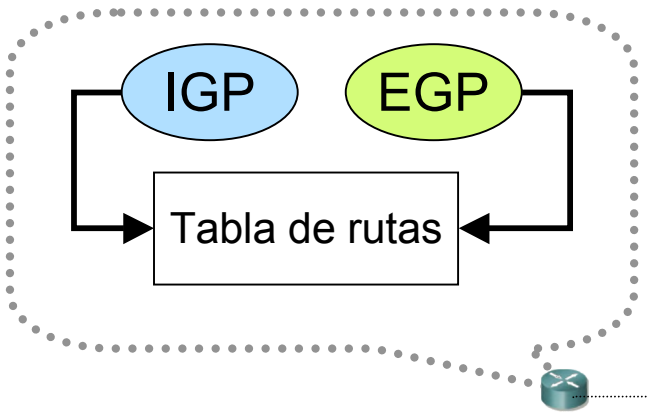


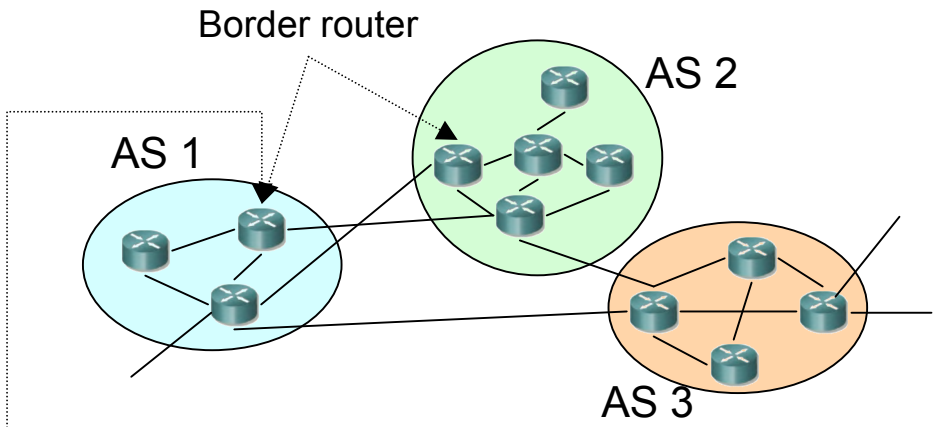
Enrutamiento jerárquico

Border router

- La tabla de rutas es configurada por ambos
- IGP: rutas a destinos internos
- EGP: rutas a destinos externos

- IGP da las rutas internas
- ¿Si hay más de un enlace al exterior?
  - EGP debe informar de a qué destinos se puede llegar por cada uno





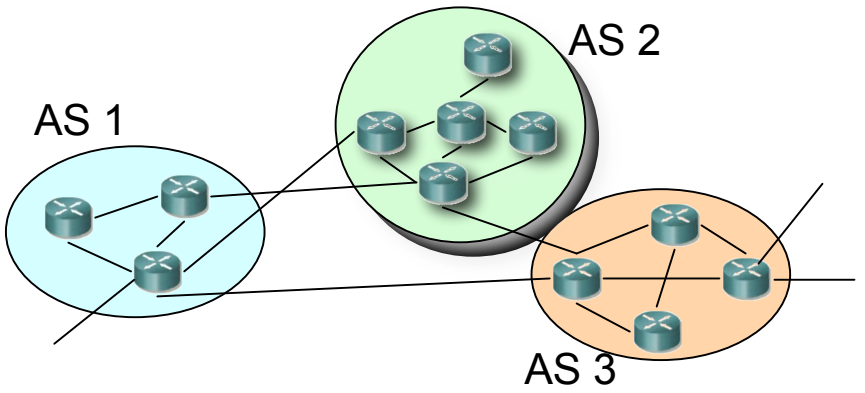
Interior Gateway Protocols (IGP)

Características:

- Simples
- Calculan caminos eficientes respecto a una métrica
- Recalculan rápidamente ante cambios
- No escalan bien para redes grandes

Los más comunes:

- RIP: Routing Information Protocol
- OSPF: Open Shortest Path First
- IGRP: Interior Gateway Routing Protocol (propietario de Cisco)



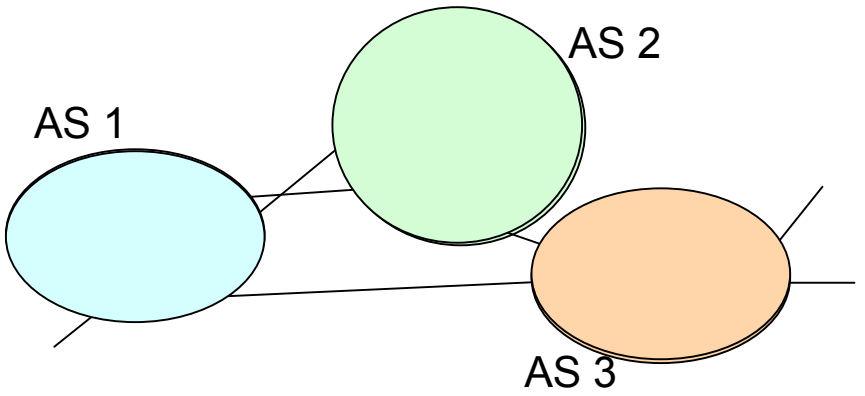
Exterior Gateway Protocols (EGP)

Características:

- Mejor escalabilidad
- Habilidad para agregar rutas
- Habilidad para expresar políticas
- Mayor carga en el router

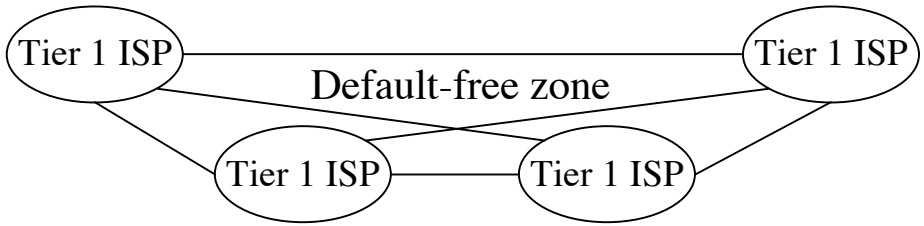
BGP (Border Gateway Protocol): *estándar de facto*

- Algoritmo *path-vector* : anuncia el camino completo al destino (como una secuencia de ASs)
- Los anuncios emplean conexiones TCP entre los routers



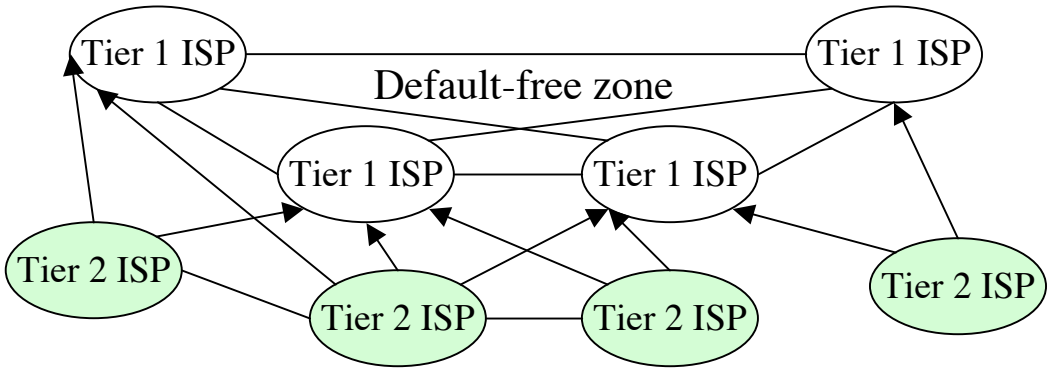
Estructura de Internet

- Tier-1 ISPs o Internet backbone networks
  - Grandes proveedores internacionales (AT&T, BBN, BT, Cable&Wireless, Sprint, UUNET, etc.)
  - Conexión completamente mallada
  - No emplean “ruta por defecto”, tienen rutas a todas las redes (Junio04: 140.396 rutas)



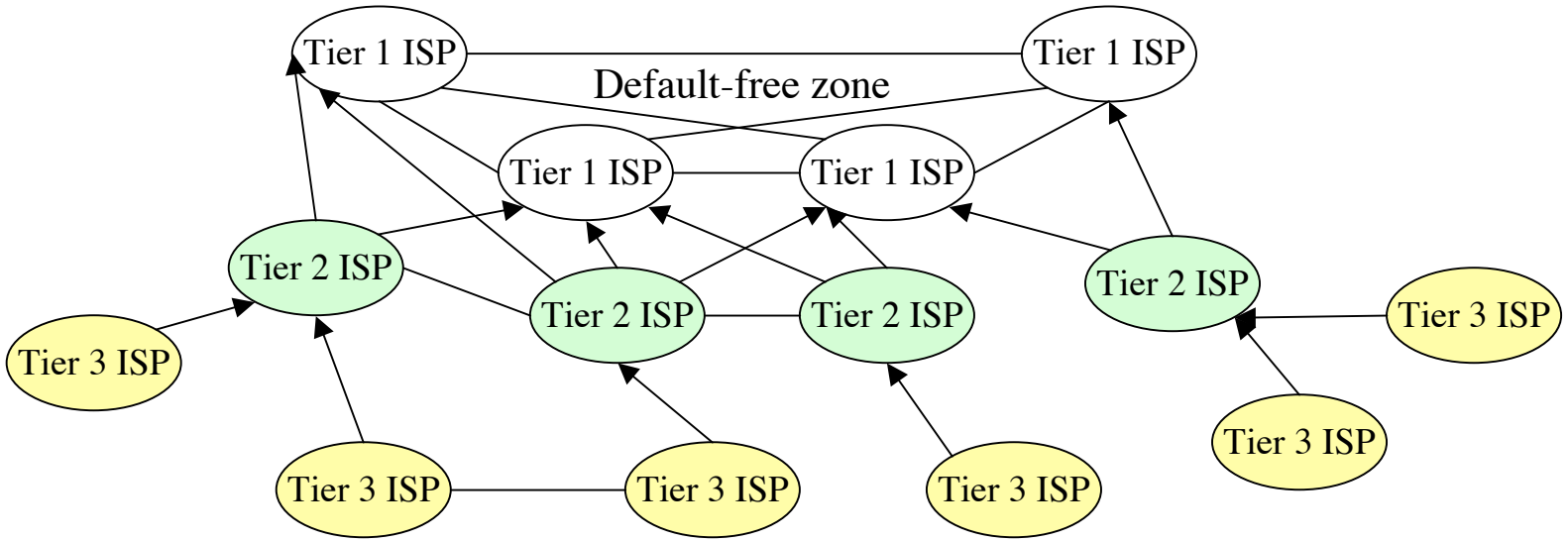
Estructura de Internet

- Tier-2 ISPs
  - Regionales o nacionales
  - Se conectan (peering agreement) a unos pocos tier-1 ISPs (ellos son los clientes y el tier-1 el proveedor de tránsito)
  - Se pueden conectar a otros tier-2



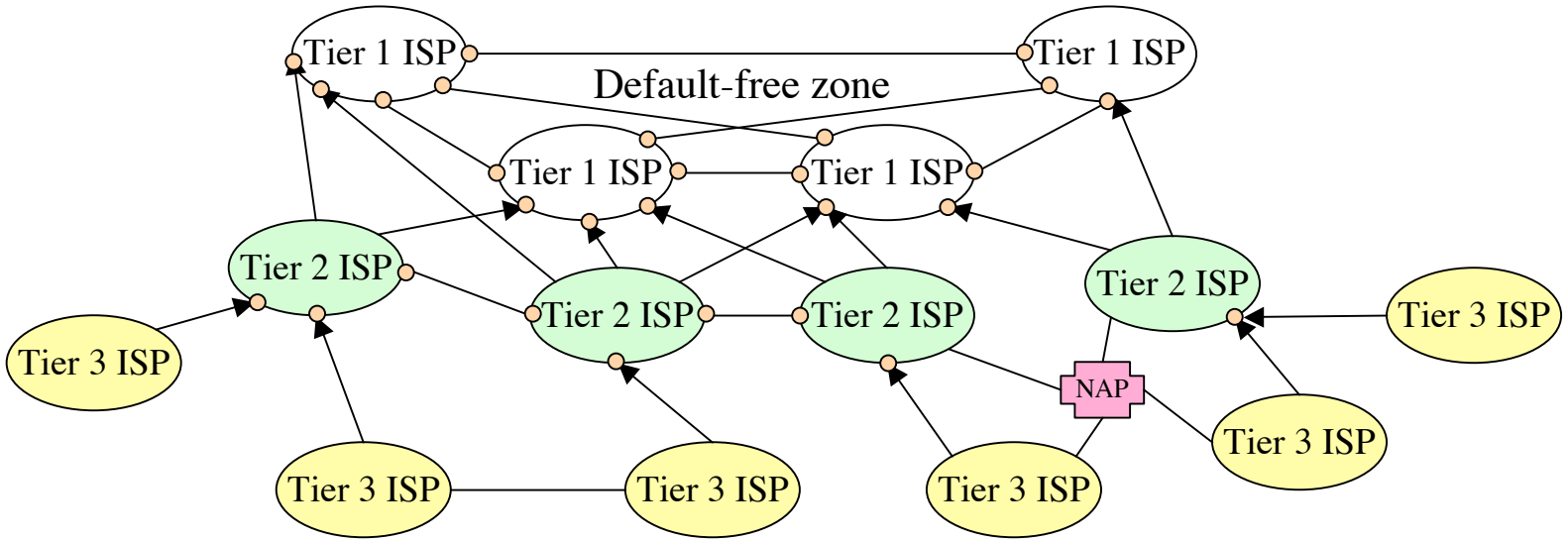
Estructura de Internet

- Tier-3 ISPs
  - ISPs locales de acceso
  - Se conectan a uno o más tier-2 y entre ellos



Estructura de Internet

- ***Points of Presence*** (POPs)
- ***NAPs*** (Network Access Points) o ***IXP*** (Internet eXchange Point)
  - Son redes de alta velocidad en sí mismas
  - Pretenden ahorrar €€
  - Reducir retardo
  - Mantener local el tráfico local (ej: Espanix)



Contenido

- Internet Protocol
  - Características
  - IP en LAN Ethernet (ARP)
  - Direccionamiento (subredes, Proxy-ARP, CIDR)
  - Fragmentación y reensamblado
  - ICMP
- Enrutamiento en Internet
  - **Distance-Vector**
  - **Link-State**
  - **Path-Vector**
- UDP
- TCP
  - Características
  - Gestión de conexiones
  - Control de flujo
  - Retransmisiones
  - Control de congestión

Lecturas recomendadas

- [Kurose05] 4.5

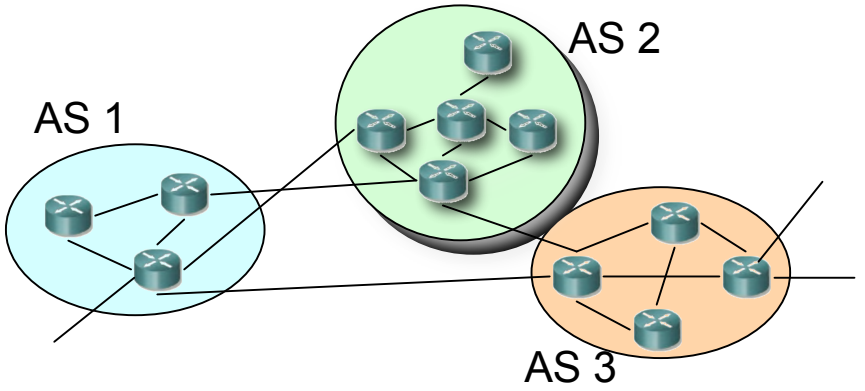
Contenido

- Introducción
- Algoritmos Link-State
  - Descripción
  - Dijkstra
- Algoritmos Distance-Vector
  - Descripción
  - Bellman-Ford
- Algoritmos Path-Vector

Tipos de Protocolos de Enrutamiento

Enrutamiento jerárquico

- Sistemas Autónomos (AS)
- Dentro de un AS:
  - ***IGP = Interior Gateway Protocol***
- Entre ASs:
  - ***EGP = Exterior Gateway Protocol***



Tipos de Algoritmos de Enrutamiento

- Deben informar de la topología y los cambios en la misma
- Según cómo diseminan la información

Link State:

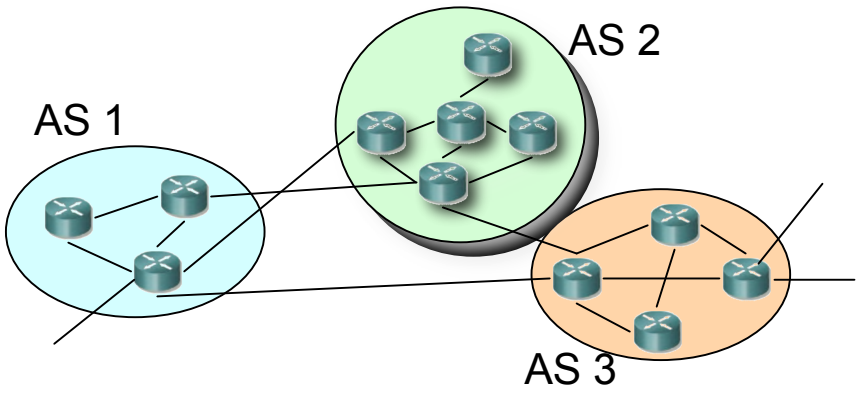
- Comunican qué vecinos tienen y el coste
- Inundan la red
- Cada nodo conoce la topología entera

Distance Vector:

- Comunican estimación de distancia a destinos
- Informan a vecinos

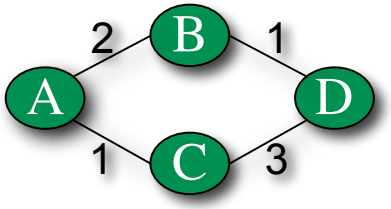
Path Vector:

- Comunican estimación de caminos preferidos a destinos



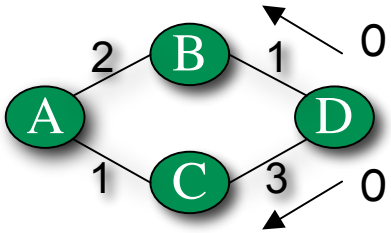
Tipos de Algoritmos de Enrutamiento

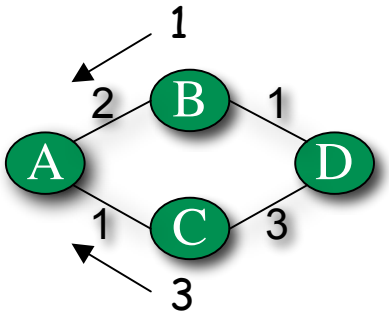
Link State

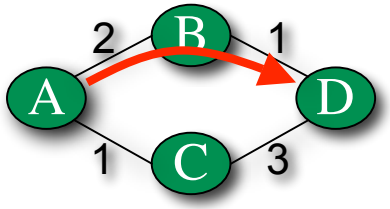


A: [B, 2], [C, 1]
B: [A, 2], [D, 1]
C: [A, 1], [D, 3]
D: [B, 1], [C, 3]

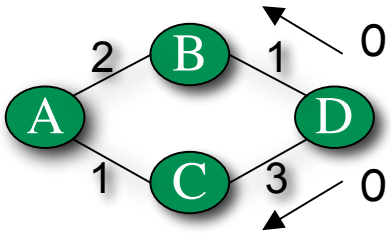
Distance Vector

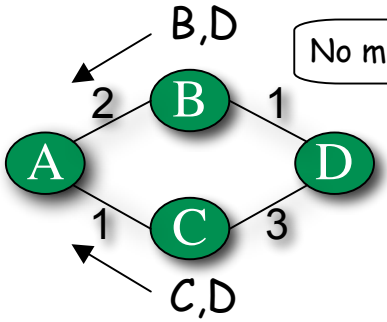


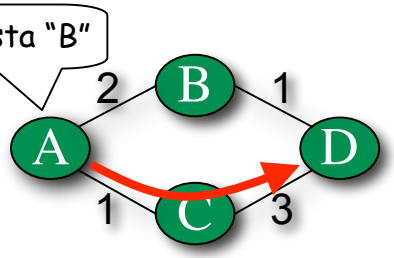




Path Vector



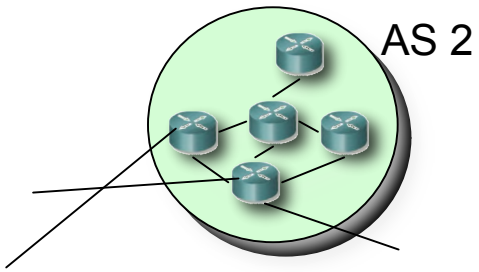


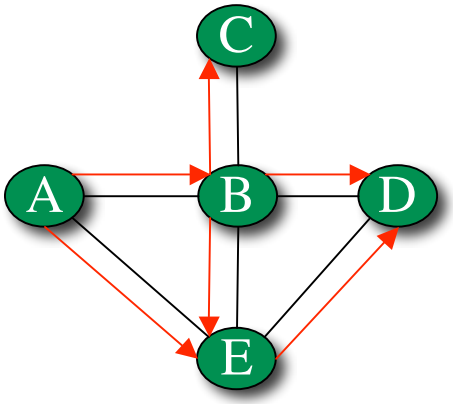


Link State

Tres pasos

1. Descubrir a los vecinos
2. Diseminar la información sobre los enlaces
   - ***Flooding*** (… …)
   - Todos conocen la topología (…)
   - (En profundidad más adelante)
3. Calcular las rutas
   - Caminos de menor coste
   - Todos calculan los mismos
   - Algoritmo de *Dijkstra*

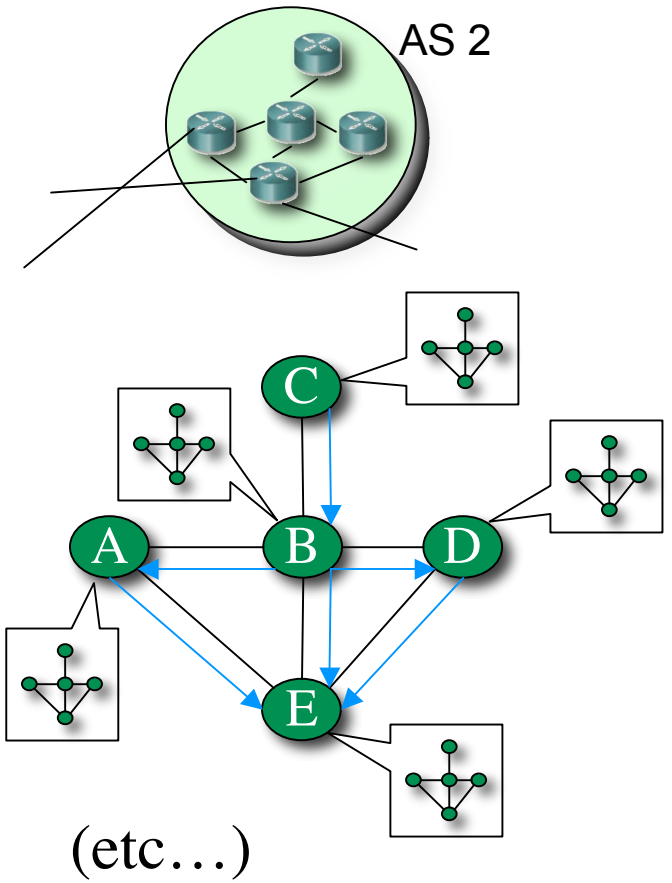




Link State

Tres pasos

1. Descubrir a los vecinos
2. Diseminar la información sobre los enlaces
   - ***Flooding*** (… …)
   - Todos conocen la topología (…)
   - (En profundidad más adelante)
3. Calcular las rutas
   - Caminos de menor coste
   - Todos calculan los mismos
   - Algoritmo de *Dijkstra*



Algoritmo de Dijkstra

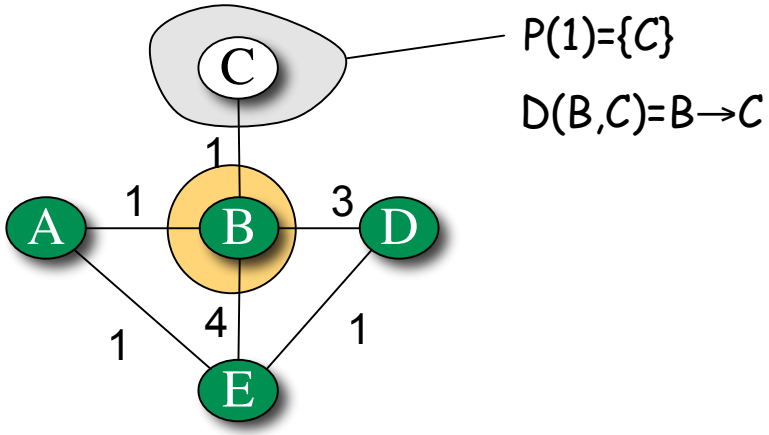
- Calcula los caminos de menor coste desde un nodos a todos los demás

c(i,j) : coste de enlace (i,j) (ej. c(B,D)=3)

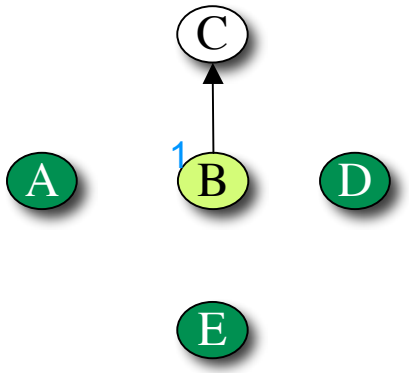
D(i,j) : camino más corto de i a j

L(i,j) : coste del camino D(i,j)

P(k) : conjunto de nodos procesaos hasta la k'esima iteración



- Origen nodo C
- Partiendo de P(k) pasar a P(k+1)
- Escoger nodo X:
  - No en P(k)
  - Conectado a uno de Y∈P(k)
  - El menor coste c(X,Y)+L(Y,C) (…)
  - Está conectado a nodo Y∈P(k)
- Actualizar (…):
  - D(X,C)=D(X,Y) ∪ D(Y,C)



Algoritmo de Dijkstra

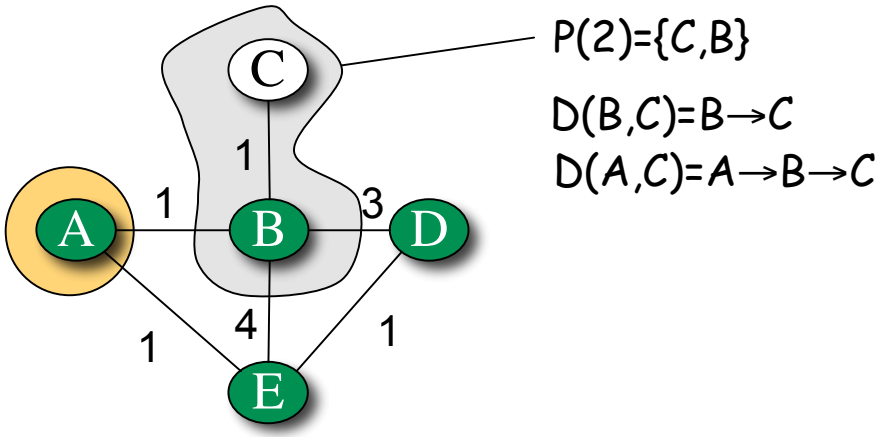
- Calcula los caminos de menor coste desde un nodos a todos los demás

c(i,j) : coste de enlace (i,j) (ej. c(B,D)=3)

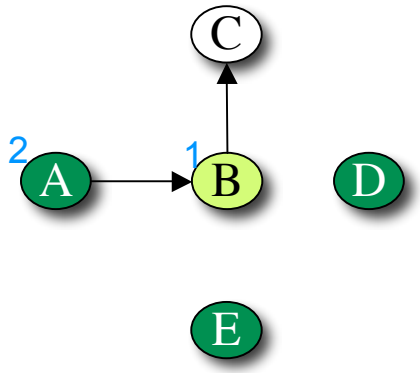
D(i,j) : camino más corto de i a j

L(i,j) : coste del camino D(i,j)

P(k) : conjunto de nodos procesaos hasta la k'esima iteración



- Origen nodo C
- Partiendo de P(k) pasar a P(k+1)
- Escoger nodo X:
  - No en P(k)
  - Conectado a uno de Y∈P(k)
  - El menor coste c(X,Y)+L(Y,C) (…)
  - Está conectado a nodo Y∈P(k)
- Actualizar (…):
  - D(X,C)=D(X,Y) ∪ D(Y,C)



Algoritmo de Dijkstra

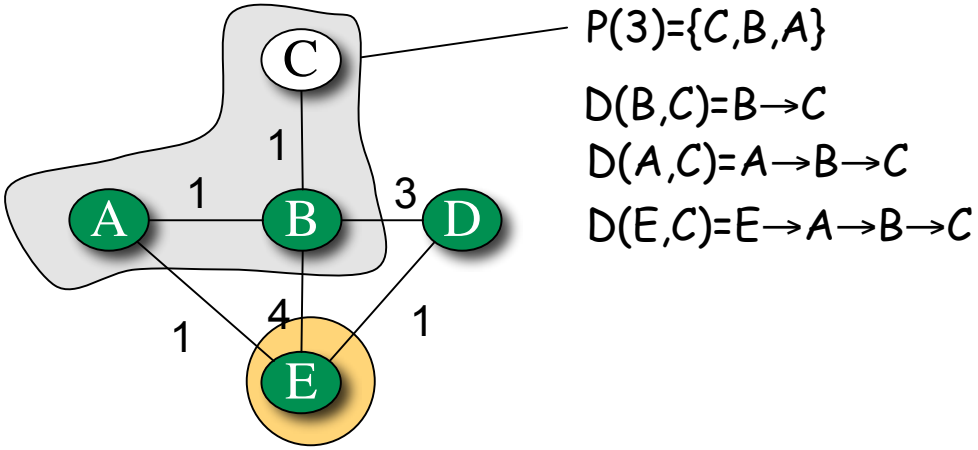
- Calcula los caminos de menor coste desde un nodos a todos los demás

c(i,j) : coste de enlace (i,j) (ej. c(B,D)=3)

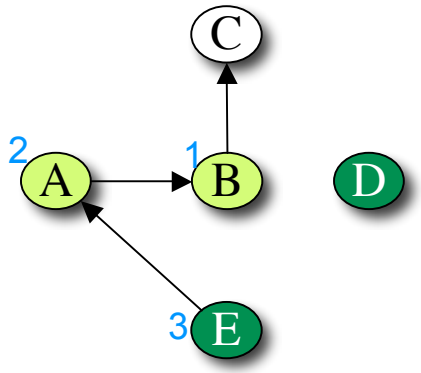
D(i,j) : camino más corto de i a j

L(i,j) : coste del camino D(i,j)

P(k) : conjunto de nodos procesaos hasta la k'esima iteración



- Origen nodo C
- Partiendo de P(k) pasar a P(k+1)
- Escoger nodo X:
  - No en P(k)
  - Conectado a uno de Y∈P(k)
  - El menor coste c(X,Y)+L(Y,C) (…)
  - Está conectado a nodo Y∈P(k)
- Actualizar (…):
  - D(X,C)=D(X,Y) ∪ D(Y,C)



Algoritmo de Dijkstra

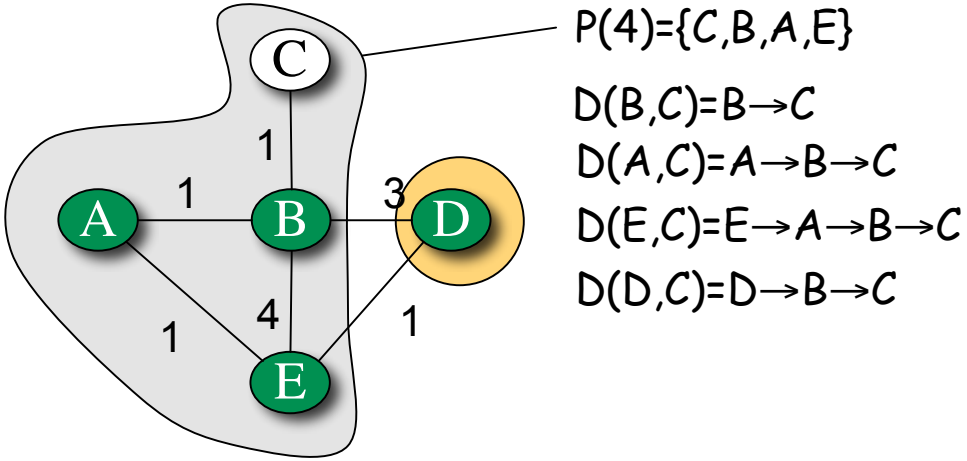
- Calcula los caminos de menor coste desde un nodos a todos los demás

c(i,j) : coste de enlace (i,j) (ej. c(B,D)=3)

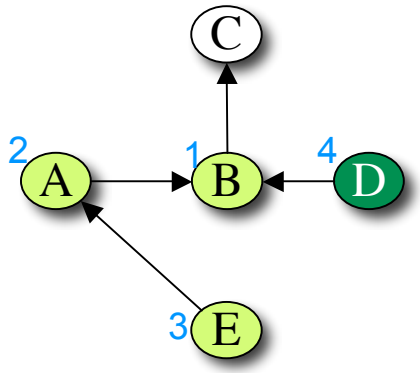
D(i,j) : camino más corto de i a j

L(i,j) : coste del camino D(i,j)

P(k) : conjunto de nodos procesaos hasta la k'esima iteración



- Origen nodo C
- Partiendo de P(k) pasar a P(k+1)
- Escoger nodo X:
  - No en P(k)
  - Conectado a uno de Y∈P(k)
  - El menor coste c(X,Y)+L(Y,C) (…)
  - Está conectado a nodo Y∈P(k)
- Actualizar (…):
  - D(X,C)=D(X,Y) ∪ D(Y,C)



Algoritmo de Dijkstra

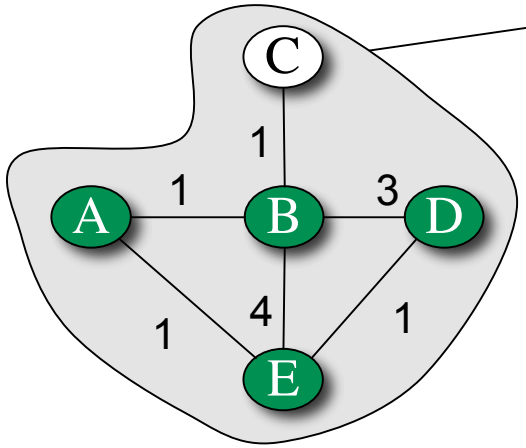
- Calcula los caminos de menor coste desde un nodos a todos los demás

c(i,j) : coste de enlace (i,j) (ej. c(B,D)=3)

D(i,j) : camino más corto de i a j

L(i,j) : coste del camino D(i,j)

P(k) : conjunto de nodos procesaos hasta la k'esima iteración



P(5)={C,B,A,E,D}

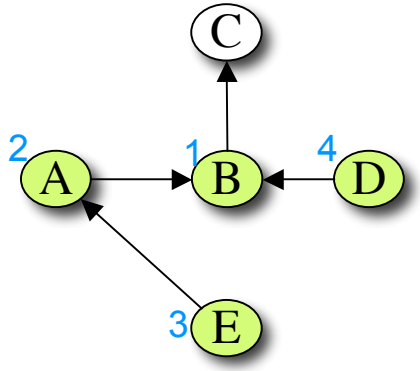
D(B,C)=B→C

D(A,C)=A→B→C

D(E,C)=E→A→B→C

D(D,C)=D→B→C

- Origen nodo C
- Partiendo de P(k) pasar a P(k+1)
- Escoger nodo X:
  - No en P(k)
  - Conectado a uno de Y∈P(k)
  - El menor coste c(X,Y)+L(Y,C) (…)
  - Está conectado a nodo Y∈P(k)
- Actualizar (…):
  - D(X,C)=D(X,Y) ∪ D(Y,C)



Link State

Algoritmo de Dijkstra

- Repetir para cada destino
- Da las tablas de rutas de todos los nodos
- Complejidad:
  - O(nlogn+m) (n nodos, m enlaces)
  - Caso peor $O(n^2)$
- Base de datos distribuida replicada

Algoritmo de distribución crítico

- Protocolos: OSPF, IS-IS, PNNI

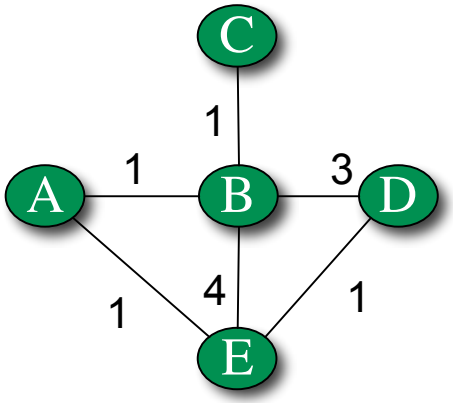
Distance Vector

- Cada nodo llega a conocer la distancia desde él a todos los destinos
  - $D(X,d_i)$
- Inicialmente cada nodo solo conoce la distancia a sus vecinos
  - $D(X,d)=c(X,d)$
- Periódicamente comunica D(X,d) a todos sus vecinos
  - Informan con un ***vector*** con las ***distancias*** a los destinos

    ($D(X,d_1)$, $D(X,d_2)$, $D(X,d_3)$, $D(X,d_4)$…)
  - Asíncrono
- Al recibir información actualiza:
  - $D(X,d) \leftarrow \text{mín}_{j/c(X,j)<\infty}\{c(X,j)+D(j,d)\}$
- Algoritmo de ***Bellman-Ford*** distribuido
- Empleado desde los comienzos de la ARPANET

Algoritmo de Bellman-Ford

- Comienzo



A

Dest	Next	Cost
B	B	1
C	-	∞
D	-	∞
E	E	1

B

Dest	Next	Cost
A	A	1
C	C	1
D	D	3
E	E	4

C

Dest	Next	Cost
A	-	∞
B	B	1
D	-	∞
E	-	∞

D

Dest	Next	Cost
A	-	∞
B	B	3
C	-	∞
E	E	1

E

Dest	Next	Cost
A	A	1
B	B	4
C	-	∞
D	D	1

Algoritmo de Bellman-Ford

A envía

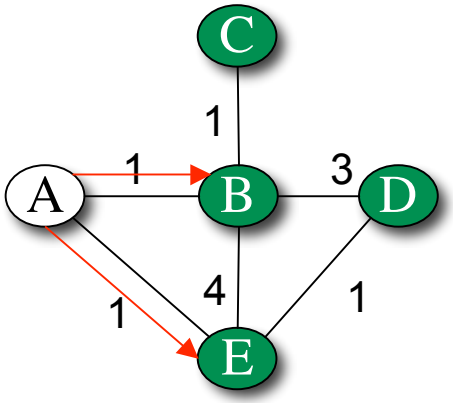
D(E,d)←mín{c(E,A)+D(A,d)}
D(B,d)←mín{c(B,A)+D(A,d)}
(…)

Dest	Next	Cost
B	B	1
C	-	∞
D	-	∞
E	E	1

A

Dest	Next	Cost
A	A	1
C	C	1
D	D	3
E	E	4

B



Dest	Next	Cost
A	-	∞
B	B	1
D	-	∞
E	-	∞

C

Dest	Next	Cost
A	-	∞
B	B	3
C	-	∞
E	E	1

D

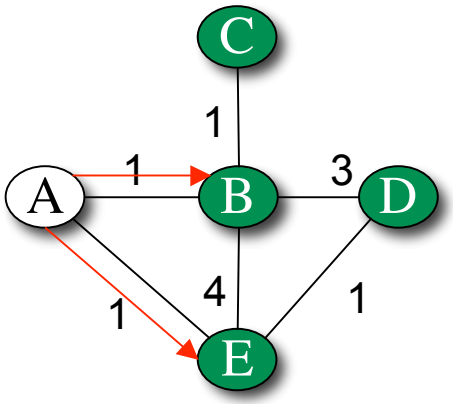
Dest	Next	Cost
A	A	1
B	B	4
C	-	∞
D	D	1

E

Algoritmo de Bellman-Ford

A envía

D(E,d)←mín{c(E,A)+D(A,d)}
D(B,d)←mín{c(B,A)+D(A,d)}



A

Dest	Next	Cost
B	B	1
C	-	∞
D	-	∞
E	E	1

B

Dest	Next	Cost
A	A	1
C	C	1
D	D	3
E	**A** (E)	**2** (4)

C

Dest	Next	Cost
A	-	∞
B	B	1
D	-	∞
E	-	∞

D

Dest	Next	Cost
A	-	∞
B	B	3
C	-	∞
E	E	1

E

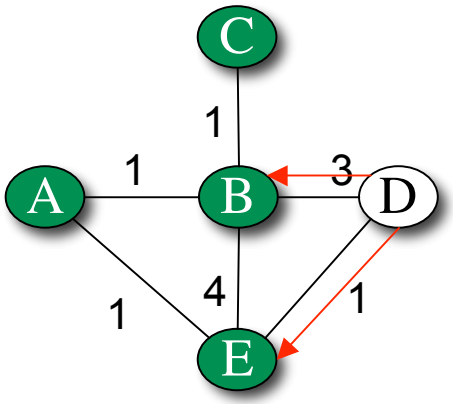
Dest	Next	Cost
A	A	1
B	**A** (B)	**2** (4)
C	-	∞
D	D	1

Algoritmo de Bellman-Ford

D envía

D(E,d)←mín{c(E,D)+D(D,d)}
D(B,d)←mín{c(B,D)+D(D,d)}

No hay cambios



A

Dest	Next	Cost
B	B	1
C	-	∞
D	-	∞
E	E	1

B

Dest	Next	Cost
A	A	1
C	C	1
D	D	3
E	A	2

C

Dest	Next	Cost
A	-	∞
B	B	1
D	-	∞
E	-	∞

D

Dest	Next	Cost
A	-	∞
B	B	3
C	-	∞
E	E	1

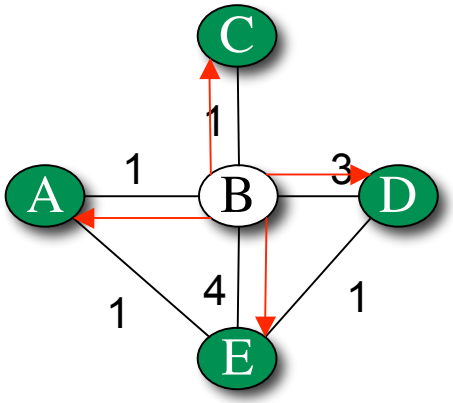
E

Dest	Next	Cost
A	A	1
B	A	2
C	-	∞
D	D	1

Algoritmo de Bellman-Ford

B envía

D(A,d)←mín{c(A,B)+D(B,d)}
D(C,d)←mín{c(C,B)+D(B,d)}
D(D,d)←mín{c(D,B)+D(B,d)}
D(E,d)←mín{c(E,B)+D(B,d)}
(…)



A

Dest	Next	Cost
B	B	1
C	-	∞
D	-	∞
E	E	1

B

Dest	Next	Cost
A	A	1
C	C	1
D	D	3
E	A	2

C

Dest	Next	Cost
A	-	∞
B	B	1
D	-	∞
E	-	∞

D

Dest	Next	Cost
A	-	∞
B	B	3
C	-	∞
E	E	1

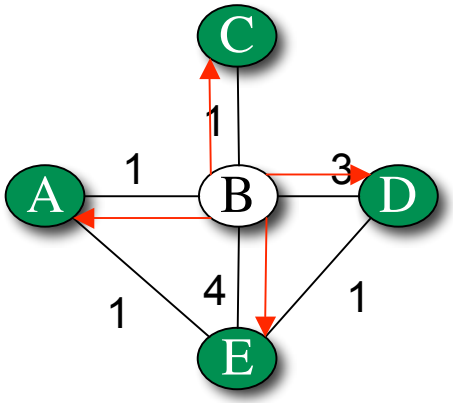
E

Dest	Next	Cost
A	A	1
B	A	2
C	-	∞
D	D	1

Algoritmo de Bellman-Ford

B envía

D(A,d)←mín{c(A,B)+D(B,d)}
D(C,d)←mín{c(C,B)+D(B,d)}
D(D,d)←mín{c(D,B)+D(B,d)}
D(E,d)←mín{c(E,B)+D(B,d)}



A

Dest	Next	Cost
B	B	1
C	B (-)	2 (∞)
D	B (-)	4 (∞)
E	E	1

B

Dest	Next	Cost
A	A	1
C	C	1
D	D	3
E	A	2

C

Dest	Next	Cost
A	B (-)	2 (∞)
B	B	1
D	B (-)	4 (∞)
E	B (-)	3 (∞)

D

Dest	Next	Cost
A	B (-)	4 (∞)
B	B	3
C	B (-)	4 (∞)
E	E	1

E

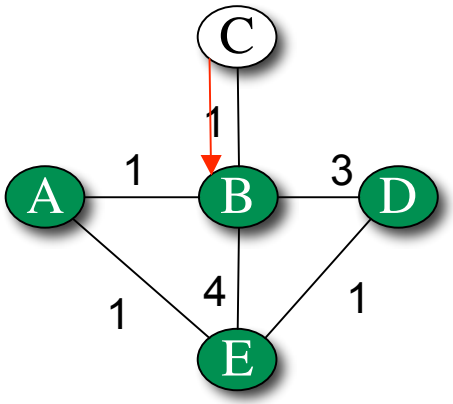
Dest	Next	Cost
A	A	1
B	A	2
C	B (-)	5 (∞)
D	D	1

Algoritmo de Bellman-Ford

C envía

D(B,d)←mín{c(B,C)+D(C,d)}

No hay cambios



A

Dest	Next	Cost
B	B	1
C	B	2
D	B	4
E	E	1

B

Dest	Next	Cost
A	A	1
C	C	1
D	D	3
E	A	2

C

Dest	Next	Cost
A	B	2
B	B	1
D	B	4
E	B	3

D

Dest	Next	Cost
A	B	4
B	B	3
C	B	4
E	E	1

E

Dest	Next	Cost
A	A	1
B	A	2
C	B	5
D	D	1

Algoritmo de Bellman-Ford

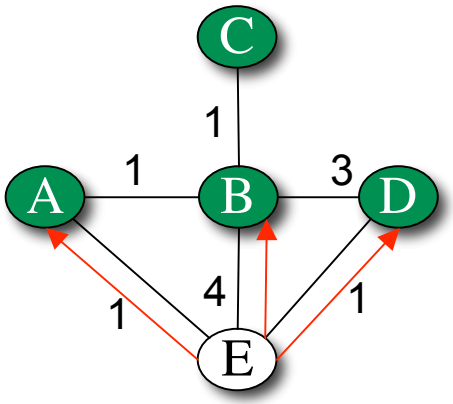
E envía

D(A,d)←mín{c(A,E)+D(E,d)}

D(B,d)←mín{c(B,E)+D(E,d)}

D(D,d)←mín{c(D,E)+D(E,d)}

(…)



A

Dest	Next	Cost
B	B	1
C	B	2
D	B	4
E	E	1

B

Dest	Next	Cost
A	A	1
C	C	1
D	D	3
E	A	2

C

Dest	Next	Cost
A	B	2
B	B	1
D	B	4
E	B	3

D

Dest	Next	Cost
A	B	4
B	B	3
C	B	4
E	E	1

E

Dest	Next	Cost
A	A	1
B	A	2
C	B	5
D	D	1

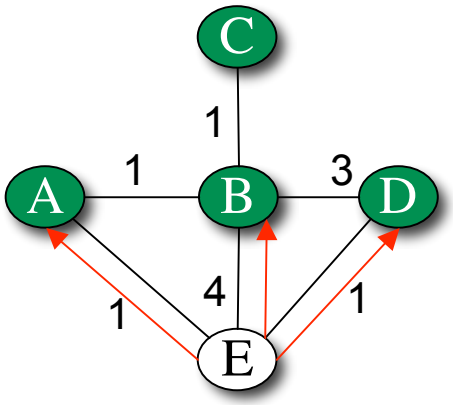
Algoritmo de Bellman-Ford

E envía

$D(A,d) \leftarrow \text{mín}\{c(A,E)+D(E,d)\}$

$D(B,d) \leftarrow \text{mín}\{c(B,E)+D(E,d)\}$

$D(D,d) \leftarrow \text{mín}\{c(D,E)+D(E,d)\}$



A

Dest	Next	Cost
B	B	1
C	B	2
D	**E** (B)	**2** (4)
E	E	1

B

Dest	Next	Cost
A	A	1
C	C	1
D	D	3
E	A	2

C

Dest	Next	Cost
A	B	2
B	B	1
D	B	4
E	B	3

D

Dest	Next	Cost
A	**E** (B)	**2** (4)
B	B	3
C	B	4
E	E	1

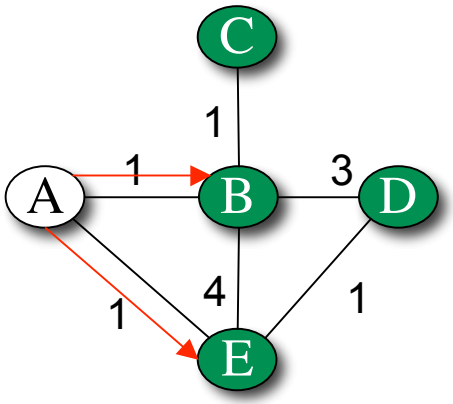
E

Dest	Next	Cost
A	A	1
B	A	2
C	B	5
D	D	1

Algoritmo de Bellman-Ford

A envía

D(E,d)←mín{c(E,A)+D(A,d)}
D(B,d)←mín{c(B,A)+D(A,d)}
(…)



A

Dest	Next	Cost
B	B	1
C	B	2
D	E	2
E	E	1

B

Dest	Next	Cost
A	A	1
C	C	1
D	D	3
E	A	2

C

Dest	Next	Cost
A	B	2
B	B	1
D	B	4
E	B	3

D

Dest	Next	Cost
A	E	2
B	B	3
C	B	4
E	E	1

E

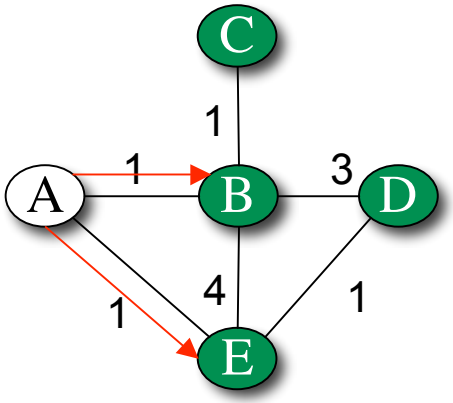
Dest	Next	Cost
A	A	1
B	A	2
C	B	5
D	D	1

Algoritmo de Bellman-Ford

A envía

D(E,d)←mín{c(E,A)+D(A,d)}

D(B,d)←mín{c(B,A)+D(A,d)}



A

Dest	Next	Cost
B	B	1
C	B	2
D	E	2
E	E	1

B

Dest	Next	Cost
A	A	1
C	C	1
D	D	3
E	A	2

C

Dest	Next	Cost
A	B	2
B	B	1
D	B	4
E	B	3

D

Dest	Next	Cost
A	E	2
B	B	3
C	B	4
E	E	1

E

Dest	Next	Cost
A	A	1
B	A	2
C	**A** (B)	**3** (5)
D	D	1

Algoritmo de Bellman-Ford

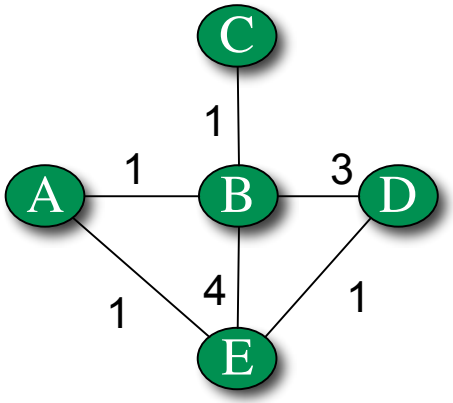
D envía
- No hay cambios

B envía
- No hay cambios

C envía
- No hay cambios

E envía
- No hay cambios

A envía
- No hay cambios



A

Dest	Next	Cost
B	B	1
C	B	2
D	E	2
E	E	1

B

Dest	Next	Cost
A	A	1
C	C	1
D	D	3
E	A	2

C

Dest	Next	Cost
A	B	2
B	B	1
D	B	4
E	B	3

D

Dest	Next	Cost
A	E	2
B	B	3
C	B	4
E	E	1

E

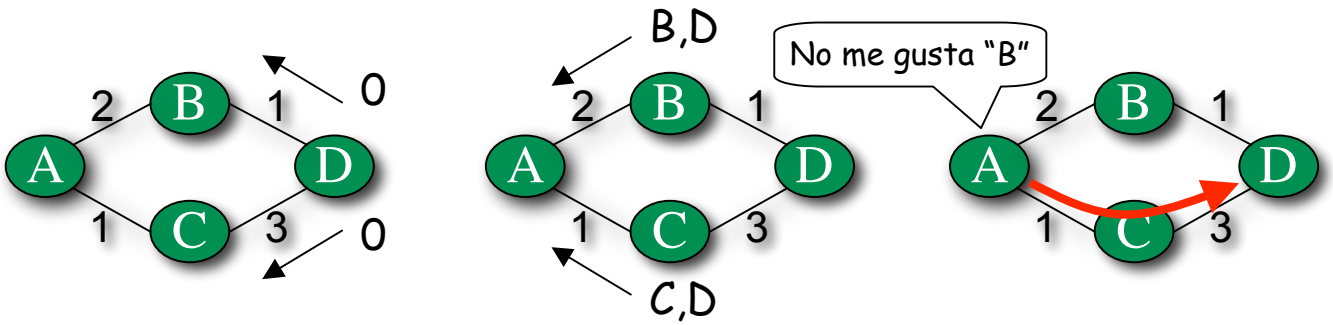
Dest	Next	Cost
A	A	1
B	A	2
C	A	3
D	D	1

Distance Vector

- Cálculo distribuido
- Iterativo e incremental
- Asíncrono
- Converge a los caminos de menor coste
- Protocolos: RIP, IPX-RIP, DECnet, IGRP, EIGRP, DSDV

Path Vector

- Similar a Distance Vector
- Cálculo distribuido
- Informan a sus vecinos de las rutas calculadas
- Incluyen todo el camino hasta el destino para cada ruta
- Protocolos: BGP



Contenido

- Internet Protocol
  - Características
  - IP en LAN Ethernet (ARP)
  - Direccionamiento (subredes, Proxy-ARP, CIDR)
  - Fragmentación y reensamblado
  - ICMP
- Enrutamiento en Internet
  - Distance-Vector
  - Link-State
  - Path-Vector
- **UDP**
- TCP
  - Características
  - Gestión de conexiones
  - Control de flujo
  - Retransmisiones
  - Control de congestión

Lecturas recomendadas

– [Forouzan03] 11.1-11.4

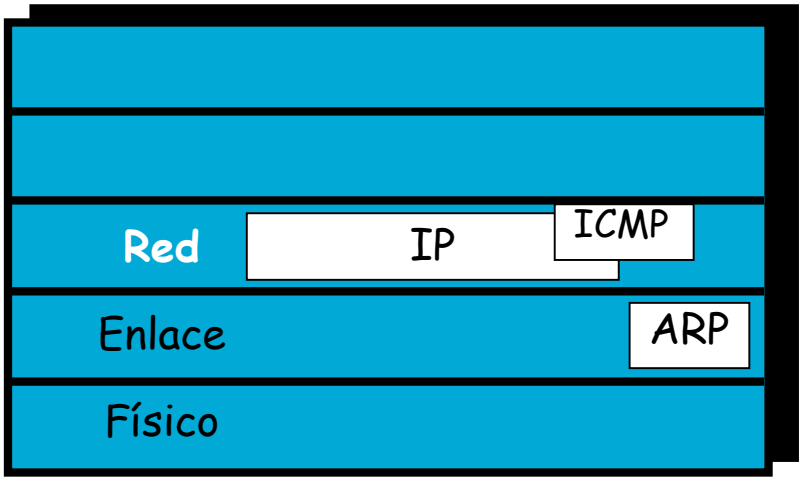
Contenido

- Introducción
- Nivel de transporte
- UDP
  - Características
  - Formato
  - Demultiplexación
- Errores ICMP asociados

Nivel de red

IP

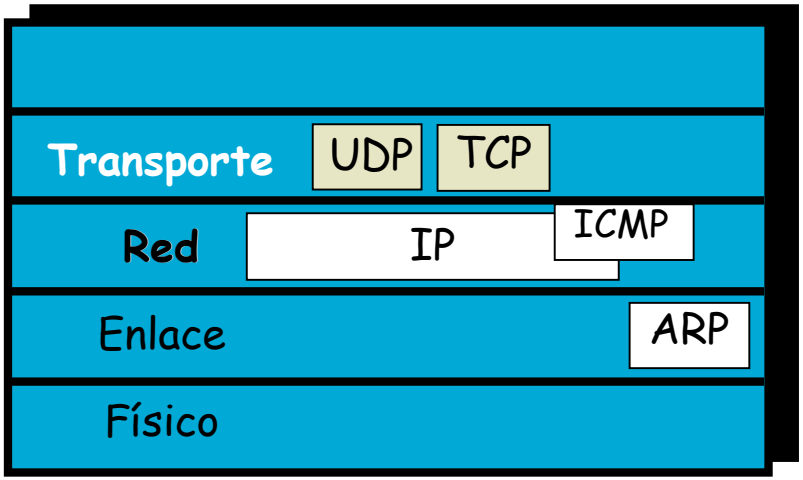
- Ofrece un servicio best-effort
- Los paquetes se pueden retrasar, perder, desordenar, duplicar, etc.
- Van dirigidos a un host, pero ¿a qué aplicación?
- ¿Cómo debería mandar el host?
  - Demasiado rápido: congestión
  - Demasiado lento: ineficiente



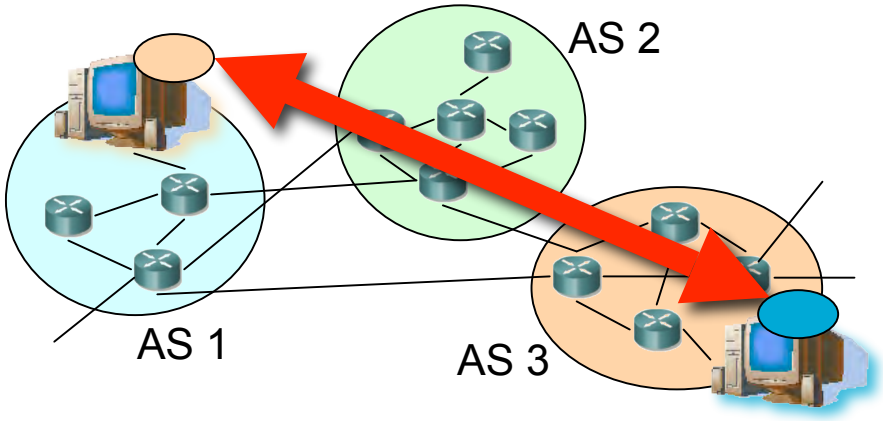
Nivel de transporte

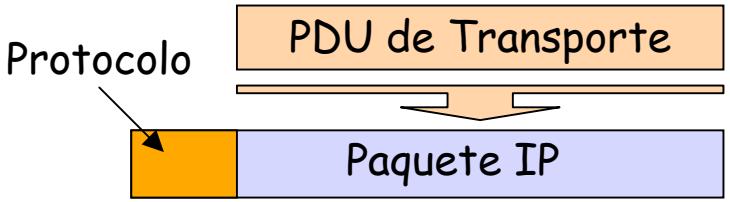
Nivel de transporte (…)

- *Comunicación lógica* extremo a extremo entre procesos (…)
- Puede ofrecer fiabilidad, orden
- Mensajes de mayor tamaño:
  - Emisor segmenta
  - Receptor reensambla
- Inteligencia en los extremos



- TCP/IP ofrece 2 protocolos (…)
- Emplean los servicios del nivel de red (…)
- PDU del nivel de transporte: segmento





Multiplexación/Demultiplexación

Multiplexación en emisor

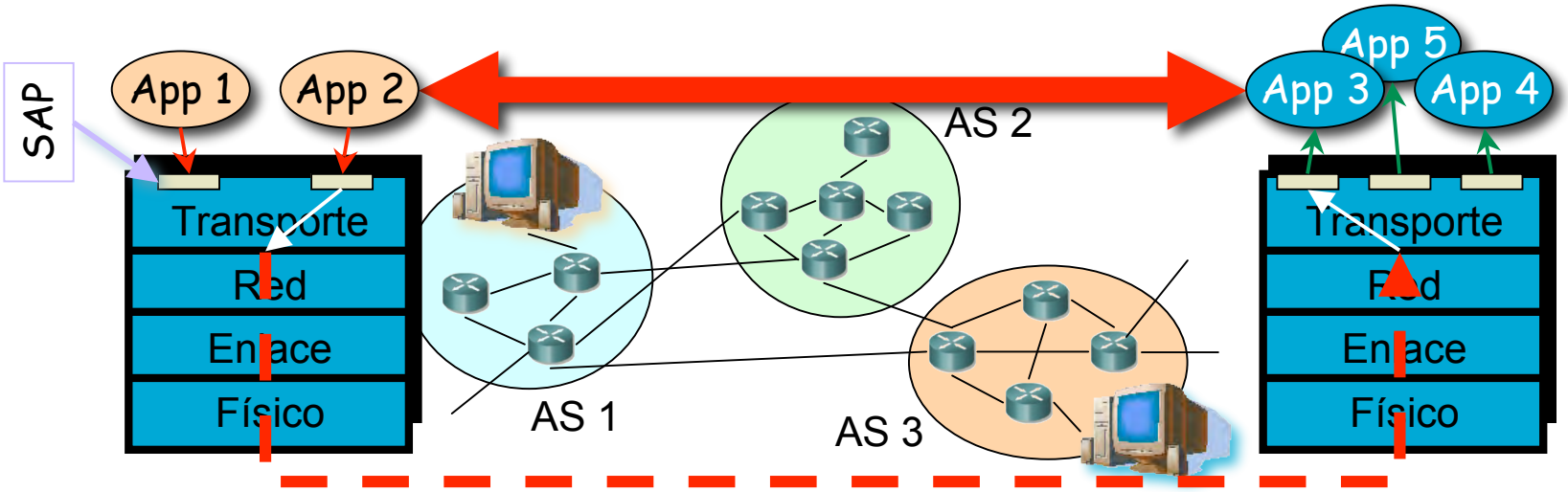
- Recoger datos de varias aplicaciones
- Añadir cabecera de transporte
- Incluye un identificador de la aplicación origen y la detino (puerto)

Enrutamiento

- Hace llegar los paquetes al host (dirección IP) correcto

Demultiplexación en receptor

- Cada datagrama IP lleva un segmento del nivel de transporte
- Según el puerto destino y tal vez mirando también el origen decide la aplicación destino



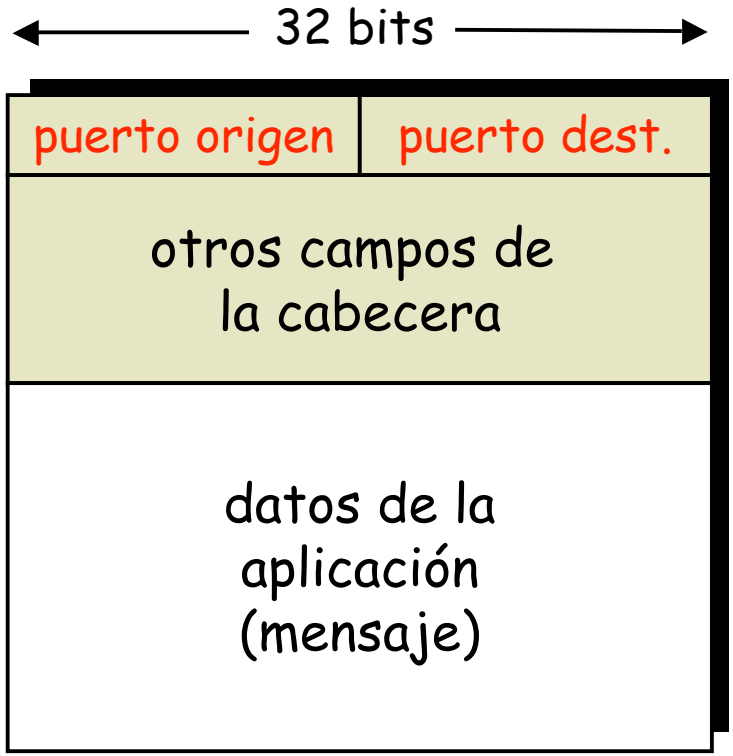
Formato de la PDU de transporte

- TDP o UDP
- **Puerto origen**
  - Identifica a la aplicación emisora en el host
- **Puerto destino**
  - Identifica a la aplicación receptora en el host
- En el sentido contrario irán al revés
- El emisor debe conocer el puerto del receptor
- Puertos

  [0,1023] *Well known*

  [1024,49151] *Registered*

  [49152,65535] Dinámicos, privados o *efímeros*

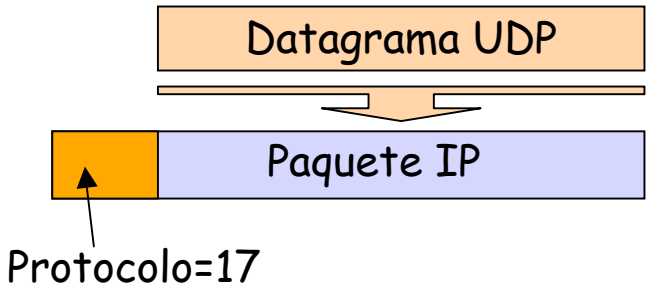


UDP: User Datagram Protocol

- RFC 768
- Protocolo de transporte **simple**, sin gran inteligencia
- Servicio “best effort”
- Datagramas
- Los datagramas UDP se pueden:
  - Perder
  - Llegar desordenados a la aplicación
- ¿Transferencia fiable sobre UDP?
  - Añadir fiabilidad en el nivel de aplicación
  - ¡Recuperación ante errores específica de cada aplicación!
- Sin conexión:
  - No hay handshaking entre emisor y receptor
  - Cada datagrama UDP es procesado de forma independiente a los demás
- Empleado frecuentemente para aplicaciones de streaming multimedia
  - Soportan pérdidas
  - Sensibles a la tasa de envío
- Otros usos de UDP:
  - DNS
  - SNMP

UDP: User Datagram Protocol

- ¿Por qué existe UDP?
  - Es simple: no hay que mantener estado
  - Un establecimiento de conexión añadiría retardo no deseado
  - Cabecera pequeña
  - No hay control de congestión: puede enviar tan rápido como desee
- Encapsulado en paquetes IP, protocolo 17
- Cuando un host recibe un datagrama UDP :
  - Comprueba el puerto destino en el mismo
  - Dirige el segmento a la aplicación esperando datos a ese puerto
- Diferentes IP origen o puertos origen van al mismo punto de acceso al servicio (SAP)



Cabecera UDP

Puerto origen

- Normalmente lo escoge el sistema operativo
- Suele ser un puerto efímero

Puerto destino

- Puerto del servidor
- *Well known* o se debe conocer por algún medio

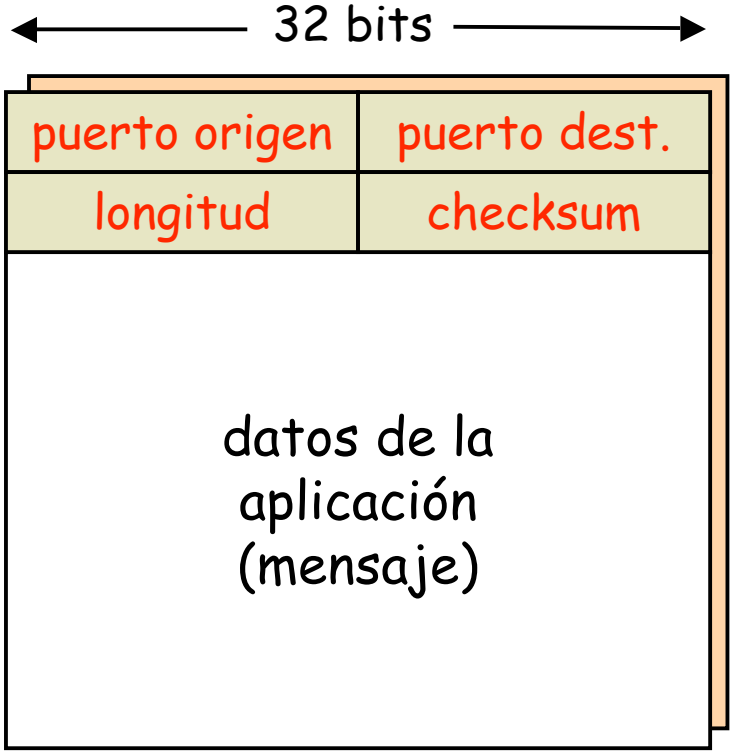
Respuesta servidor→cliente

- Sentido contrario
- Puerto origen es el del servidor (*well known*)
- Puerto destino el efímero del cliente

Longitud

- Bytes del datagrama UDP

Checksum (…)



Checksum UDP

Objetivo: detectar “errores” (ej., bits cambiados) en un datagrama

Cubre a la cabecera y los datos (y parte de la cabecera IP)

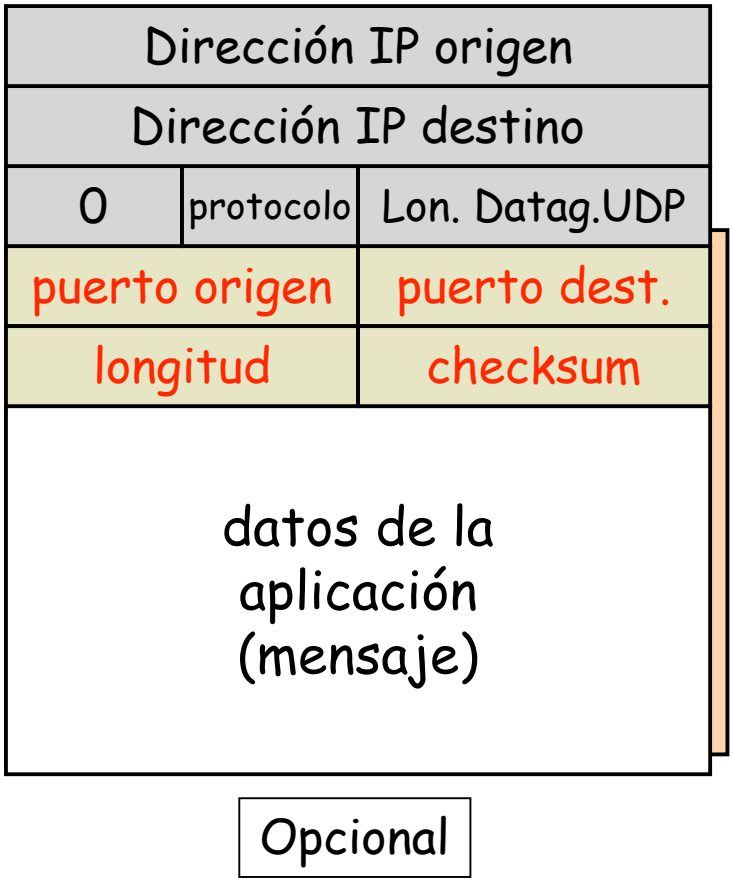
Emisor:

- Trata el datagrama como una secuencia de enteros de 16 bits
- Complemento a 1 de la suma (en complemento a 1) del datagrama y *pseudocabecera*
- Coloca el checksum en el campo

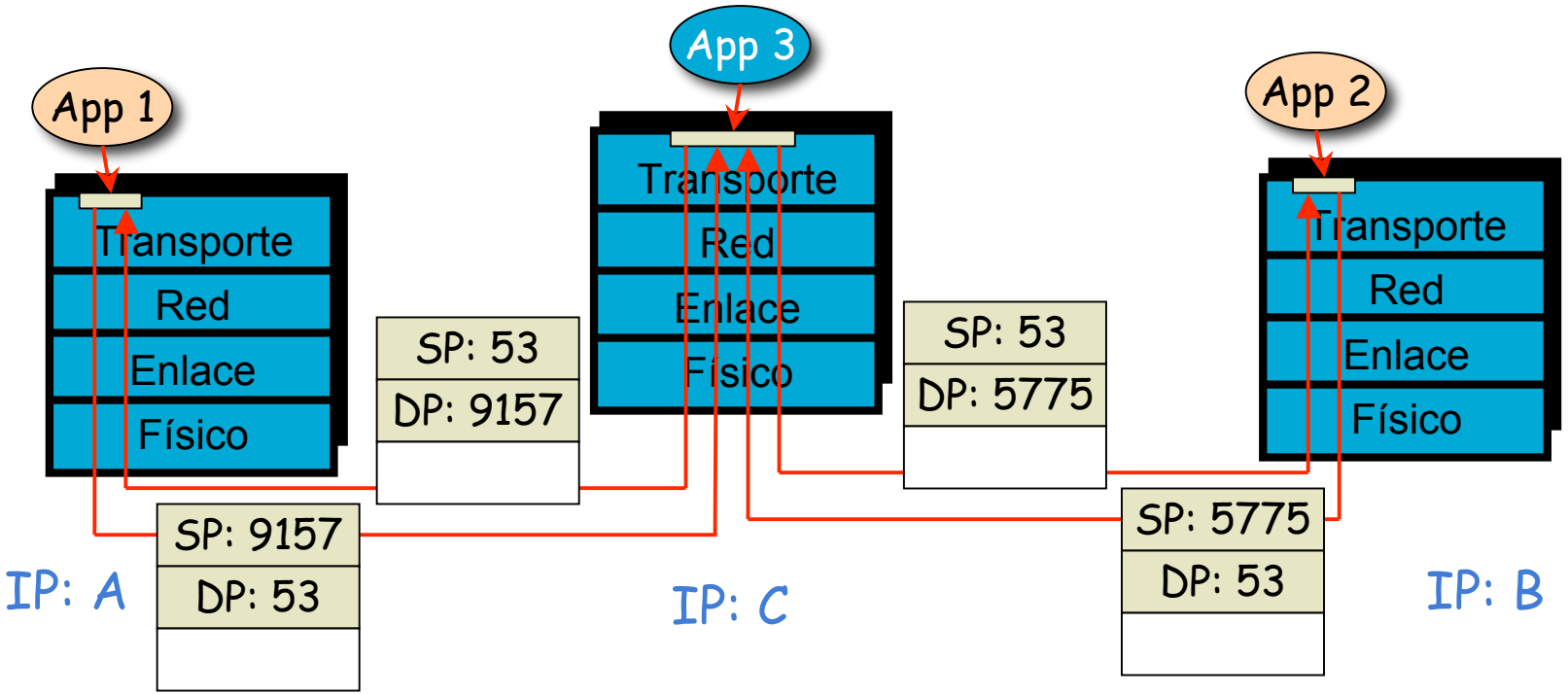
Receptor:

- Hace la suma en complemento a 1 de todo el datagrama
- ¿Da 0?
  - NO - error detectado
  - Sí - no hay errores detectados

    ¡Pero aún así puede haberlos!

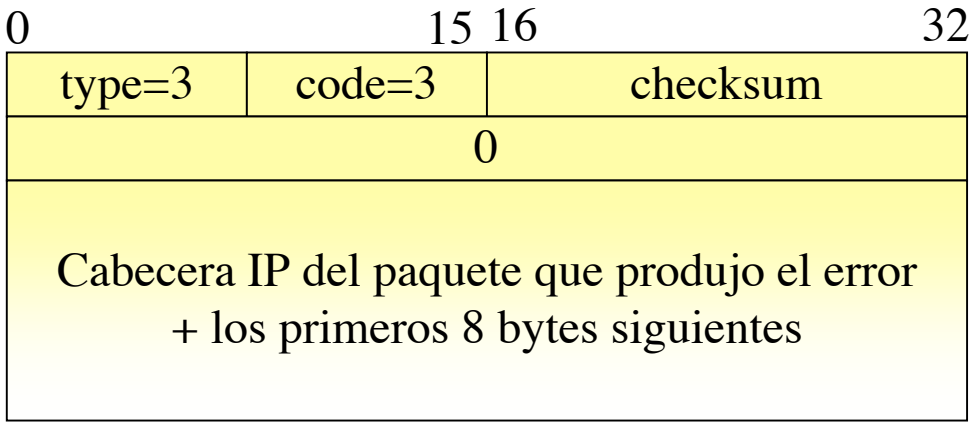


Demultiplexación: Ejemplo



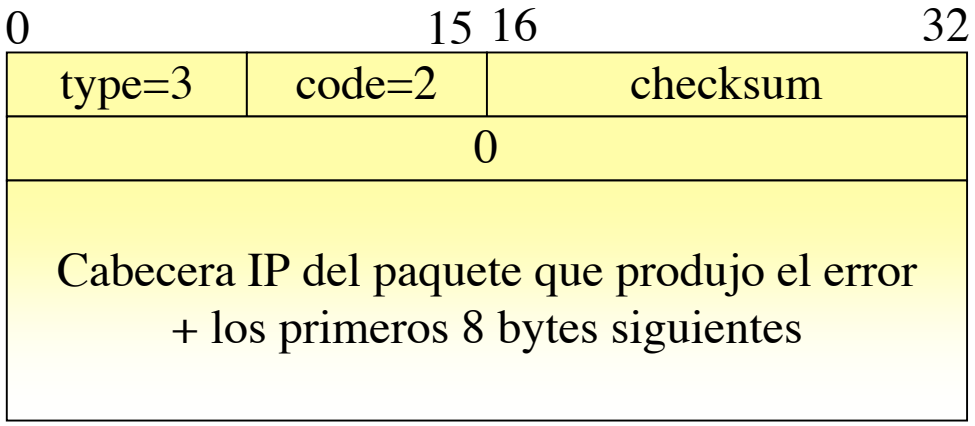
Mensajes ICMP

- Puerto destino inalcanzable (*destination port unreachable*)
  - Generado por un host que recibe un datagrama UDP para cuyo puerto destino no espera mensajes ninguna aplicación
  - tipo=3 (destino inalcanzable), código=3



Mensajes ICMP

- Protocolo inalcanzable
  - Generado cuando el host receptor del paquete IP no conoce el protocolo que viene indicado en la cabecera del mismo
  - tipo=3 (destino inalcanzable), código=2



Contenido

- Internet Protocol
  - Características
  - IP en LAN Ethernet (ARP)
  - Direccionamiento (subredes, Proxy-ARP, CIDR)
  - Fragmentación y reensamblado
  - ICMP
- Enrutamiento en Internet
  - Distance-Vector
  - Link-State
  - Path-Vector
- UDP
- TCP
  - **Características**
  - **Gestión de conexiones**
  - Control de flujo
  - Retransmisiones
  - Control de congestión

Lecturas recomendadas

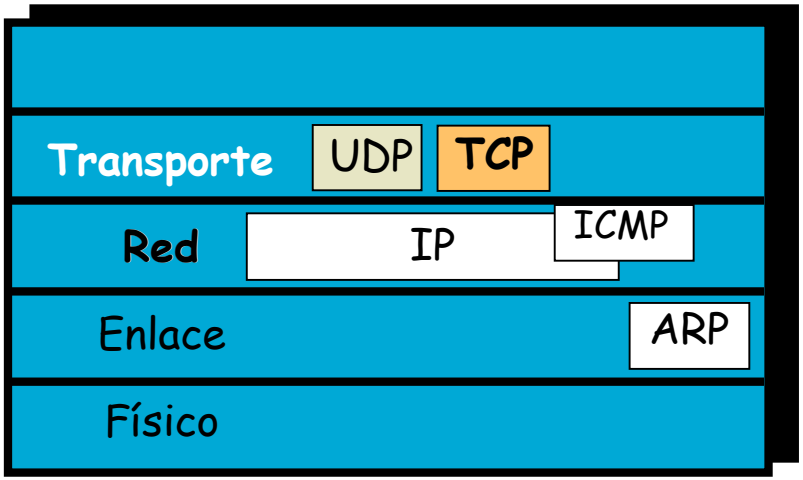
– [Stevens] 17-18

Contenido

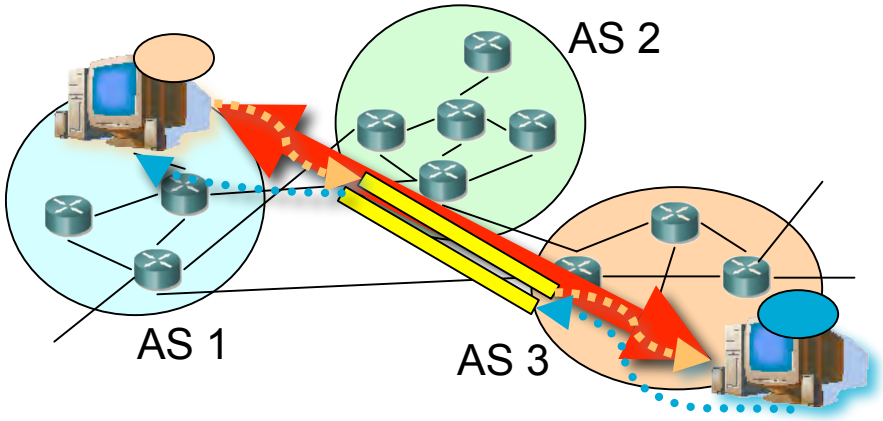
- Introducción
- Demultiplexación en TCP
- Gestión de conexiones

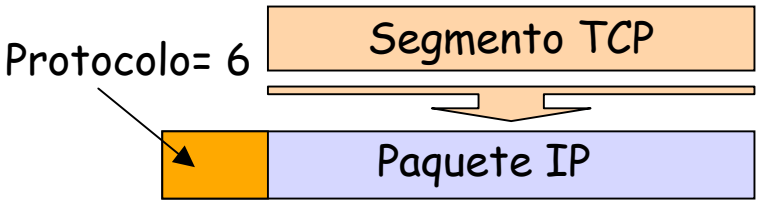
TCP

- *Transmission Control Protocol*
- Nivel de transporte
- RFCs 793, 1122, 1323, 2018, 2581
- Orientado a conexión
- Flujo de datos:
  - *Stream* de bytes
  - Fiable
  - Ordenado
  - Full duplex



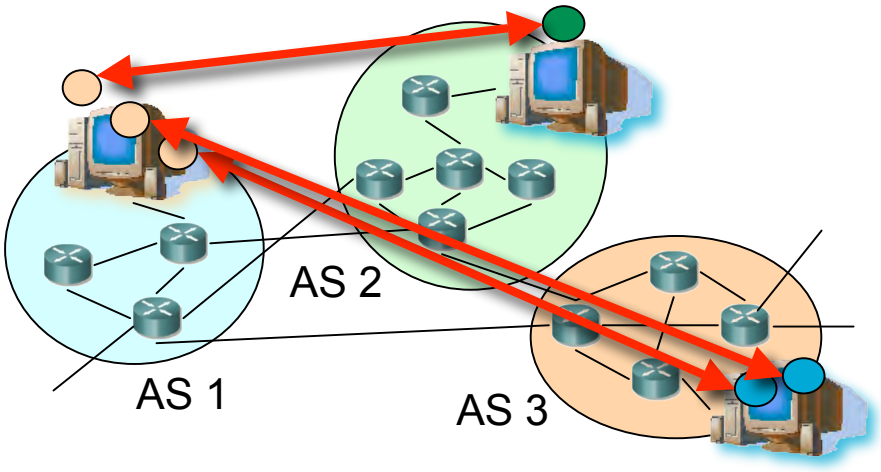
- Control de flujo
  - Evitar congestionar al receptor
- Control de congestión
  - Evitar congestionar la red



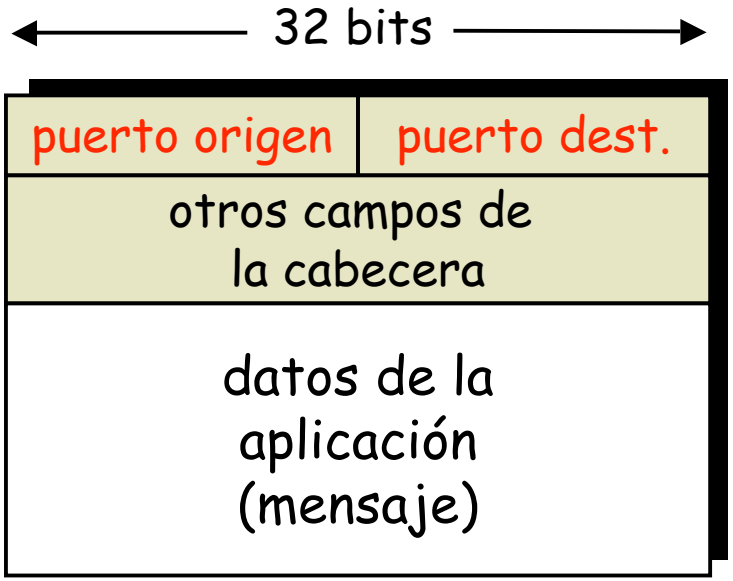


Demultiplexación con conexión

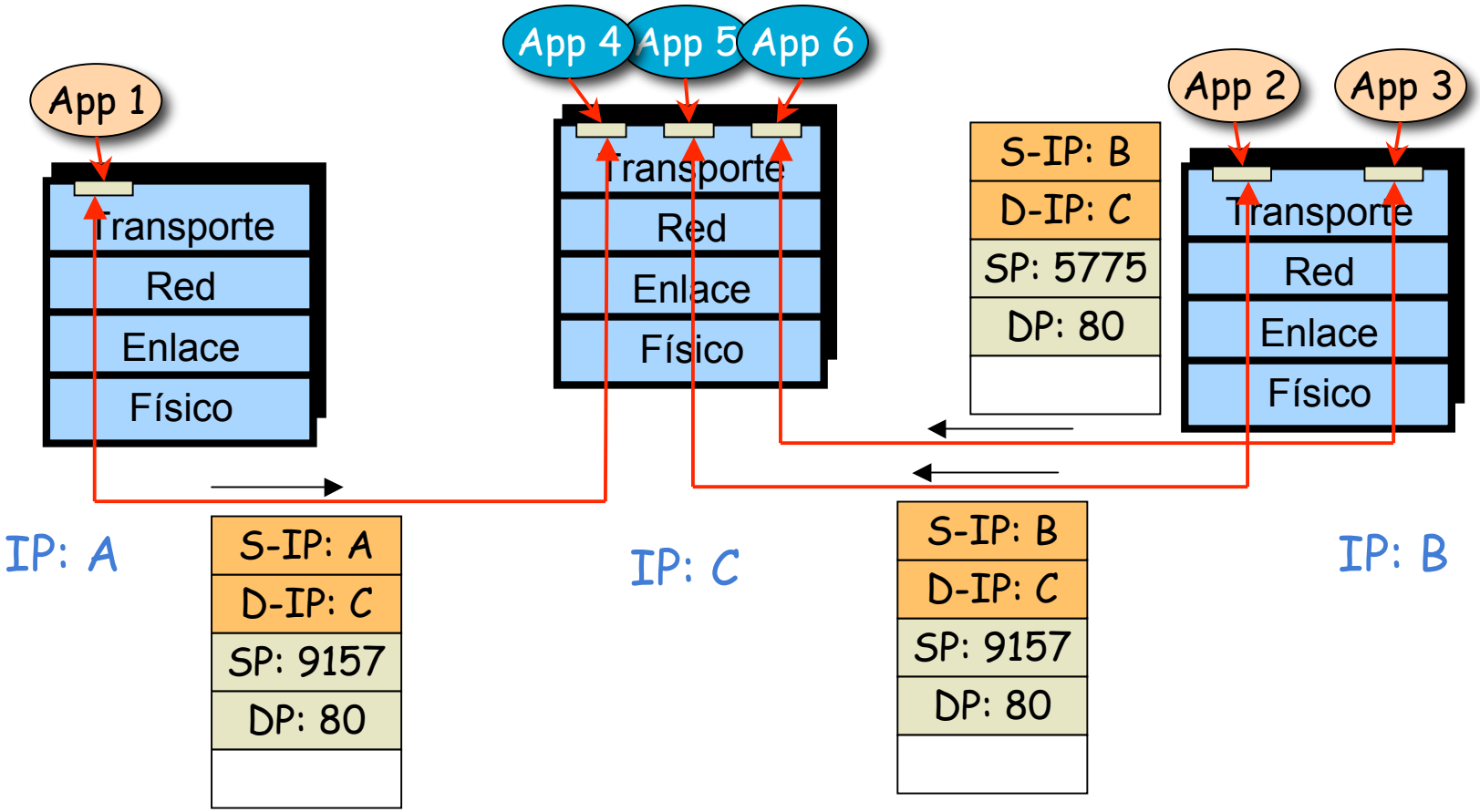
- Conexión identificada por 2 *sockets*
- Cada *socket* identificado por: Dirección IP y Puerto TCP
- Es decir, la conexión viene identificada por:
  - **Dirección IP (1), Puerto TCP (1)**
  - **Dirección IP (2), Puerto TCP (2)**
- El receptor emplea la cuaterna para demultiplexar



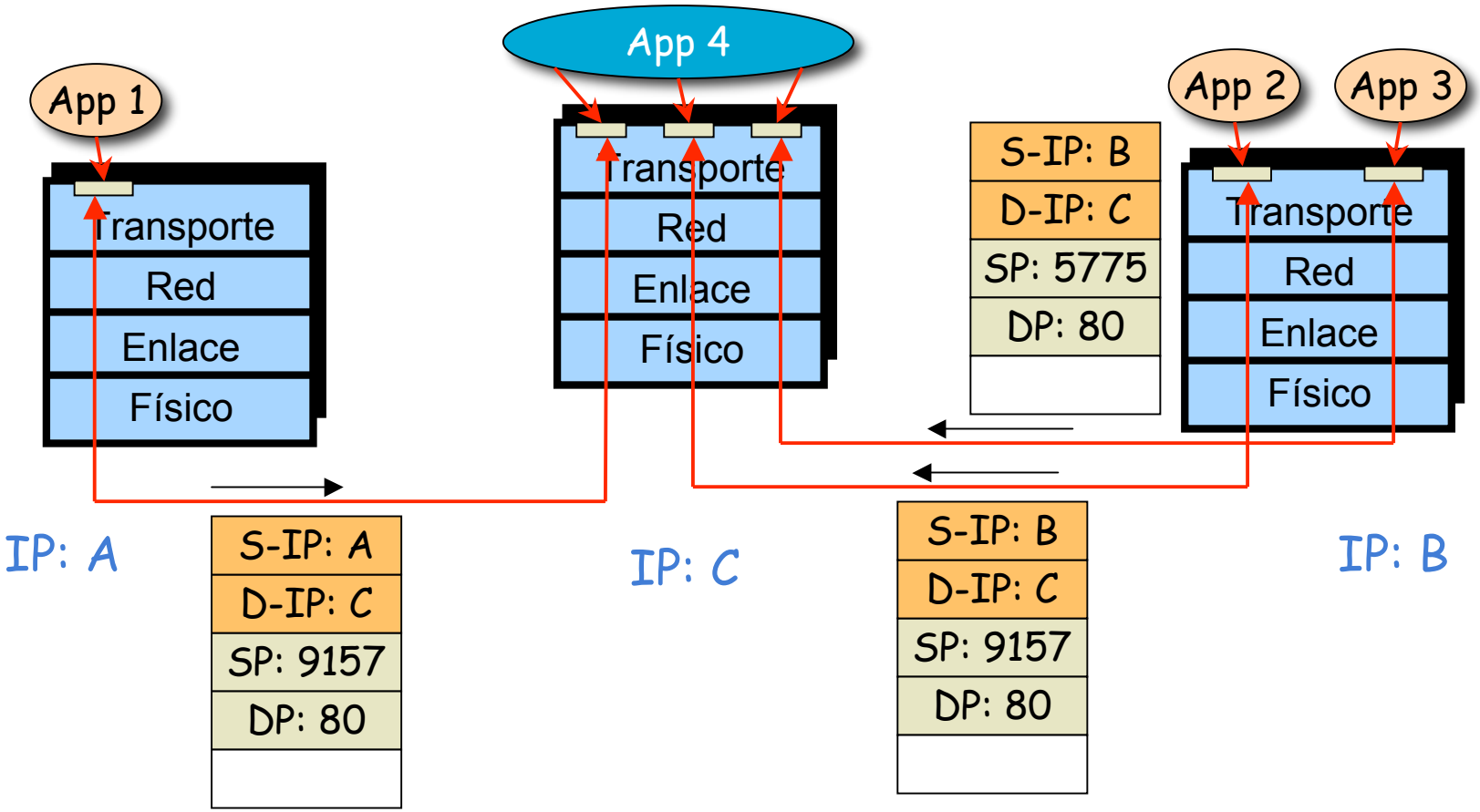
- Cada host soporta múltiples conexiones TCP simultáneas
- Con que uno de los 4 valores sea diferente la conexión ya es diferente
- *Well-known ports*, registrados, efímeros, igual que para UDP



Demultiplexación: Ejemplo



Demultiplexación: Ejemplo



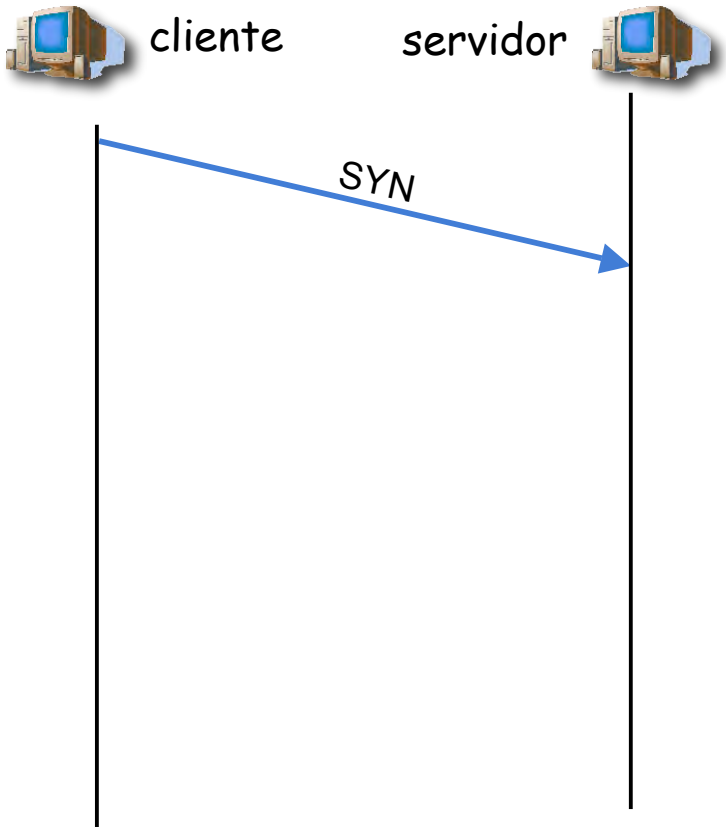
Gestión de conexiones

Estableciendo una conexión:

- *Three way handshake*

Paso 1:

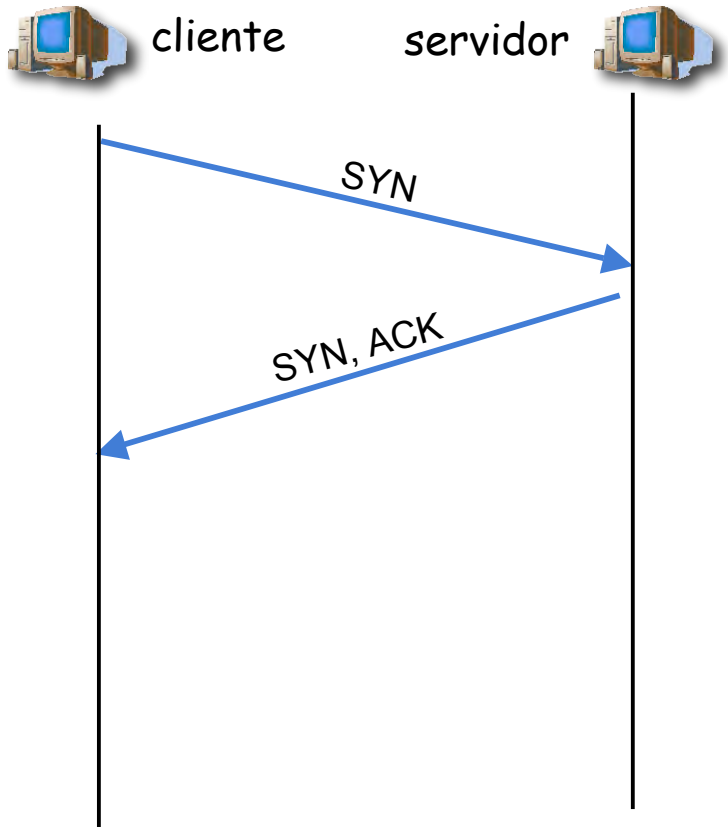
- El extremo **cliente** envía un segmento solicitando una conexión al servidor
- El segmento **no tiene datos**, solo cabecera
- **SYN**



Gestión de conexiones

Paso 2:

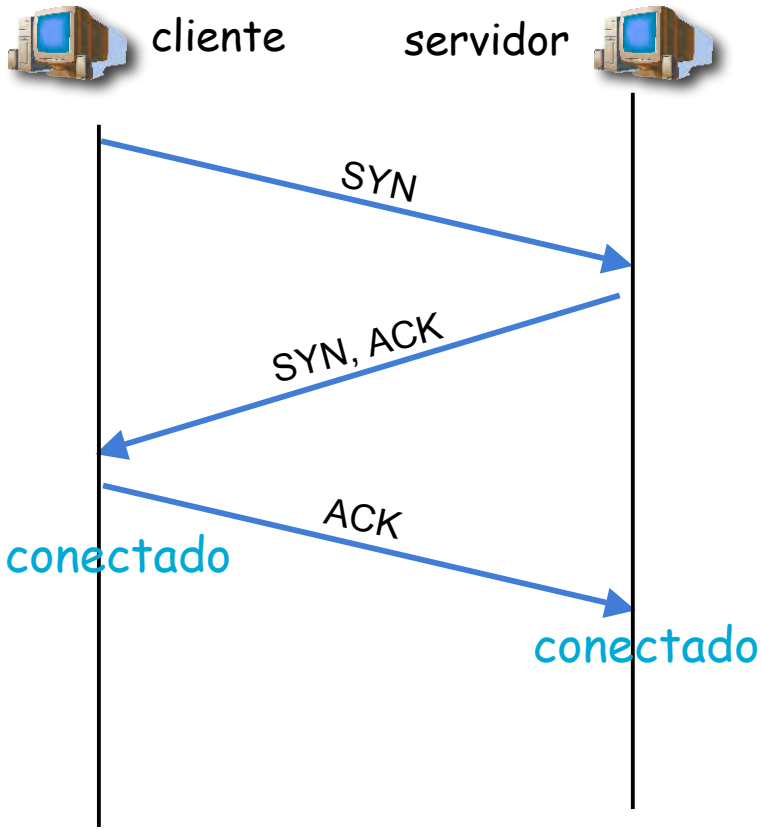
- El extremo **servidor** envía un segmento al cliente confirmando (acknowledgement) la recepción del SYN
- En el mismo segmento el servidor indica su deseo de establecer la conexión (SYN)
- El segmento **no tiene datos**, solo cabecera



Gestión de conexiones

Paso 3:

- El extremo **cliente** envía una confirmación al SYN del servidor
- El segmento **no tiene datos**, solo cabecera
- Conexión establecida



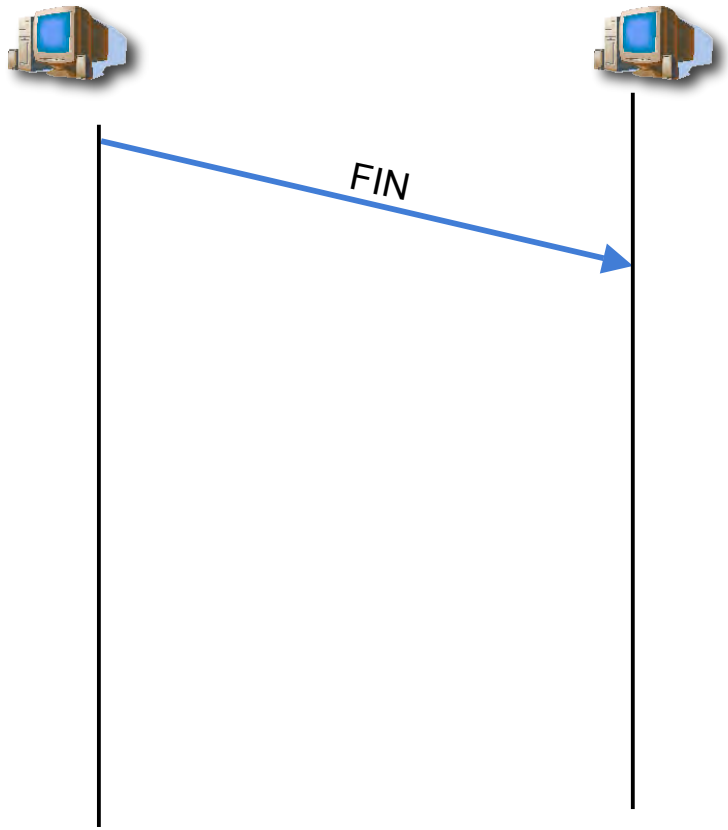
Transferencia de datos…

Gestión de conexiones

Cerrando una conexión

Paso 1:

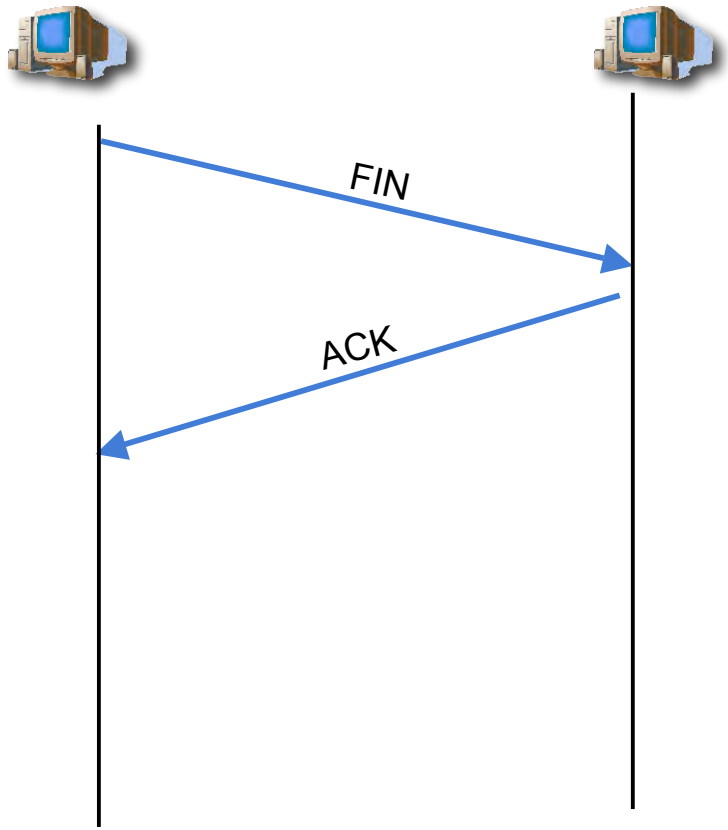
- **Un extremo** envía un segmento solicitando el cierre de la conexión
- El segmento **no tiene datos**, solo cabecera
- **FIN**



Gestión de conexiones

Paso 2:

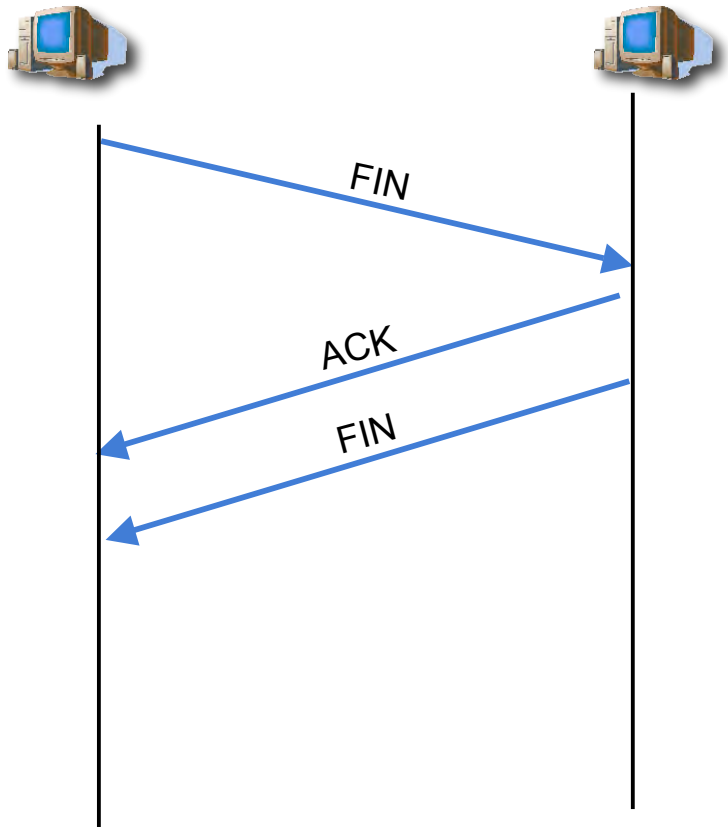
- El otro extremo confirma (ACK) la recepción del FIN
- El extremo que ha enviado el FIN ya no puede enviar más datos nuevos
- **Cierre solo de un sentido** de la comunicación



Gestión de conexiones

Paso 3:

- El otro extremo envía un segmento solicitando el cierre de la conexión
- El segmento no tiene datos, solo cabecera



Gestión de conexiones

Paso 4:

- Confirmación de ese segundo FIN
- Por si ese último ACK se pierde, el que lo envió espera un tiempo (podría tener que volverlo a enviar)
- Conexión cerrada

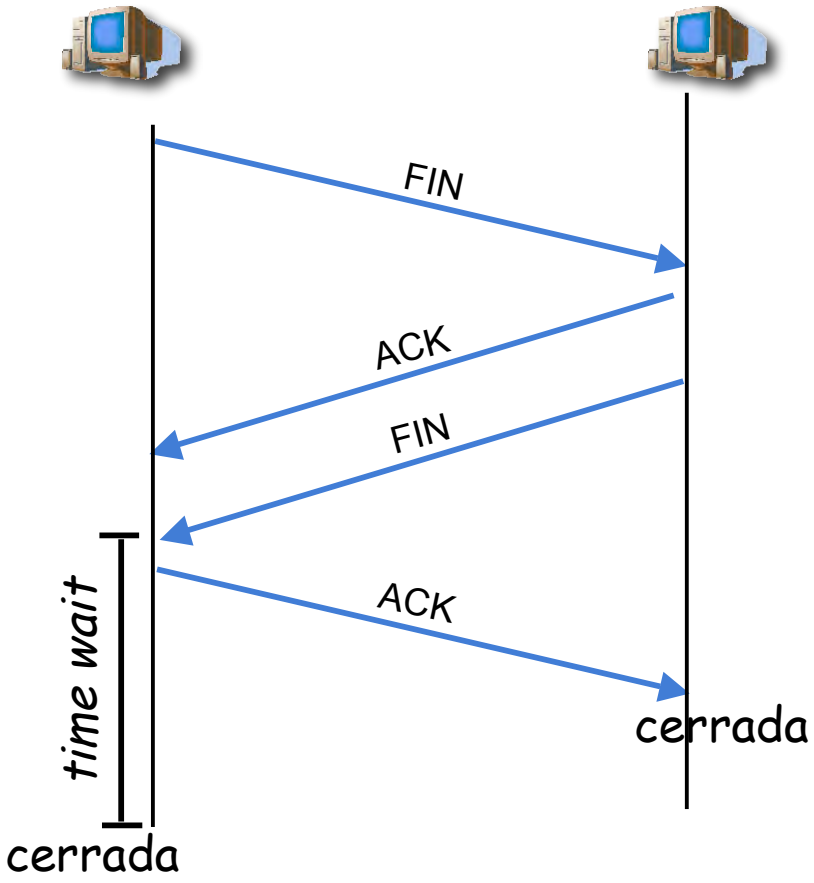
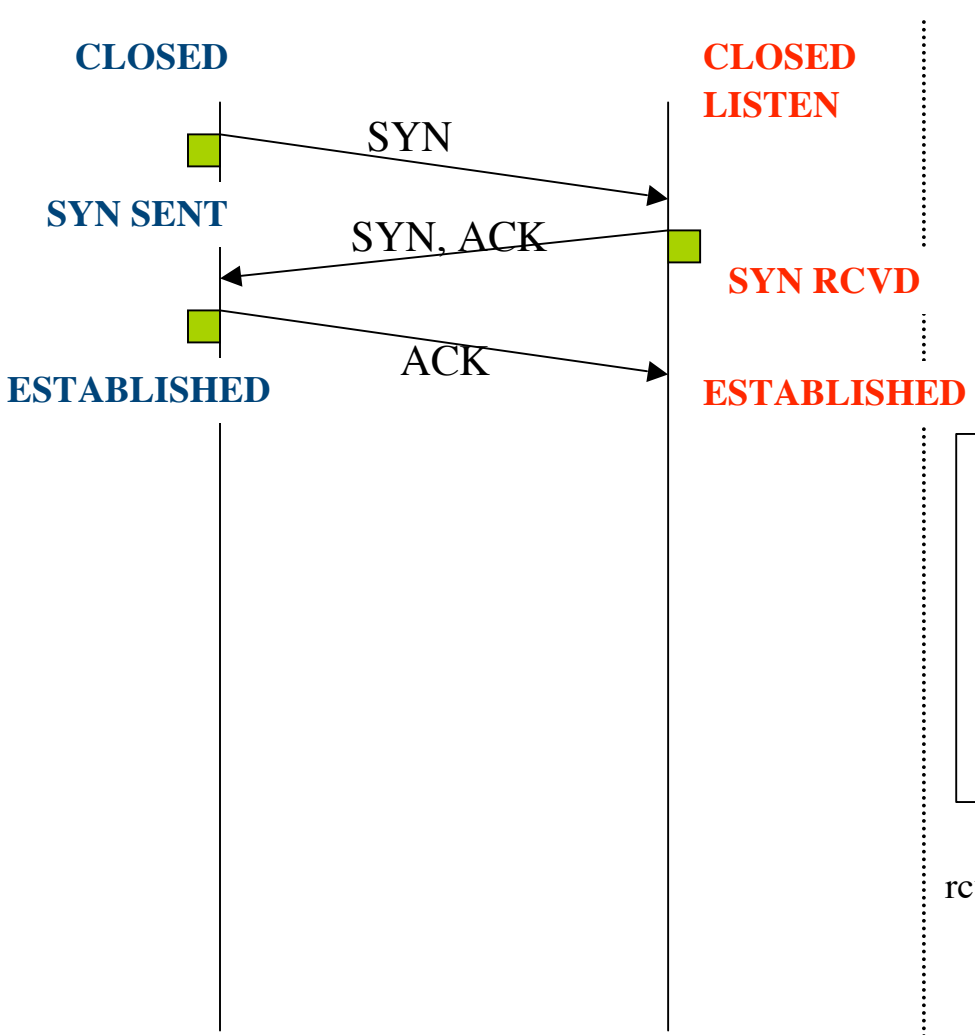


Diagrama de estados



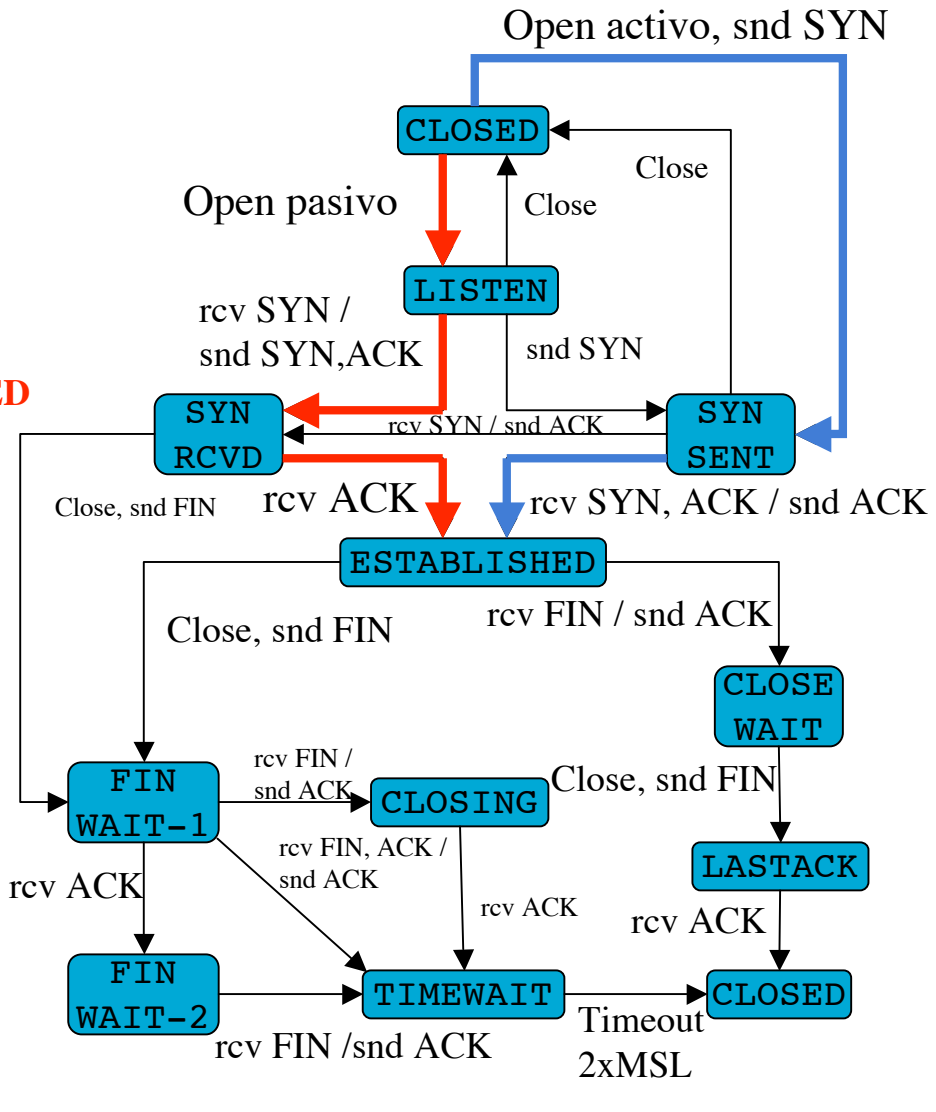
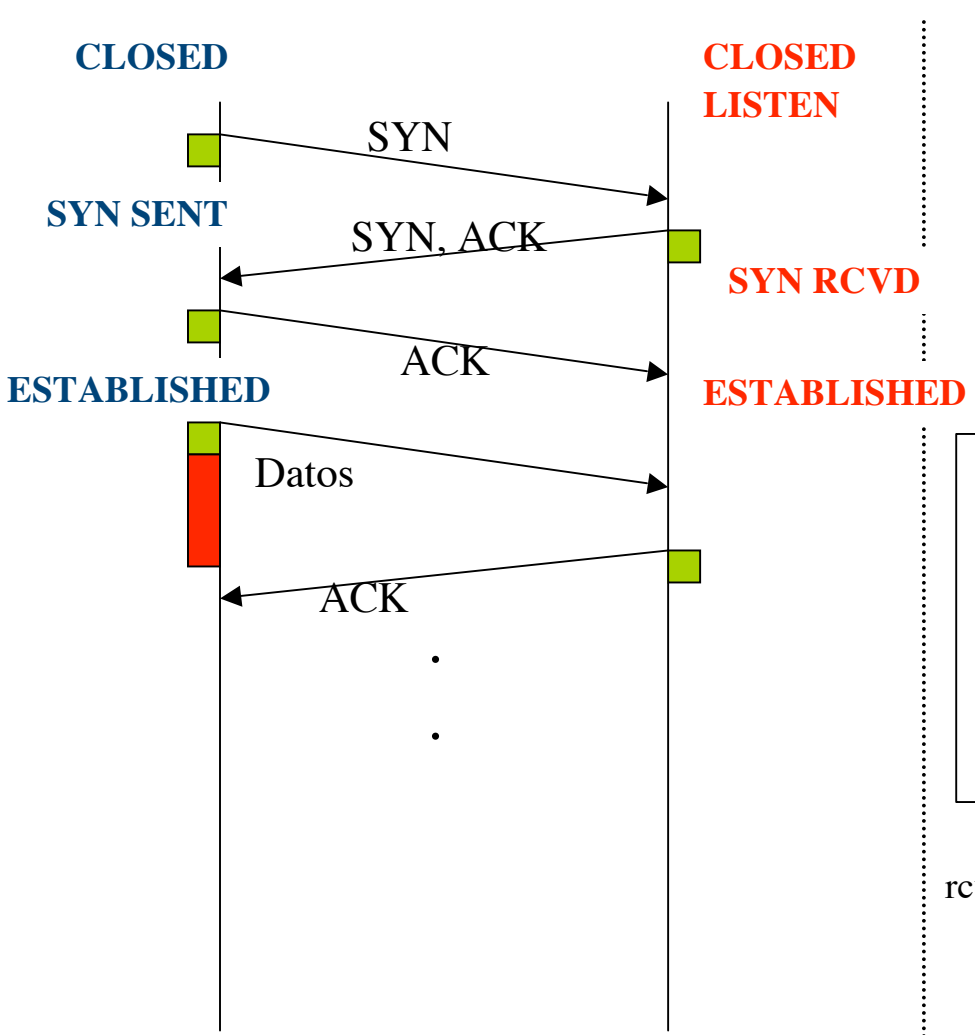


Diagrama de estados



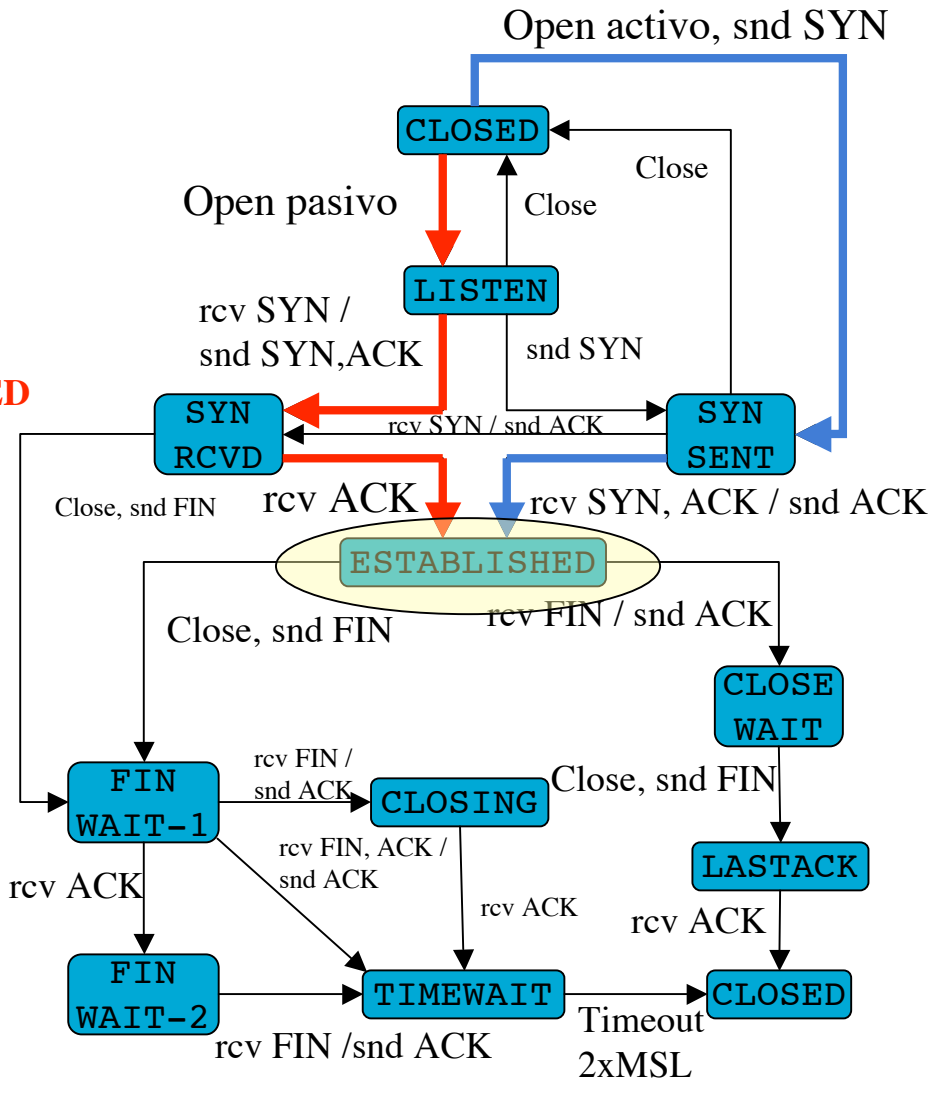
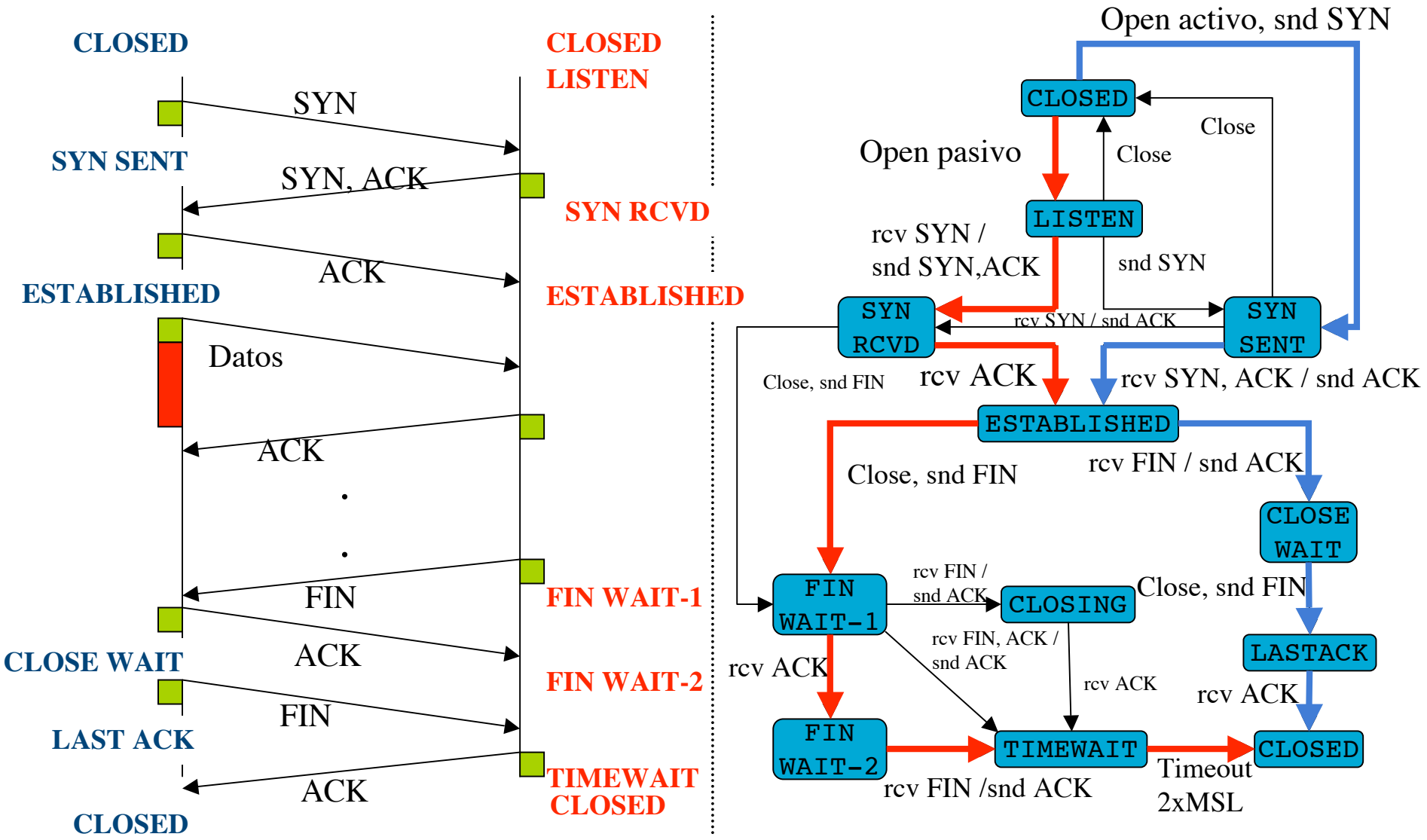
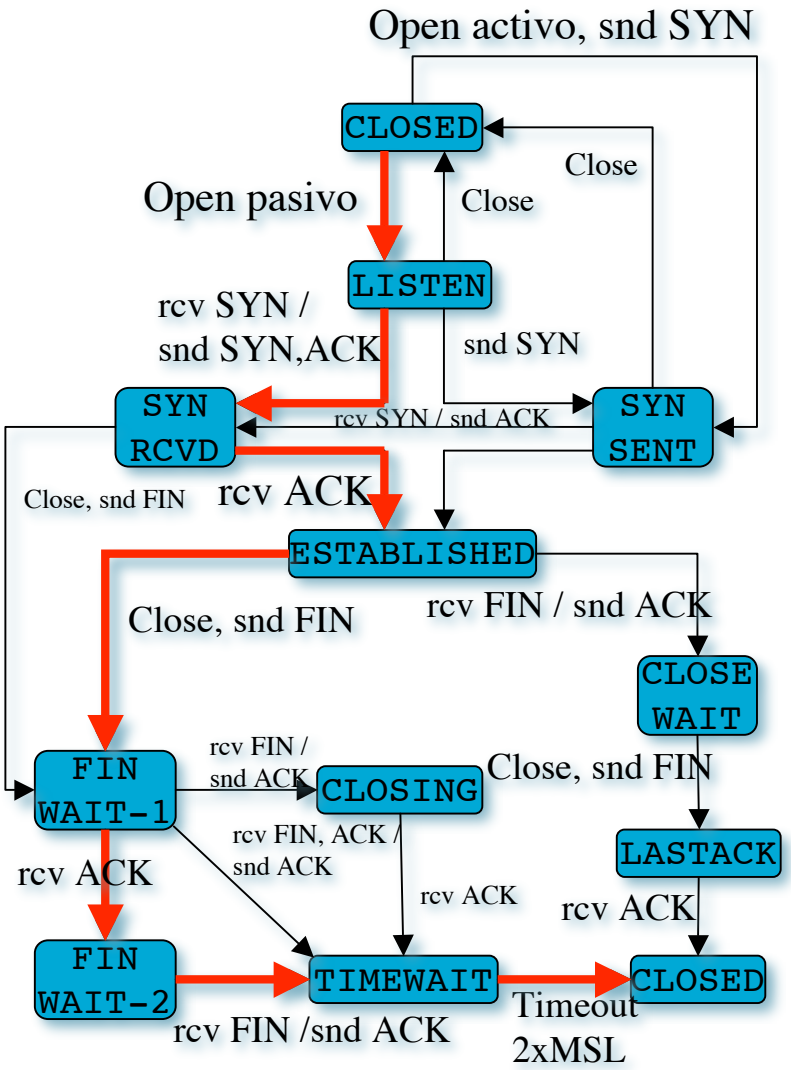


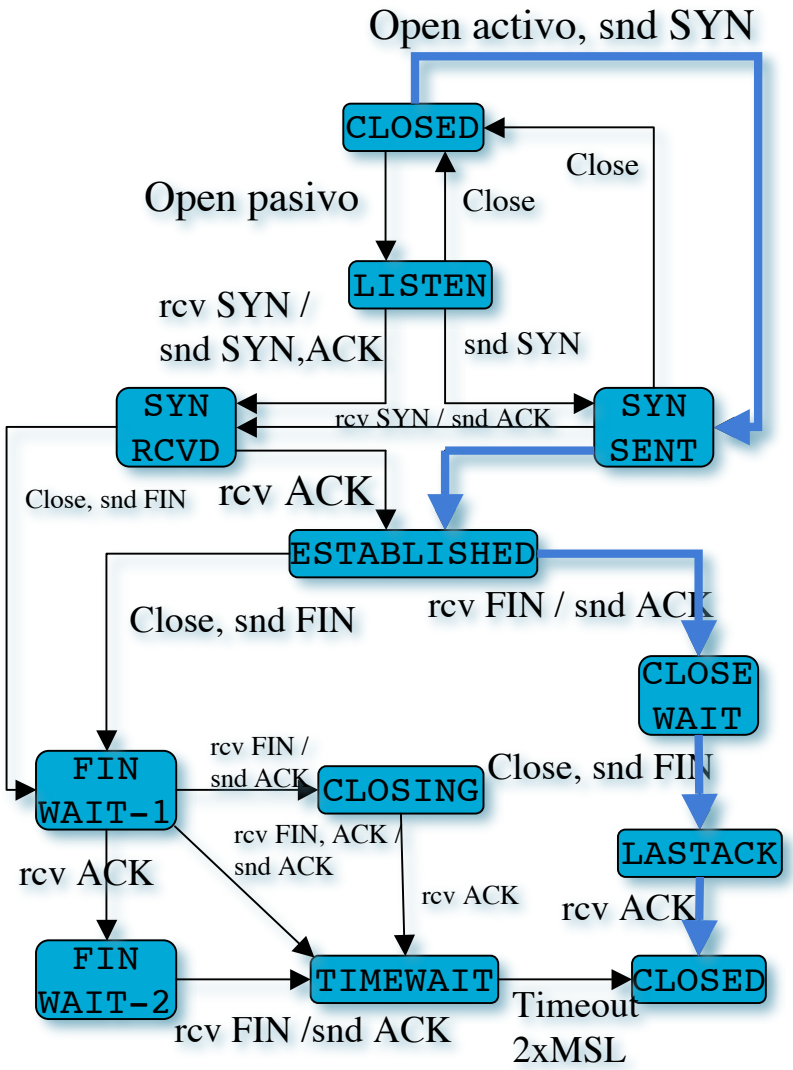
Diagrama de estados



Servidor



Cliente



Ejemplo

```
$ tcpdump -ttnlS tcp and host 10.1.11.1
Kernel filter, protocol ALL, datagram packet socket
tcpdump: listening on all devices
54.171 1.1.1.12.1798 > 10.1.11.1.telnet: S 3462181145:3462181145(0)
54.175 10.1.11.1.telnet > 1.1.1.12.1798: S 1997882026:1997882026(0) ack 3462181146
54.175 1.1.1.12.1798 > 10.1.11.1.telnet: . 3462181146:3462181146(0) ack 1997882027

54.177 1.1.1.12.1798 > 10.1.11.1.telnet: P 3462181146:3462181173(27) ack 1997882027
54.178 10.1.11.1.telnet > 1.1.1.12.1798: . 1997882027:1997882027(0) ack 3462181173
...

66.816 10.1.11.1.telnet > 1.1.1.12.1798: FP 1997882551:1997882559(8) ack 3462181333
66.816 1.1.1.12.1798 > 10.1.11.1.telnet: . 3462181333:3462181333(0) ack 1997882560
66.817 1.1.1.12.1798 > 10.1.11.1.telnet: F 3462181333:3462181333(0) ack 1997882560
66.818 10.1.11.1.telnet > 1.1.1.12.1798: . 1997882560:1997882560(0) ack 3462181334
```

Contenido

- Internet Protocol
  - Características
  - IP en LAN Ethernet (ARP)
  - Direccionamiento (subredes, Proxy-ARP, CIDR)
  - Fragmentación y reensamblado
  - ICMP
- Enrutamiento en Internet
  - Distance-Vector
  - Link-State
  - Path-Vector
- UDP
- TCP
  - Características
  - Gestión de conexiones
  - **Control de flujo**
  - Retransmisiones
  - Control de congestión

Lecturas recomendadas

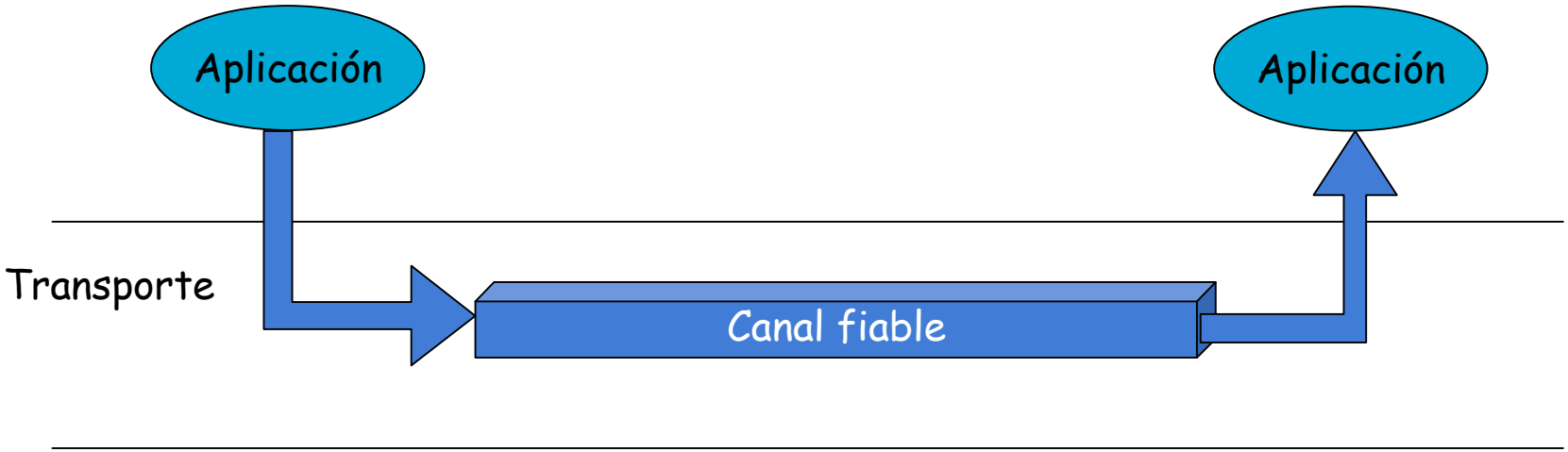
– [Forouzan] 12.1-12.3, 12.9

Contenido

- Introducción
- Control de flujo
- Formato del segmento TCP

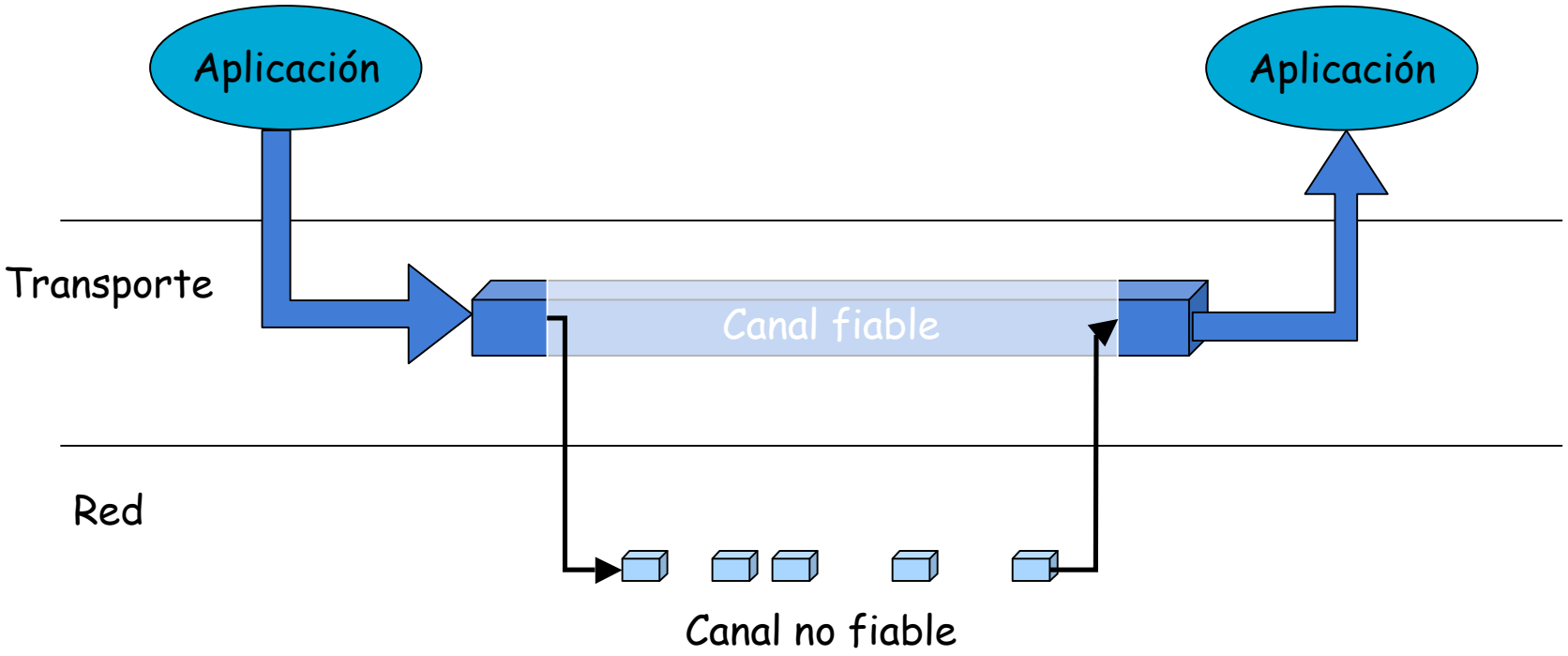
Transferencia fiable de datos

- Importante en nivel de aplicación, transporte, enlace



Transferencia fiable de datos

- Importante en nivel de aplicación, transporte, enlace

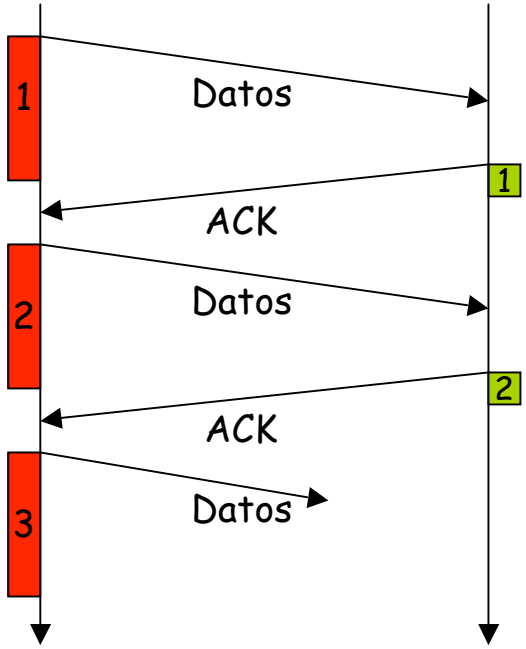


- Basado en nivel no fiable

Transferencia fiable de datos

¿Cómo lograrla?

- Mecanismo de confirmaciones
- Se numeran los bytes para confirmarlos
  - Los SYNs establecen los números de secuencia iniciales



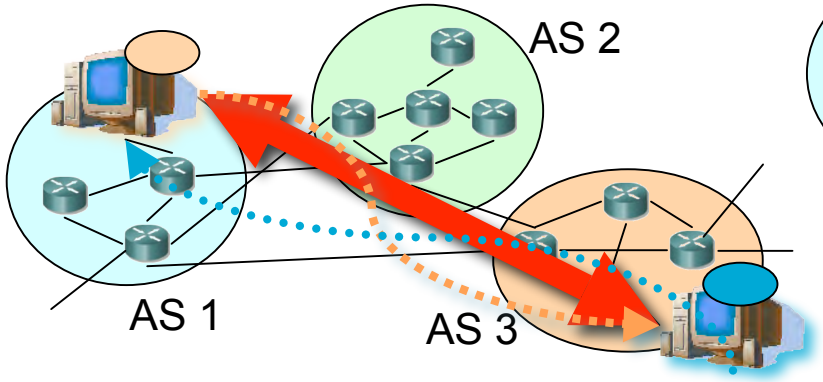
Servicio de entrega por *Stream*

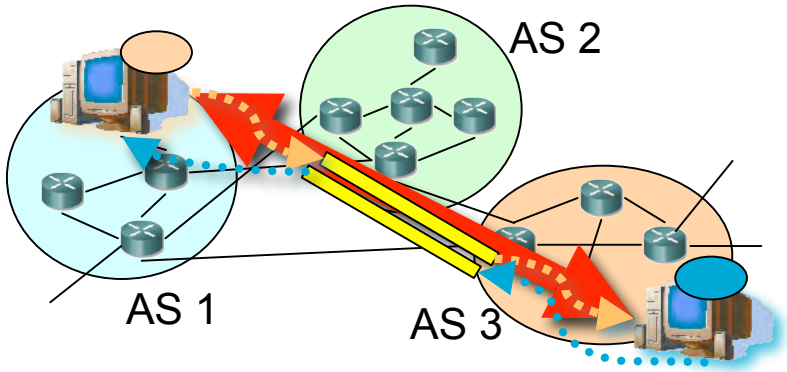
UDP

- Orientado a datagramas
- Recibe un bloque de datos de la aplicación
- Le añade su cabecera
- Se lo entrega a IP

TCP

- El proceso no ve un flujo de paquetes
- Ve que escribe datos y se reciben en el mismo orden
- Un *flujo* de datos





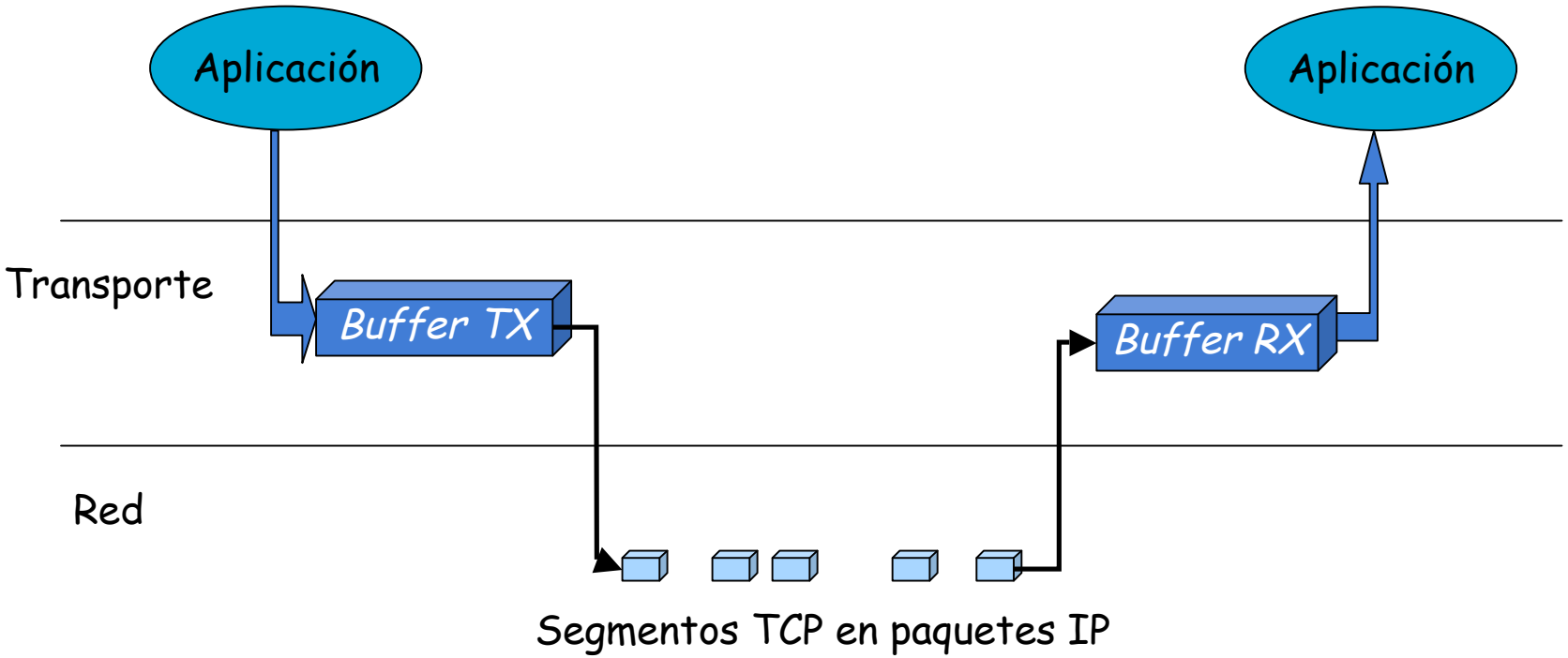
Servicio de entrega por *Stream*

TCP Emisor

- Emisor acumula datos para mandar paquetes *grandes*
- Mantiene los datos hasta que son confirmados

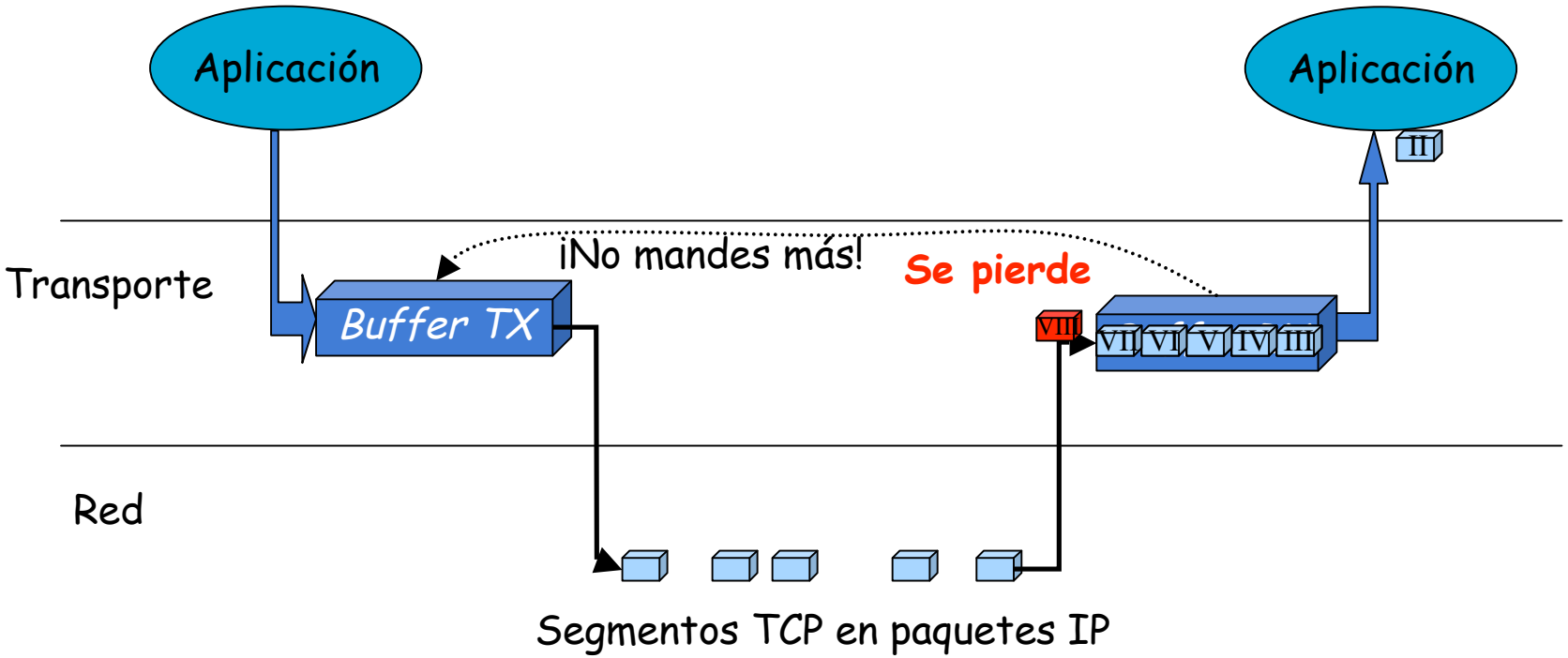
TCP Receptor

- Hay que reordenar los paquetes
- Aplicación puede que lea más despacio



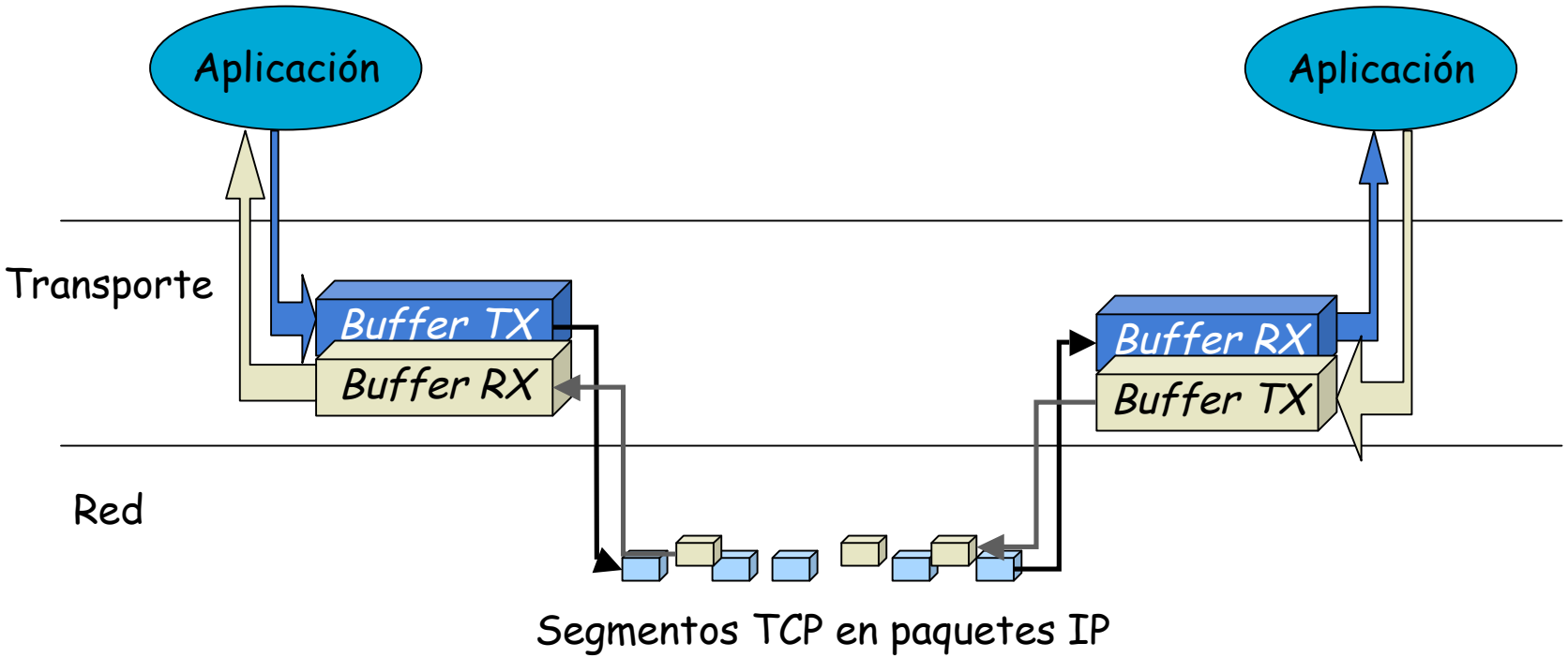
Control de flujo

- Receptor lee más despacio que lo que recibe (. . .)
- *Buffer* se desbordaría
- Receptor informa a emisor del espacio libre



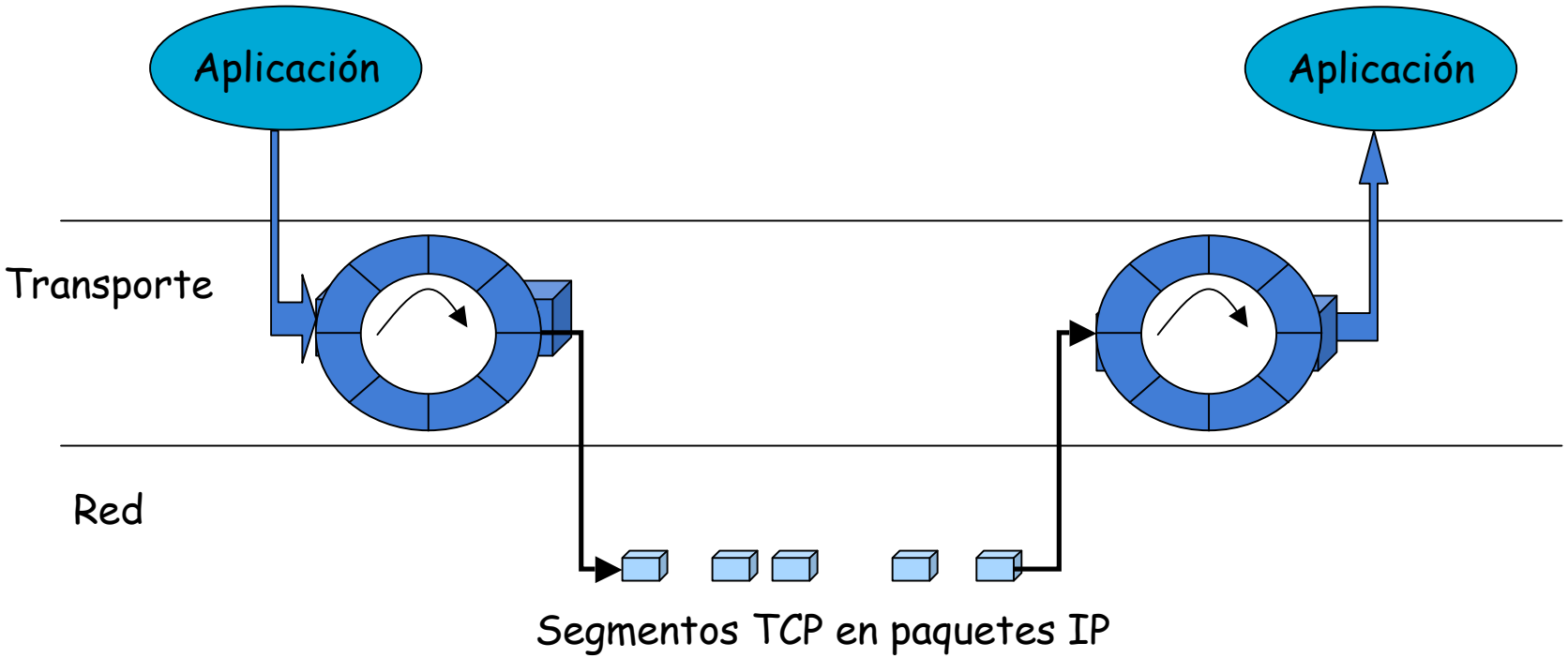
Un recordatorio (…)

- **Comunicación full-duplex**
- Por simplicidad hablaremos solo de un sentido



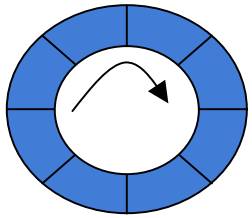
Buffers de emisión y recepción

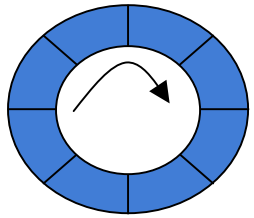
- Buffer circular (…)
- Protocolo de **Ventana Deslizante**
- Se confirma el último dato consecutivo recibido



Ventana deslizante en TCP

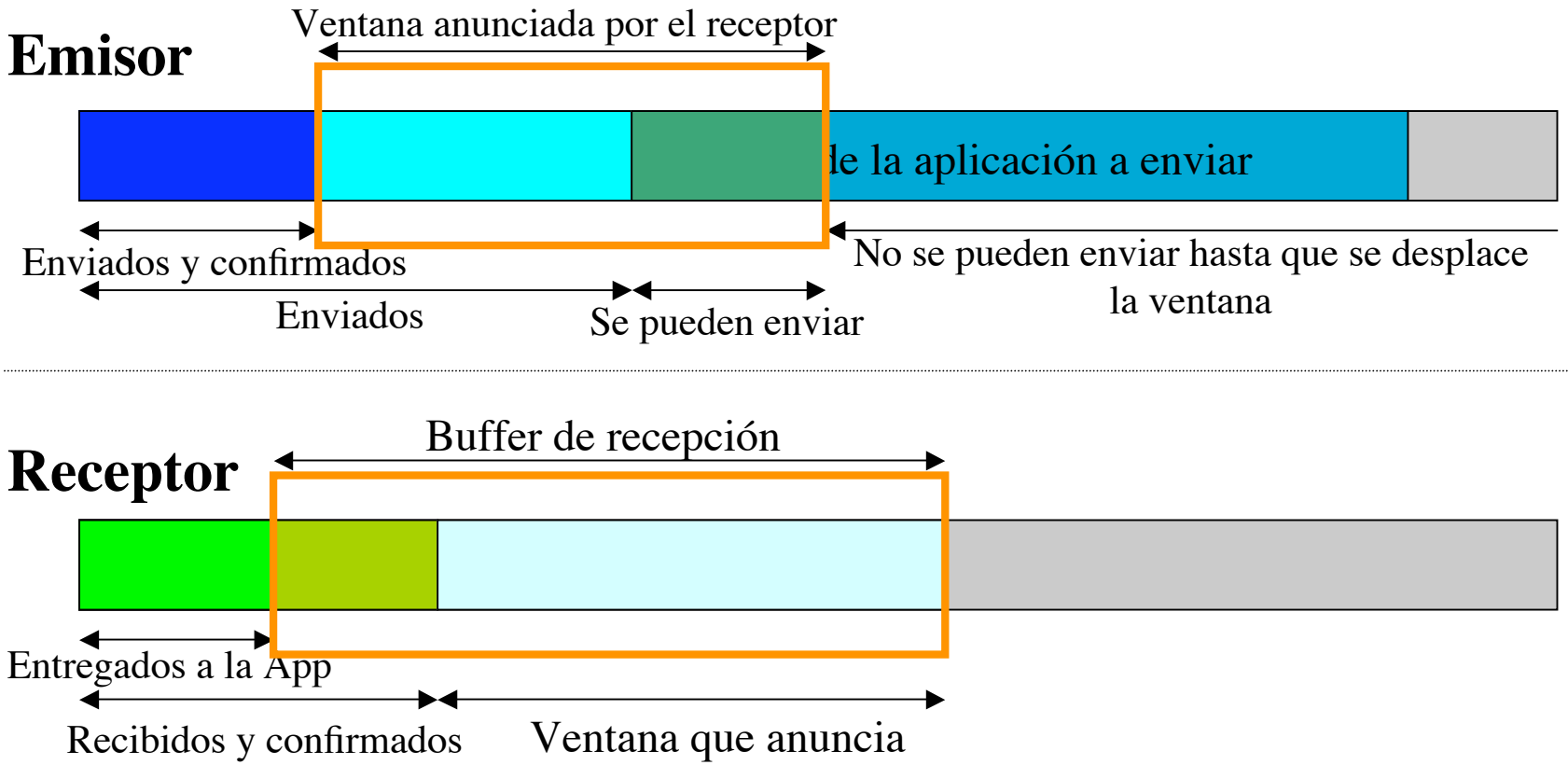
- Por simplicidad analicemos solo un sentido





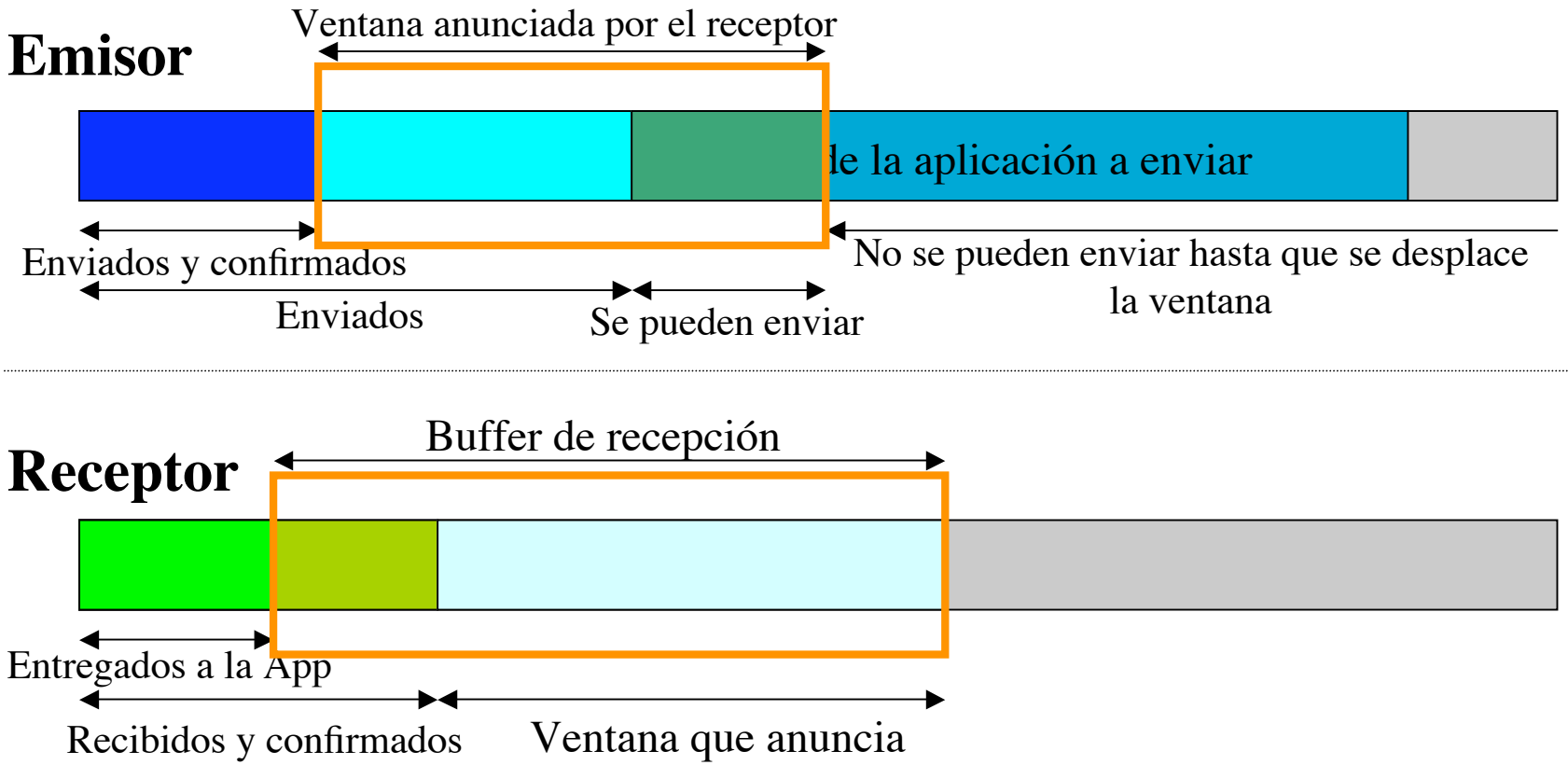
Ventana deslizante en TCP

- Por simplicidad analicemos solo un sentido



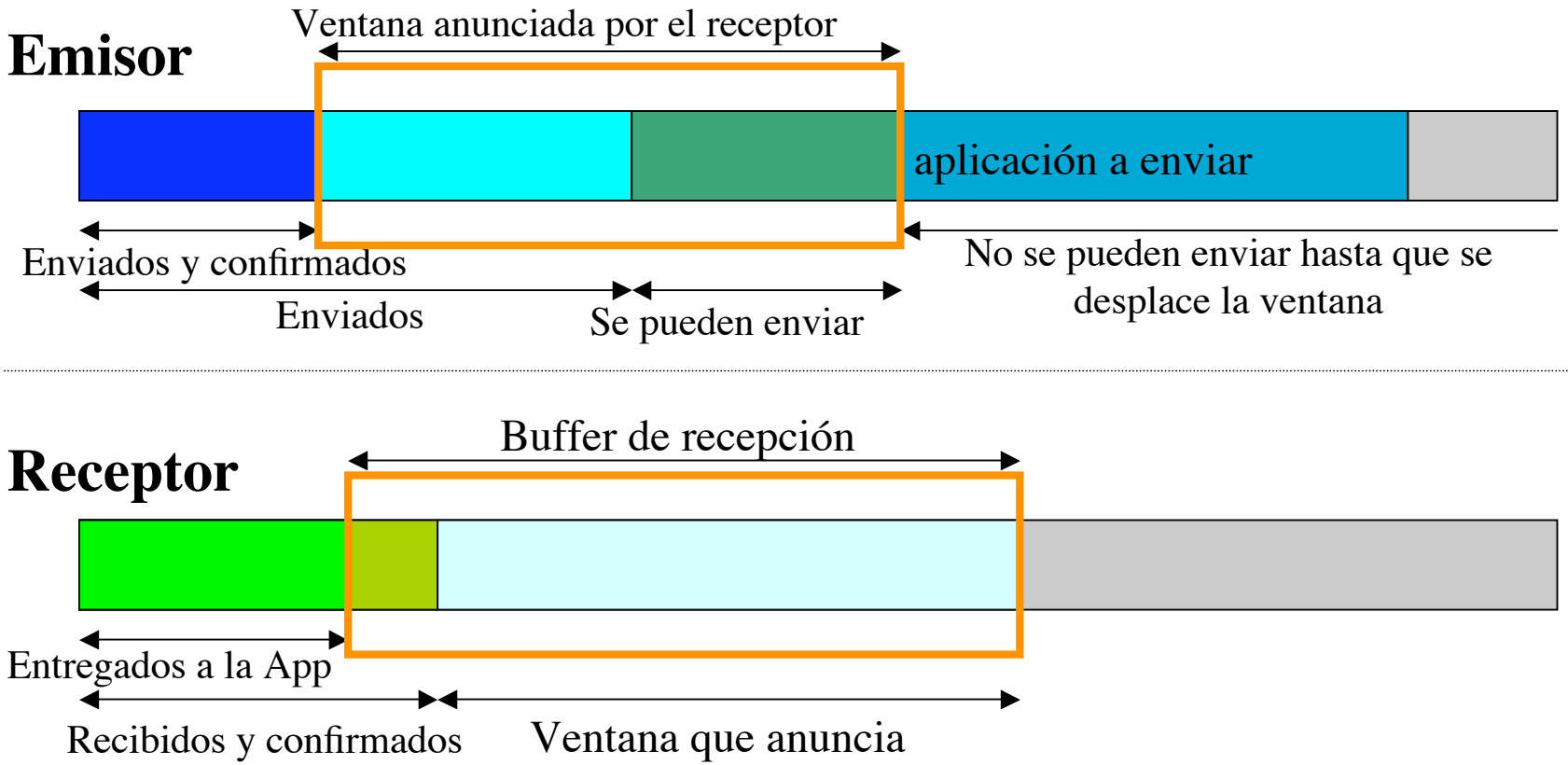
Ventana deslizante en TCP

- La aplicación **receptor lee bytes** del stream
  - La ventana **se abre** en el emisor
  - **Se desliza** en el receptor (…)



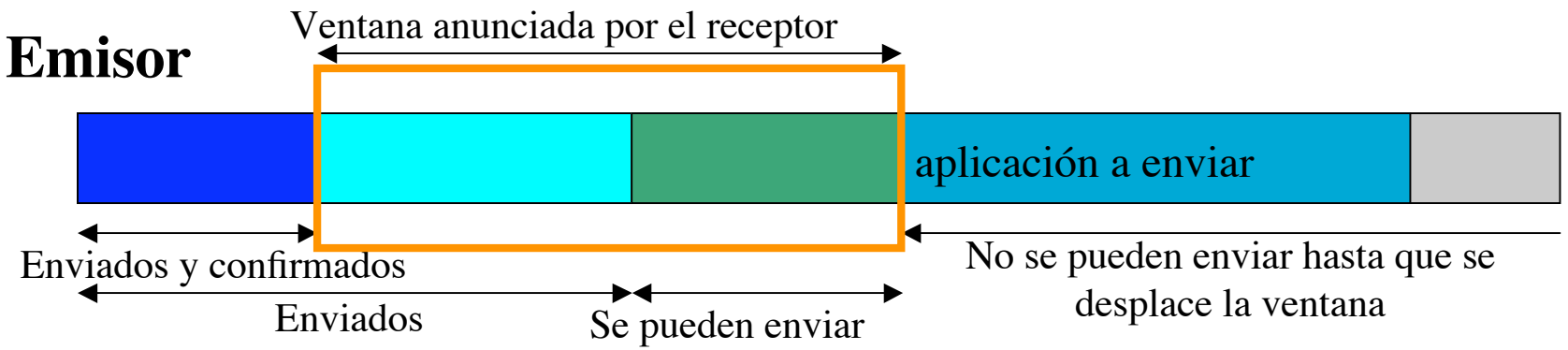
Ventana deslizante en TCP

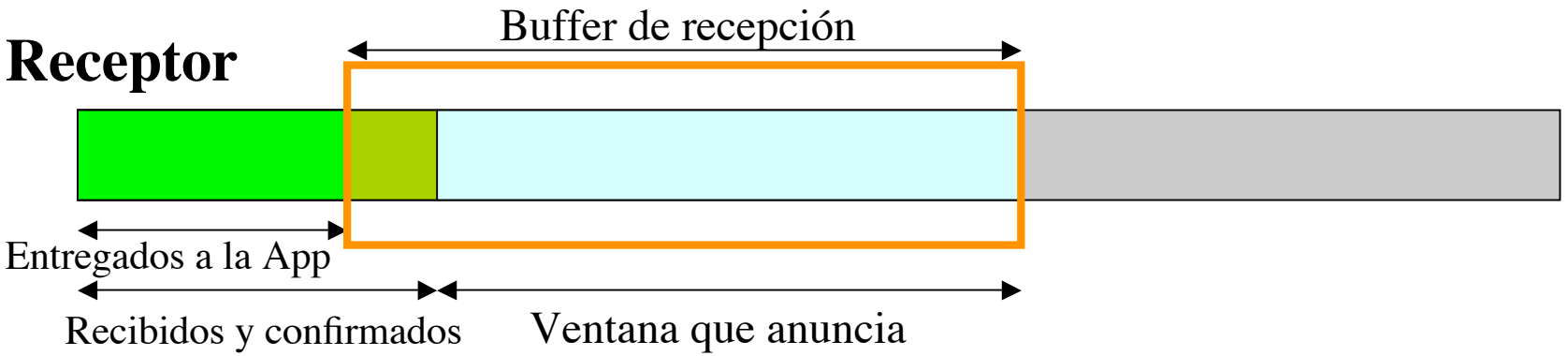
- La aplicación **receptor lee bytes** del stream
  - La ventana **se abre** en el emisor
  - **Se desliza** en el receptor



Ventana deslizante en TCP

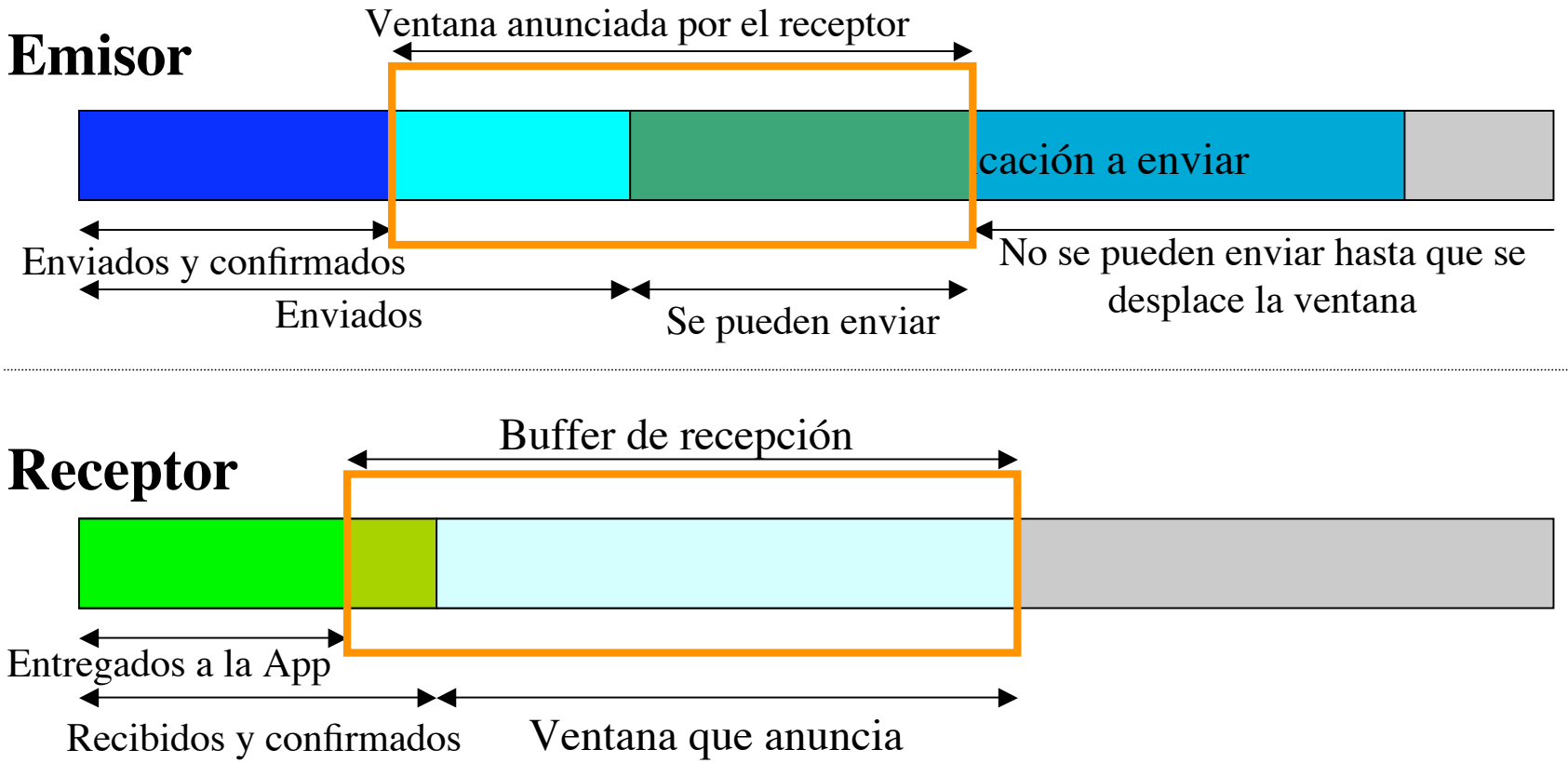
- Se reciben más confirmaciones
- La ventana **se desliza** en el emisor (…)



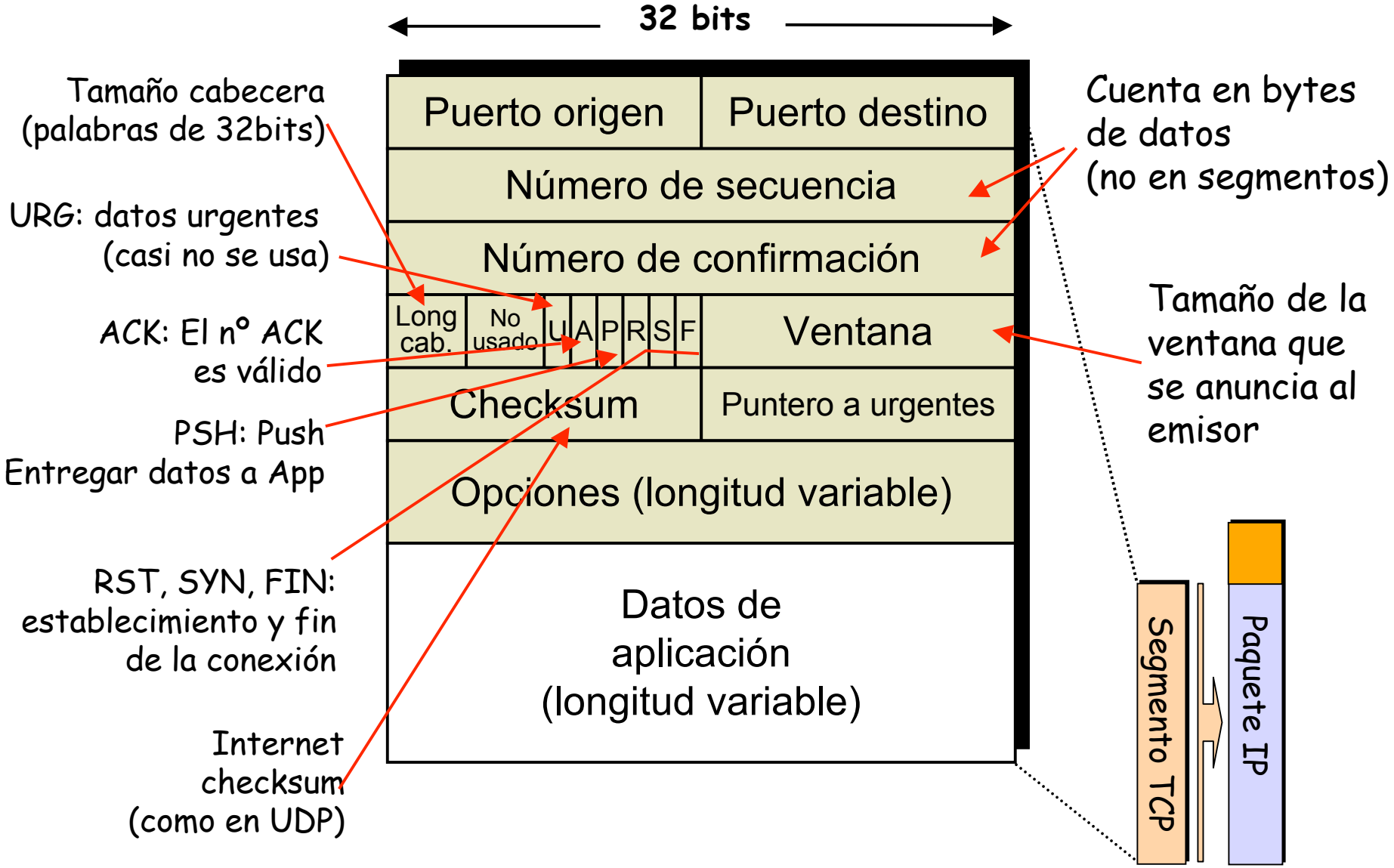


Ventana deslizante en TCP

- Se reciben más confirmaciones
- La ventana **se desliza** en el emisor



Segmento TCP



Contenido

- Internet Protocol
  - Características
  - IP en LAN Ethernet (ARP)
  - Direccionamiento (subredes, Proxy-ARP, CIDR)
  - Fragmentación y reensamblado
  - ICMP
- Enrutamiento en Internet
  - Distance-Vector
  - Link-State
  - Path-Vector
- UDP
- TCP
  - Características
  - Gestión de conexiones
  - Control de flujo
  - **Retransmisiones**
  - Control de congestión

Lecturas recomendadas

– [Stevens] 18.10, 19, 20.7, 21.2-21.4

Contenido

- Opciones
- Número de secuencia y confirmación
- *Delayed ACK*
- Tamaño de la ventana
- Retransmisiones
  - Algoritmo de Jackobson
  - Algoritmo de Karn
- Algoritmo de Nagle

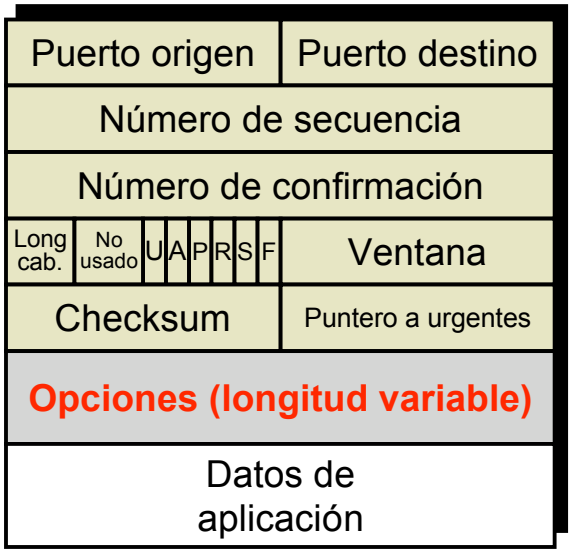
Opciones

De 1 byte

- *End-of-option*
- *No-operation* (para alinear)

Múltiples bytes

- *Maximum Segment Size* (MSS)
  - Máximo bloque de datos en un segmento
  - En el establecimiento
  - El menor de los dos
  - Por defecto 536




```
1.1.1.12.1798 > 10.1.11.1.23: S 0:0(0) <mss 1460>
10.1.11.1.23 > 1.1.1.12.1798: S 0:0(0) ack 1 <mss 960>
1.1.1.12.1798 > 10.1.11.1.23: . 1:1(0) ack 1
```

Números de Secuencia y ACK

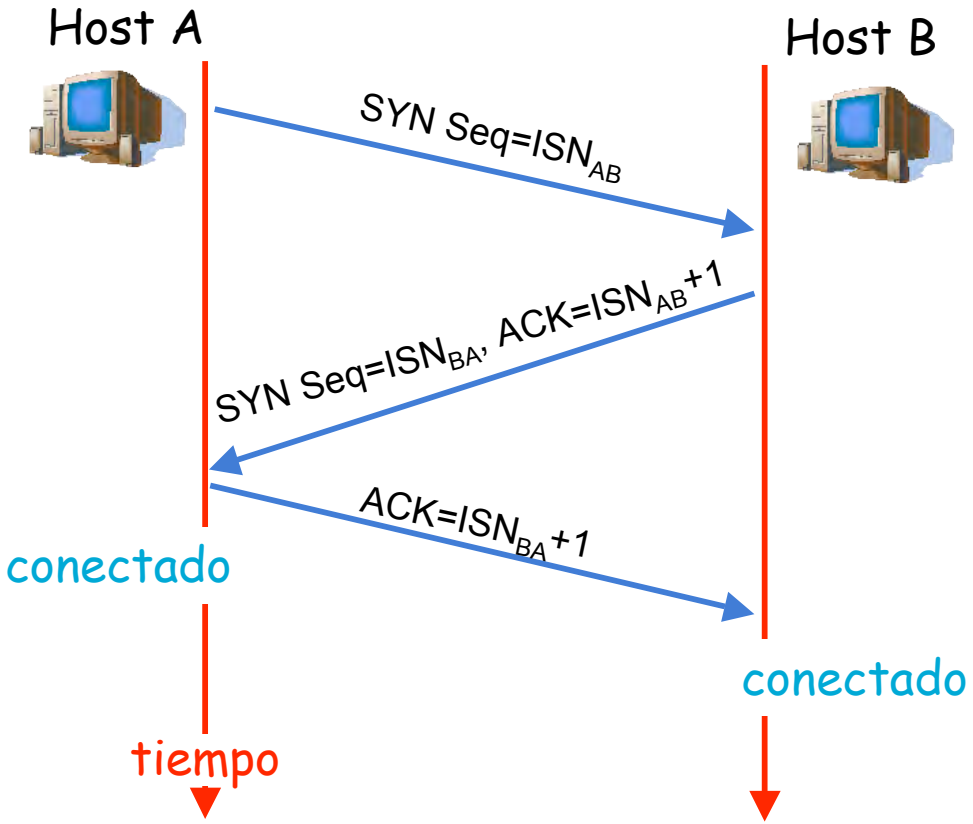
Nº de secuencia:

- Primer byte de datos en el segmento
- SYN y FIN gastan 1 nº de secuencia
- Para poder ser confirmados

Puerto origen	Puerto destino
Número de secuencia	
Número de confirmación	
Long cab. / No usado / U **A** P R **S** F	Ventana
Checksum	Puntero a urgentes
Opciones (longitud variable)	
Datos de aplicación	

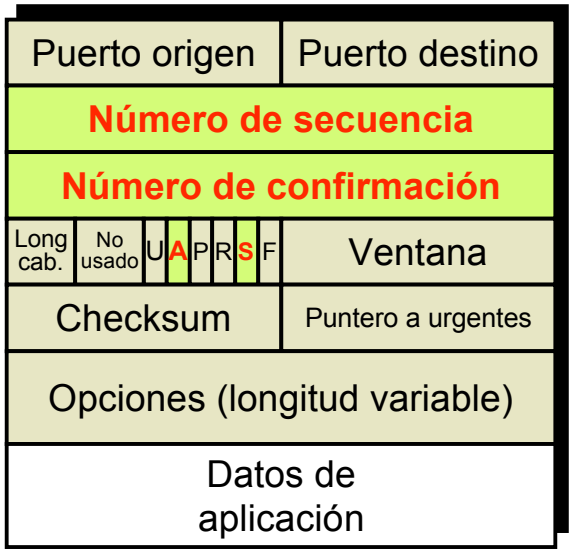
Nº de ACK:

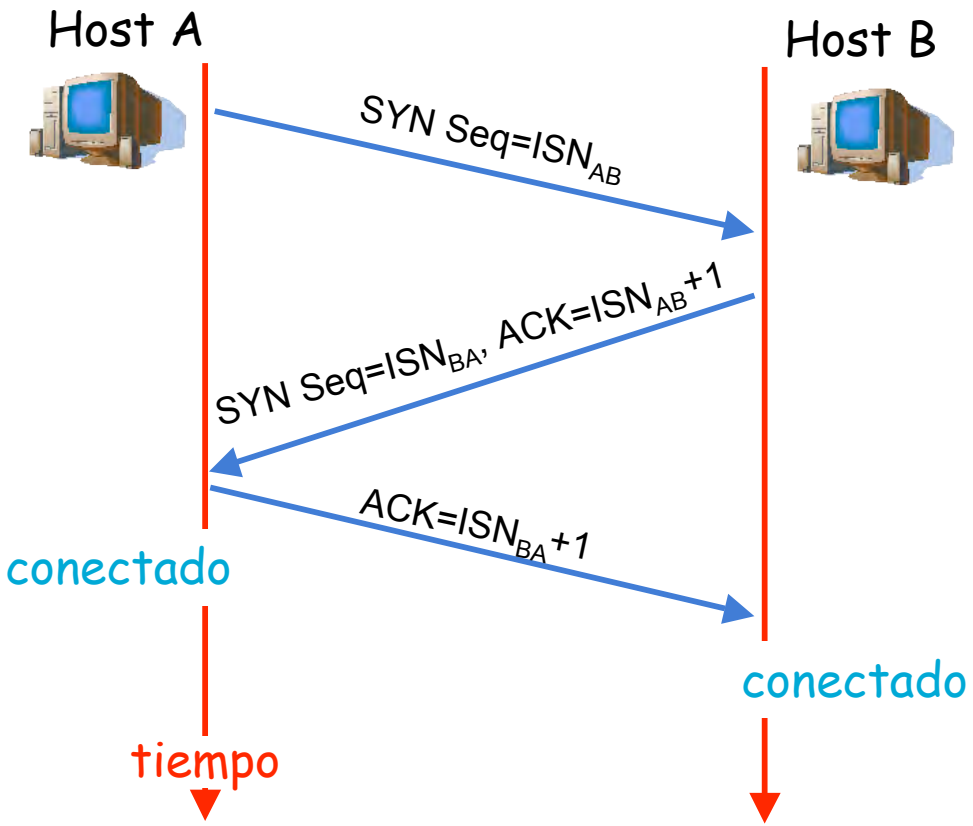
- Siguiente byte que se espera recibir
- ACK acumulado



Números de Secuencia y ACK

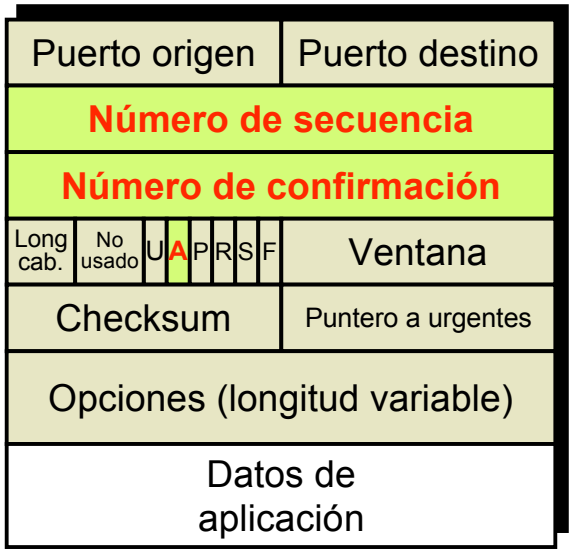
```
1.1.1.12.1798 > 10.1.11.1.23: S 3462181145:3462181145(0)
10.1.11.1.23 > 1.1.1.12.1798: S 1997882026:1997882026(0) ack 3462181146
1.1.1.12.1798 > 10.1.11.1.23: . 3462181146:3462181146(0) ack 1997882027
```

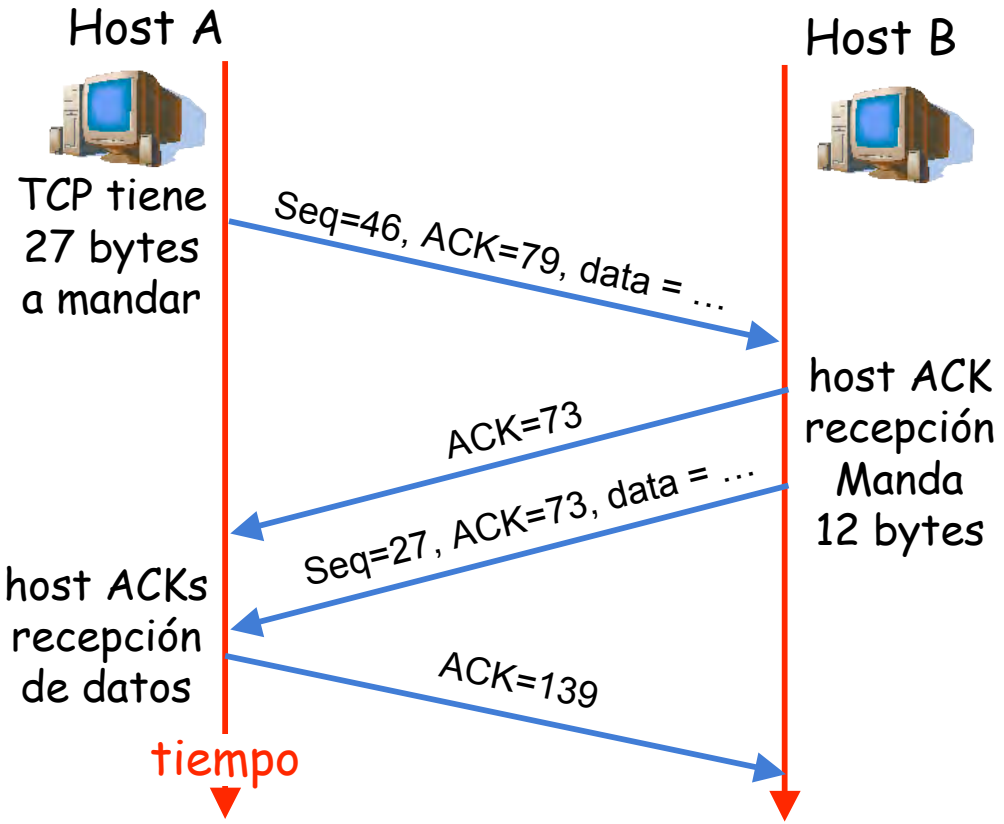




Números de Secuencia y ACK

```
1.1.1.12.1798 > 10.1.11.1.23: P 3462181146:3462181173(27) ack 1997882027
10.1.11.1.23 > 1.1.1.12.1798: . 1997882027:1997882027(0) ack 3462181173
10.1.11.1.23 > 1.1.1.12.1798: P 1997882027:1997882039(12) ack 3462181173
1.1.1.12.1798 > 10.1.11.1.23: . 3462181173:3462181173(0) ack 1997882039
```

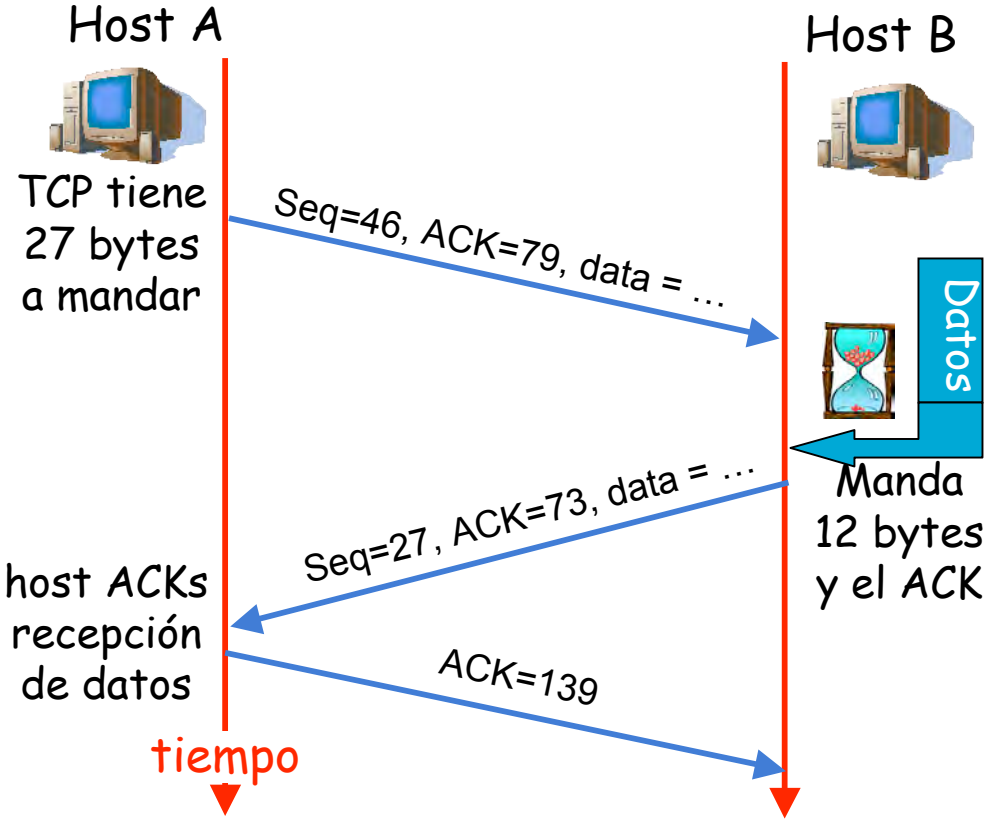




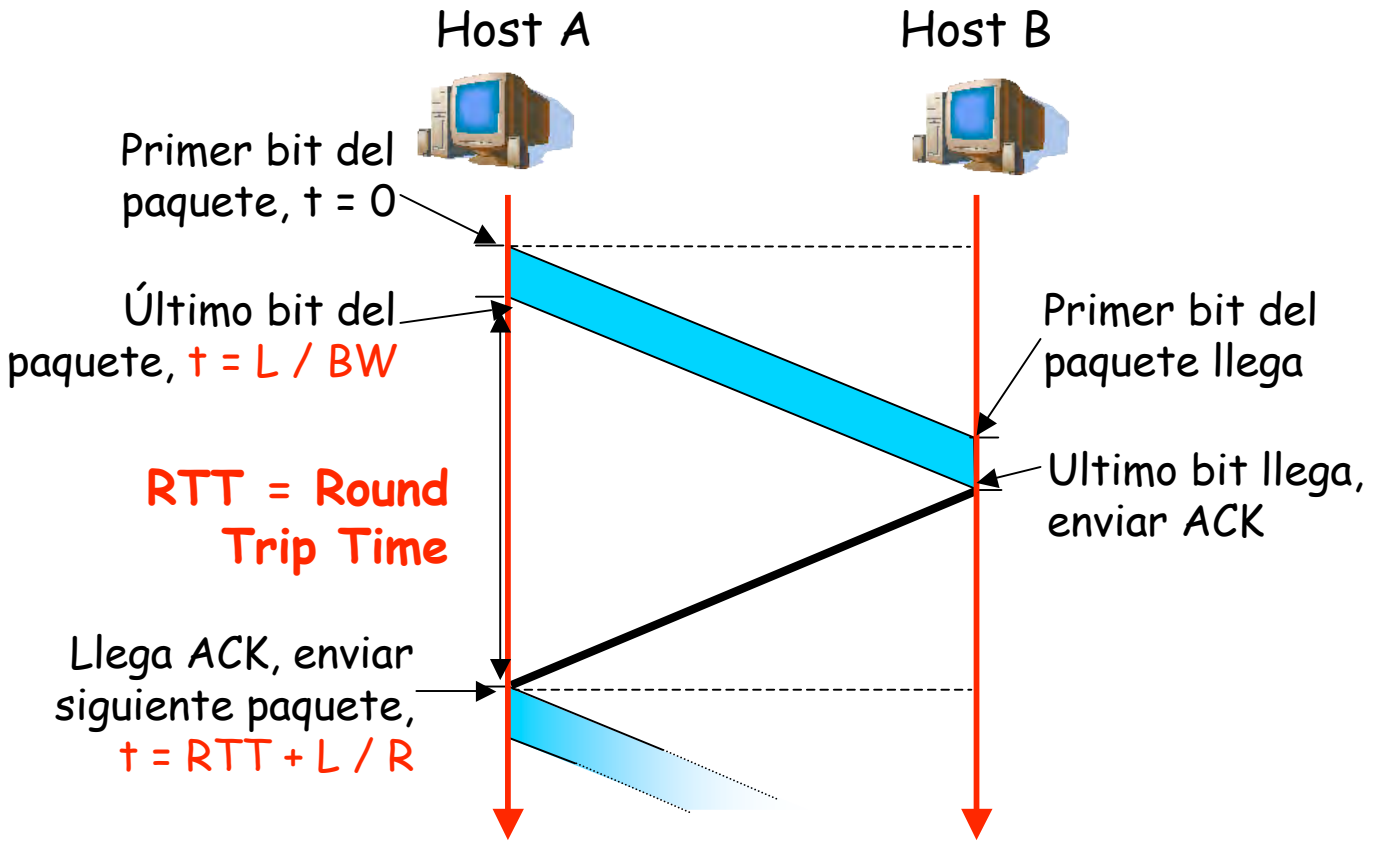
Delayed ACK

- Esperar un tiempo a mandar ACK (…)
- Intenta conseguir datos que enviar (…)
- Incluirá el ACK con ellos (…)
- *Piggybacking*

- Debe mandar ACK (Host Req.) si:
  - Pasan 500ms
  - 1 ACK al menos por cada 2 segmentos “llenos” (del *MSS*)



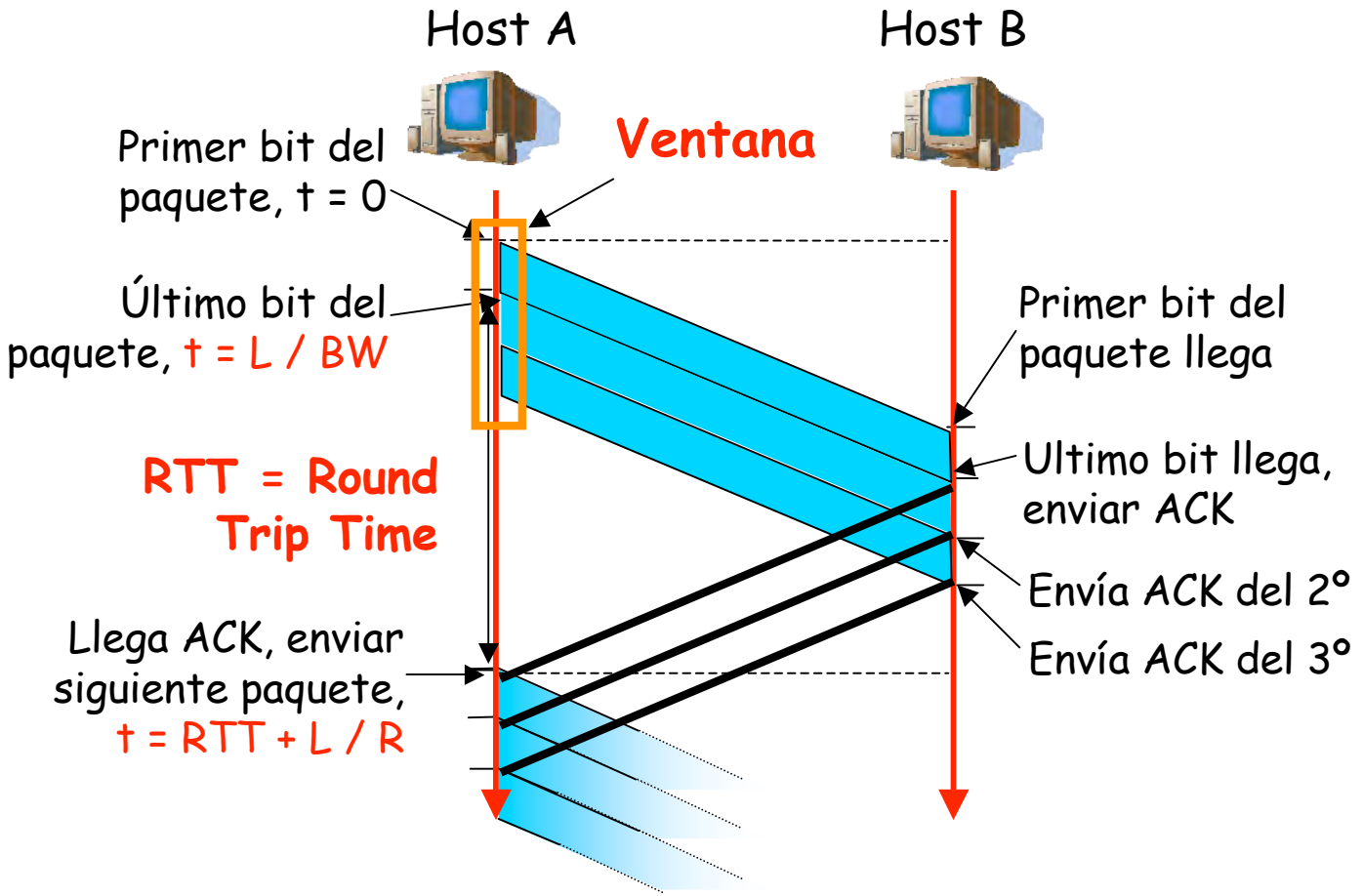
Tamaño de la ventana



Ejemplo: 1 Gbps, RTT = 30 ms, L = 1KByte

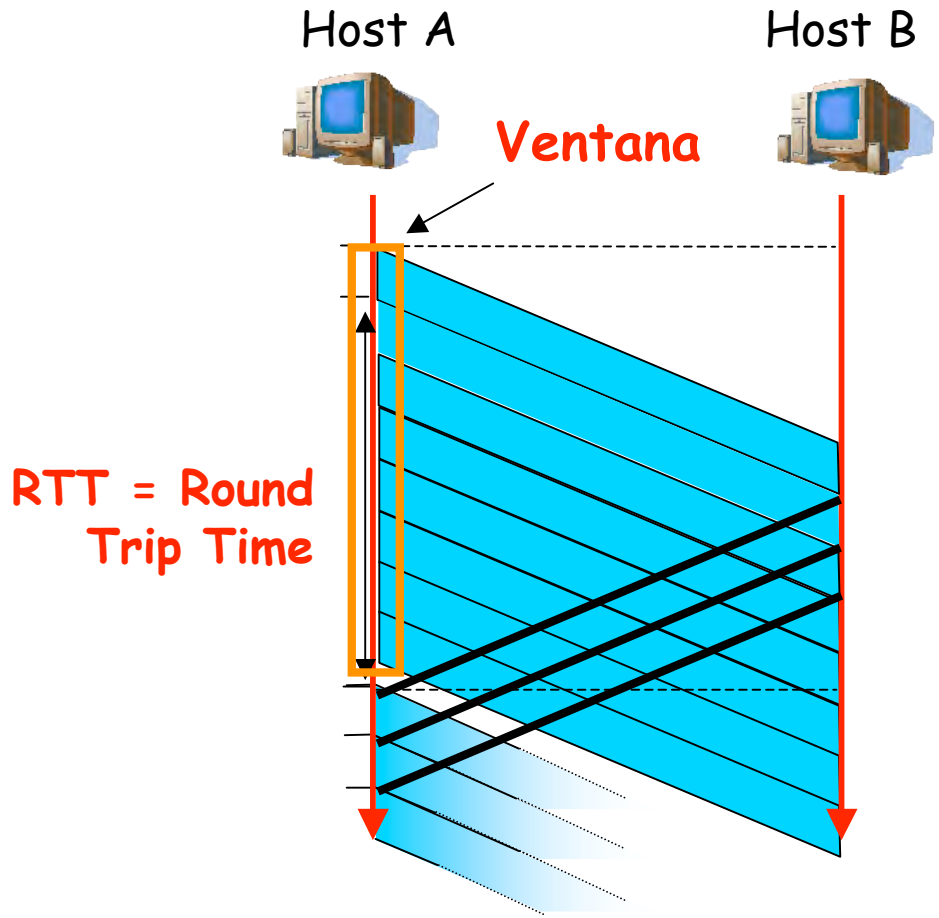
- ¡ L / RTT = 270 Kbps !
- ¡ Se emplea el enlace el 0.027% del tiempo !

Tamaño de la ventana



- ¿Cuánto puede aumentar la utilización?
- ¿Qué tamaño de ventana poner?

Tamaño de la ventana



- Ventana ≥ RTT x BW

Producto RTTxBW

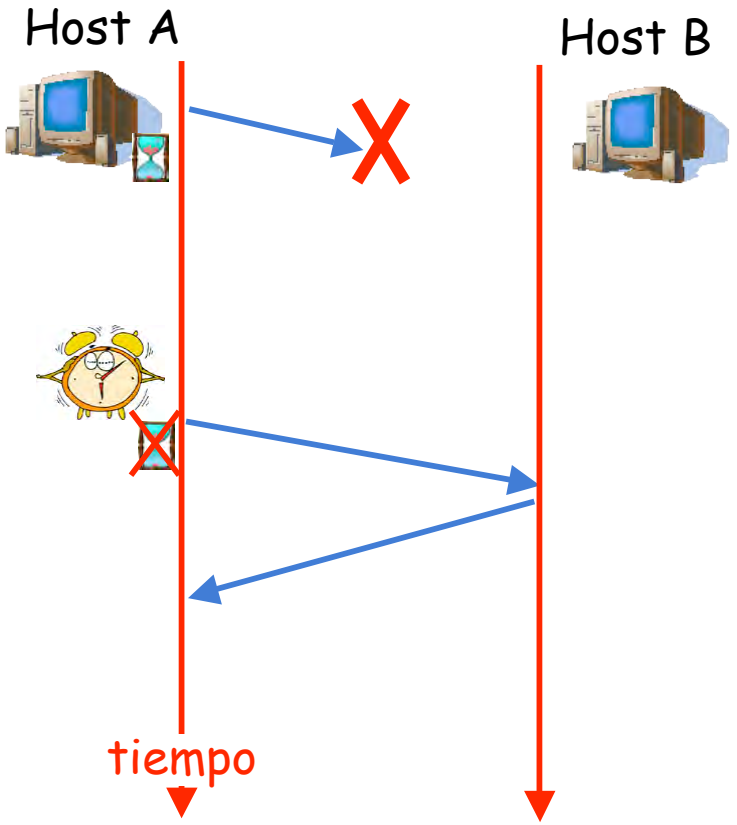
- Enlace E1 (2.048Kbps) a través de Europa (60ms)
  - Ventana > 15KBytes
  - Tamaño máximo de la ventana es 64KBytes
- Problemas:
  - Enlace de 10Mbps a través de Europa
    - Ventana > 74KBytes
    - Mayor que el máximo que permiten 16bits (!!)
  - Enlace E1 transoceánico (300ms)
    - Ventana > 76KBytes (!!)
  - Enlace Gigabit dentro de España (20ms)
    - Ventana > 2.5MBytes (!!!)
- Soluciones:
  - Aumentar el tamaño máximo de la ventana a 32bits (window scale)
  - Realizar varias conexiones simultáneamente (P2P)

Retransmisiones

- Al enviar un paquete con datos se inicia un *timer* (…)
- Si caduca sin que llegue la confirmación es que se ha perdido (…)

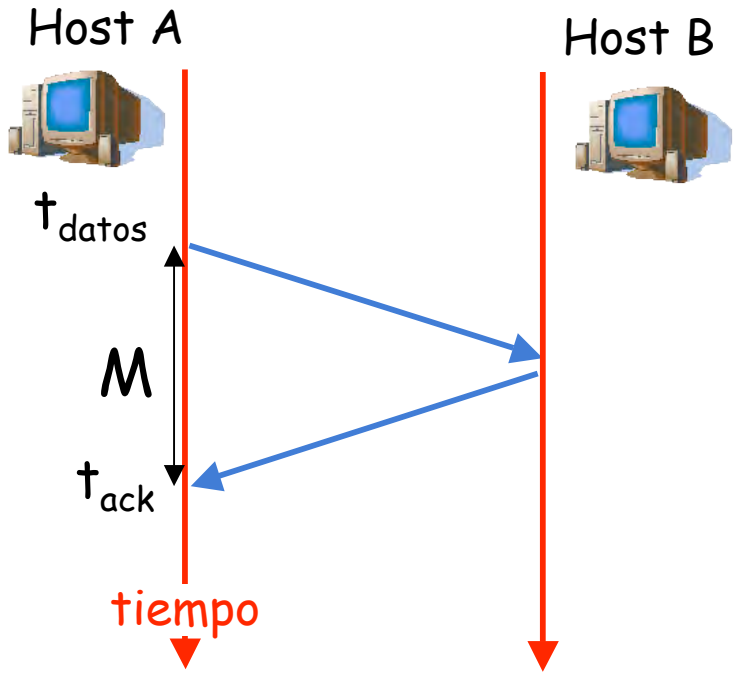
***Retransmission Timeout* (RTO):**

- Debería tener el valor del RTT
  - Menor ⇒ expira sin que llegue el *ack*
  - Mayor ⇒ tarda en reconocer la pérdida
- Cuando expira se reenvía y el timeout se dobla (*exponential backoff*)
- Hace falta conocer el RTT
  - Teniendo en cuenta que las rutas o su estado pueden cambiar, y con ellos el retardo



Estimación del RTT

- Tiempo entre el envío de datos y la llegada de su confirmación
- Desviados por el *Delayed-ack*



RTT = R
tack - tdatos = M
$R = \alpha R + (1-\alpha)M$
$RTO = \beta R$

$\alpha = 0.8$ ó 0.9 (*smoothing factor*)
$\beta = 2$ (*delay variance factor*)
R = "RTT suavizado"

Algoritmo de Jackobson

- *Must*
- Estima la desviación estándar aproximándola con la desviación media
- Especialmente importante en enlaces lentos donde el tamaño del paquete afecta bastante al RTT

$$Err = M - R$$
$$R = R + g\,Err$$
$$D = D + h\,(|Err| - D)$$
$$RTO = R + 4D$$

$$RTT = R$$
$$tack - tdatos = M$$
$$R = \alpha R + (1-\alpha)M$$
$$RTO = \beta R$$

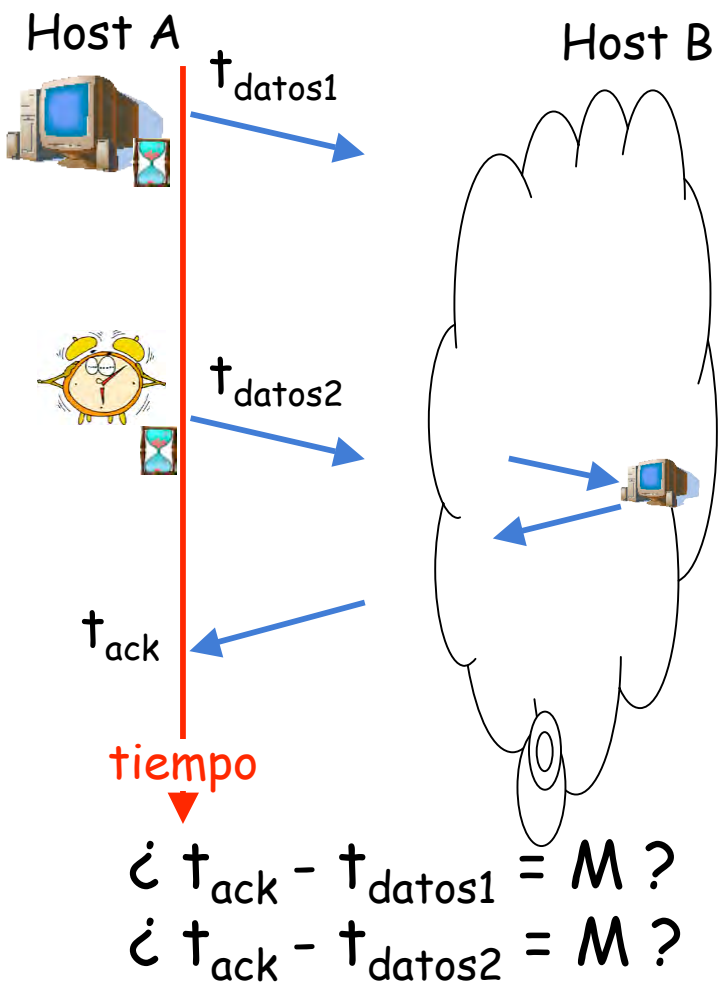
g = 1/8
h = 1/4
D = "Desviación media suavizada"

$\alpha = 0.8$ ó 0.9 (*smoothing factor*)
$\beta = 2$ (*delay variance factor*)
R = "RTT suavizado"

Algoritmo de Karn

Si hay retransmisiones

- No se sabe a quién corresponde el ACK
- No se toma muestra en caso de caducar el timer
- Se hace el *backoff* normal
- No se emplea el valor modificado por *backoff* para actualizar R
- Cuando haya una estimación sin pérdidas se continúa
- *Must*



Algoritmo de Nagle

Objetivo:

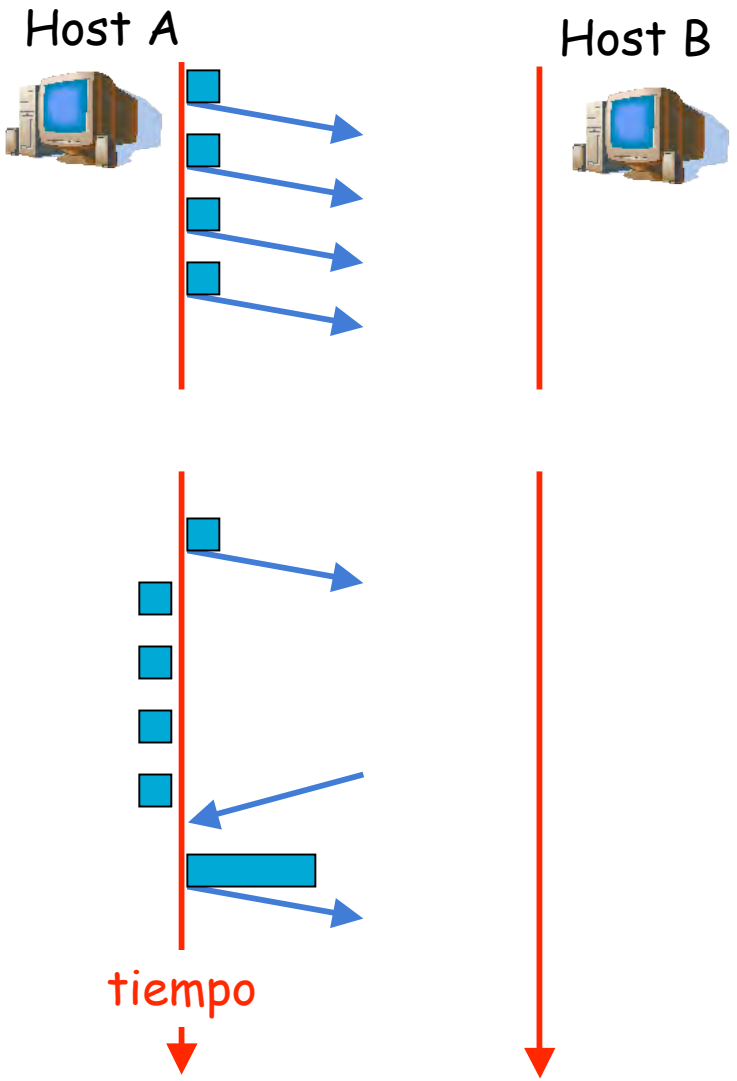
- Que no haya tantos paquetes pequeños de 40+1 bytes

Algoritmo:

- Si hay datos enviados sin confirmar, acumular
- Enviar cuando llegue ACK
- O cuando se tenga un segmento del MSS

Self-clocked (autoregulado)

- Cuanto más rápido llegan los ACKs más rápido se envían los datos



Contenido

- Internet Protocol
  - Características
  - IP en LAN Ethernet (ARP)
  - Direccionamiento (subredes, Proxy-ARP, CIDR)
  - Fragmentación y reensamblado
  - ICMP
- Enrutamiento en Internet
  - Distance-Vector
  - Link-State
  - Path-Vector
- UDP
- TCP
  - Características
  - Gestión de conexiones
  - Control de flujo
  - Retransmisiones
  - **Control de congestión**

Lecturas recomendadas

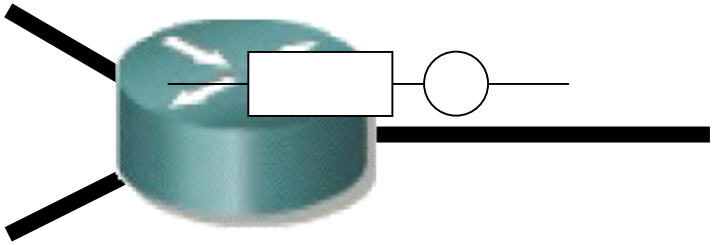
- [Kurose05] 3.6, 3.7.1
- [Stevens] 20.6, 21.6-21.8

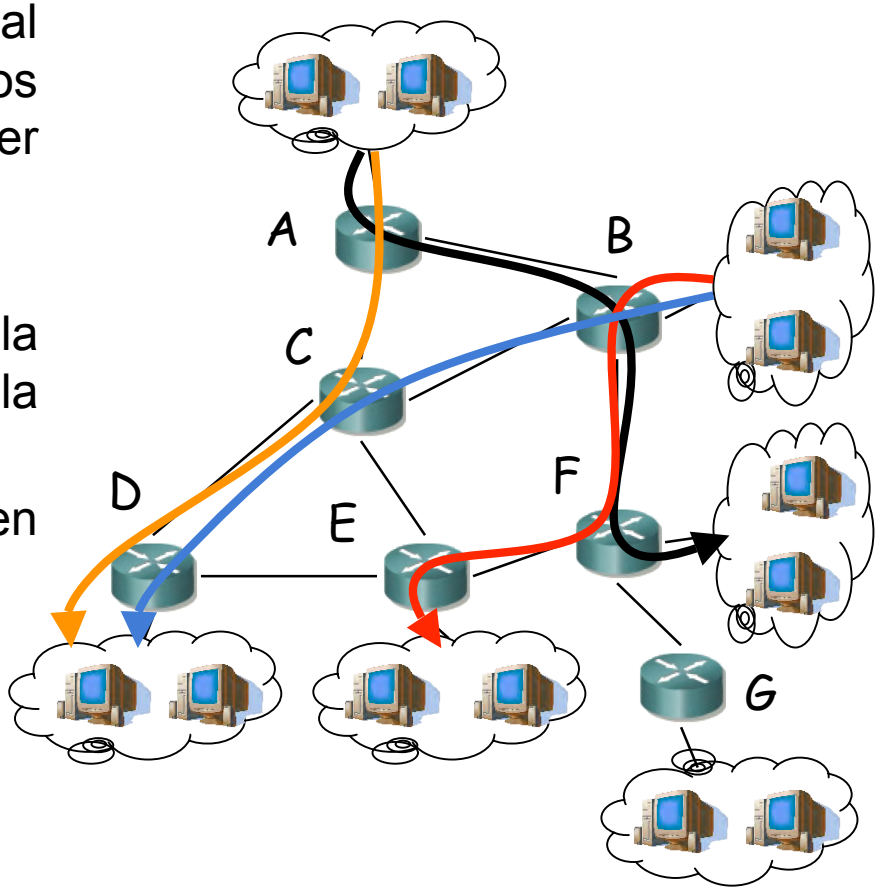
Contenido

- Introducción
- Cómo detectar la congestión
- Cómo controlarla
- *Slow Start*
- *Congestion avoidance*
- *Fast recovery, fast retransmit*

Congestión

- “Demasiado” tráfico en un enlace hace que la “calidad” se “degrade”: Congestión
- Cuando el tráfico se acerca al límite de la capacidad los routers comienzan a perder paquetes
- Un servidor con cola
- También limitado por la capacidad de reenvío de la CPU
- No hay suficiente memoria en los *buffers*
- Aunque la hubiera no sería la solución (excesivo retardo y retransmisiones)





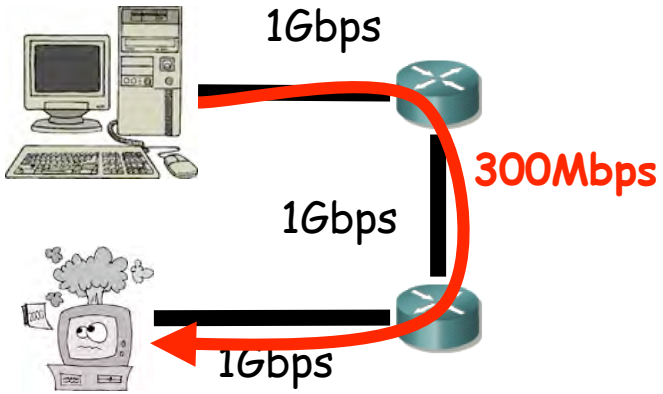
Control de Congestión y de Flujo

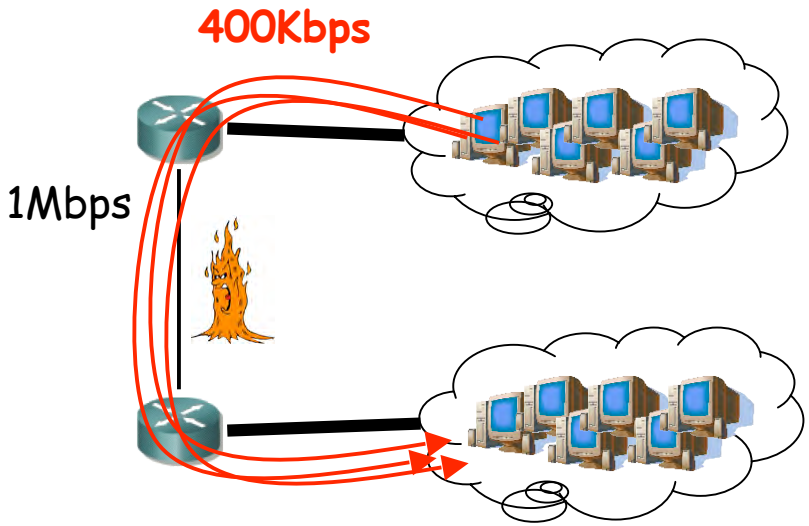
Control de congestión

- La red no es capaz de cursar todo el tráfico
- Es un problema global
- Depende del comportamiento de hosts y routers

Control de flujo

- Diferente
- Entre un transmisor y un receptor
- Evitar que el transmisor sature al receptor





Control de Congestión en TCP

Procedimiento:

- No hay congestión ⇒ Aumentar tasa
- Congestión ⇒ Disminuir tasa
- ¿¡ Tan sencillo !?

¿Cómo detectar la congestión?

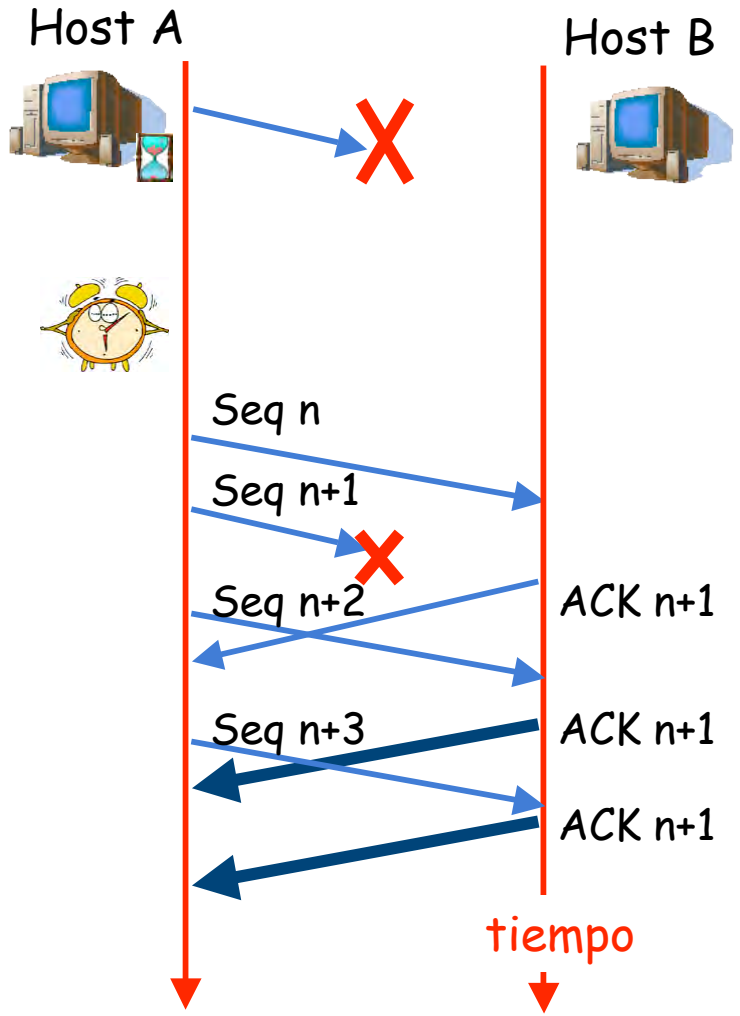
¿Cómo reducir la tasa?

¿Cómo aumentar la tasa?

¿Cómo detectar la congestión?

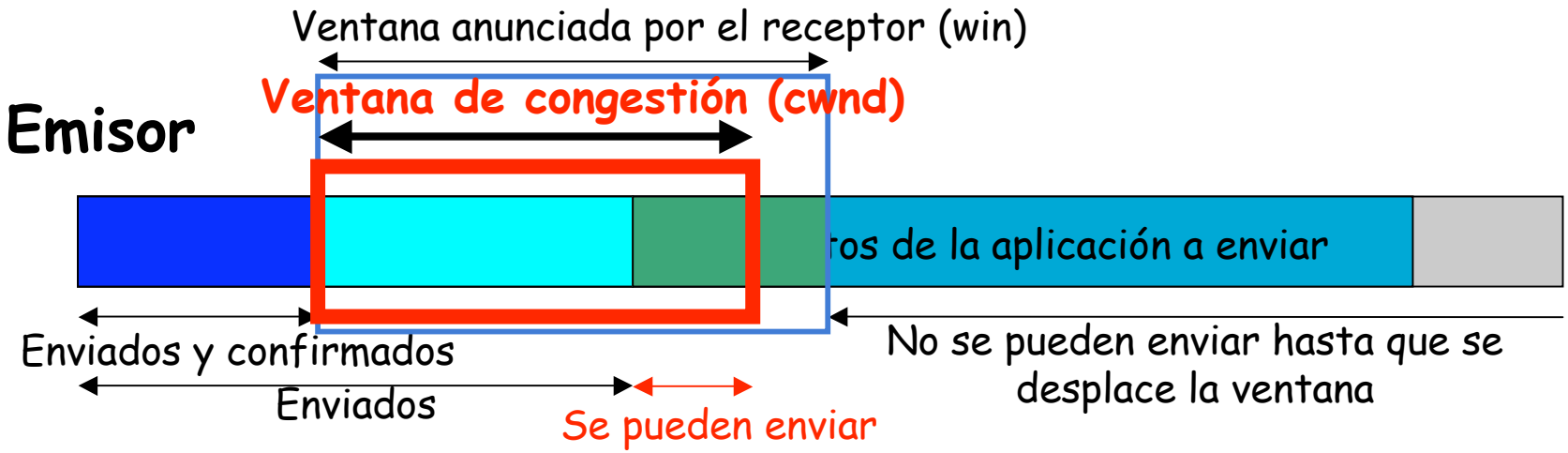
Pérdidas

- Por errores en el canal: Poco probable
- Descarte por desbordamiento de *buffer* ⇒ Congestión
- Detección de pérdidas
  - *Timeout*
  - ACKs duplicados



¿Cómo controlar la velocidad?

- Ventana de congestión
- Se puede enviar = mín{ Disponibles, win, cwnd }
- Tasa máxima = mín{ win, cwnd } / RTT



Slow Start

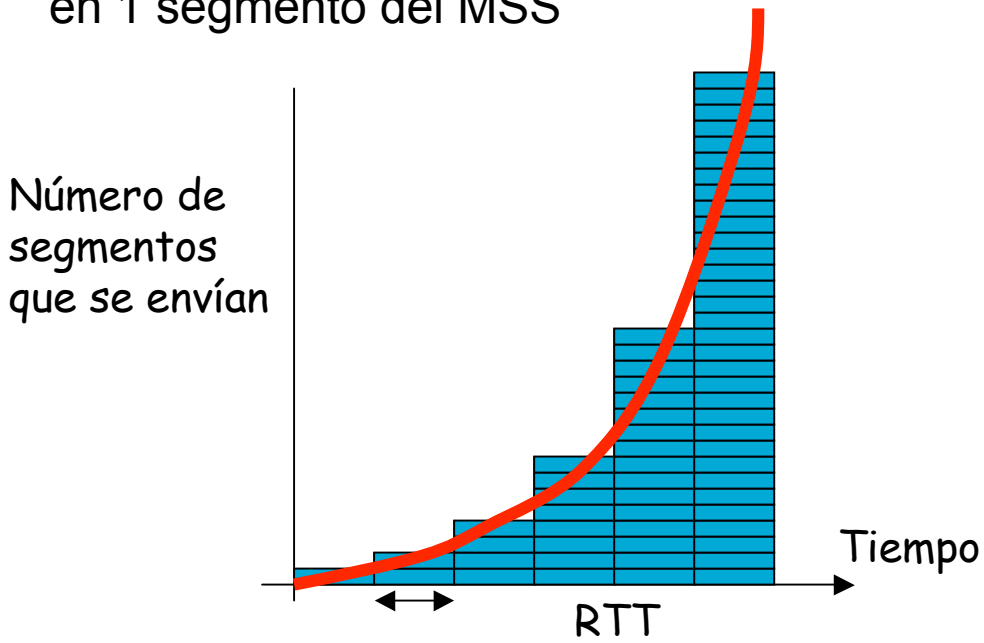
- Al comienzo de la conexión no se conoce el estado de congestión de la red
- El protocolo de ventana deslizante permite enviar hasta el tamaño de la ventana anunciada (*win*)
- *Win* depende solo de los buffers del receptor
- Es una ráfaga que puede crear congestión y pérdidas
- Mejor "probar" poco a poco el estado de la red aumentando "lentamente" la tasa de transmisión

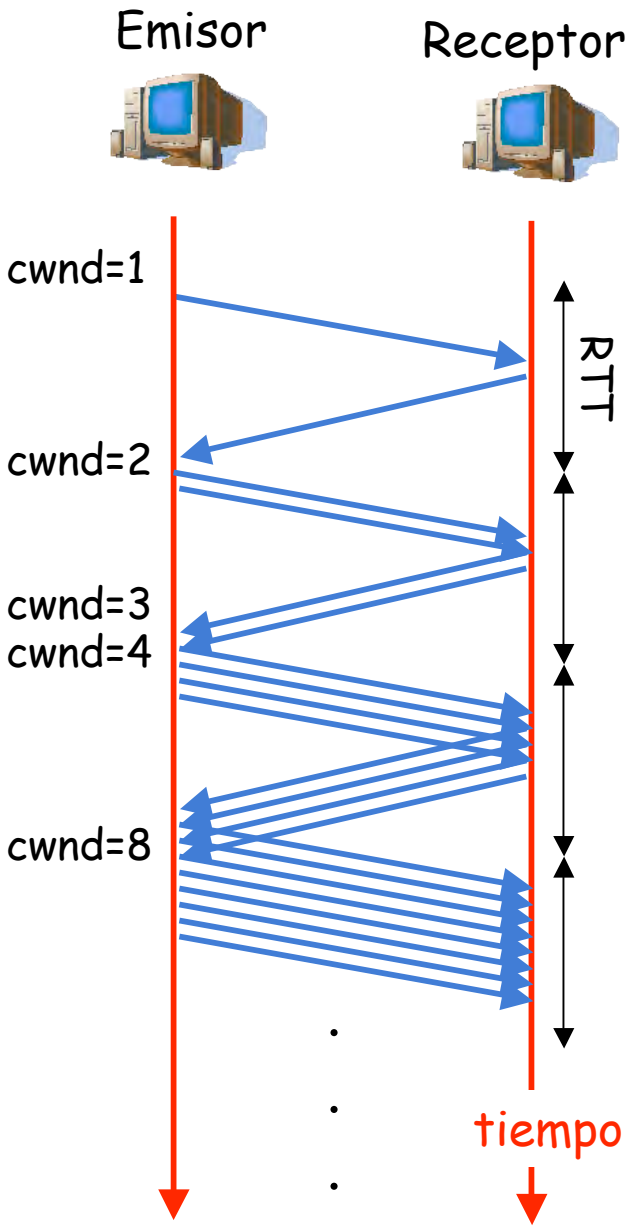
Slow start

- En realidad la tasa aumenta exponencialmente (no es tan "slow")
- Es control de flujo impuesto por el emisor a través de la ventana de congestión
- 4.3BSD Tahoe (1988)

Slow Start

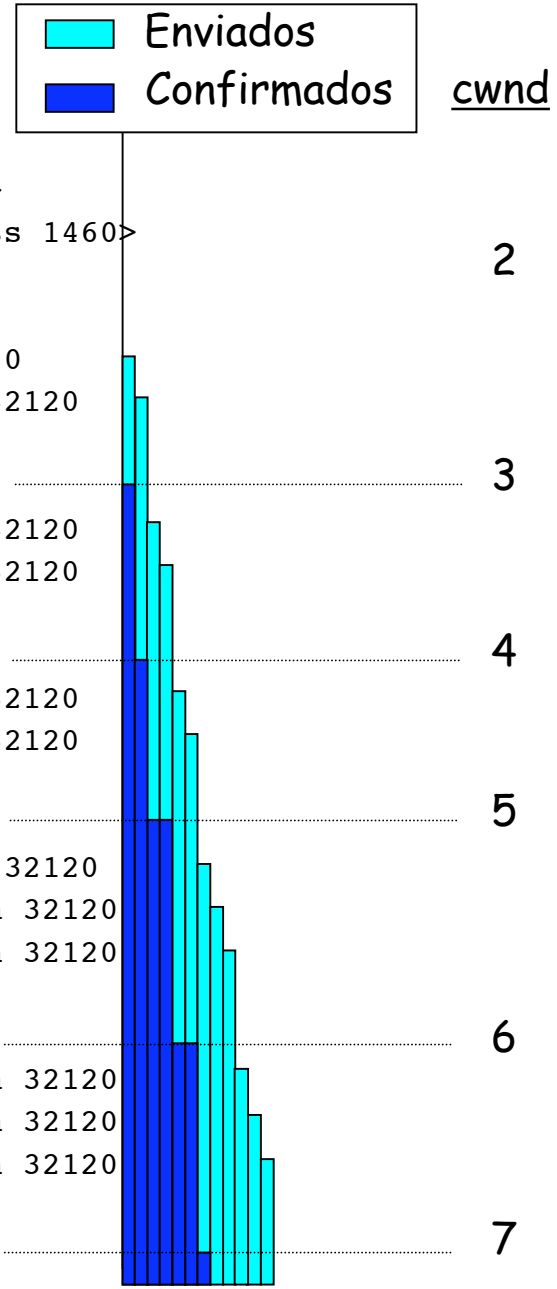
- Al comenzar cwnd = 1 segmento del MSS
  - La ventana es mín{win, cwnd}
  - Al principio solo se puede enviar un segmento
  - En realidad (RFC 2581) debe ser: $cwnd_{inicial} < 2xMSS$
- Por cada ACK (que confirme nuevos datos) que se reciba se incrementa cwnd en 1 segmento del MSS





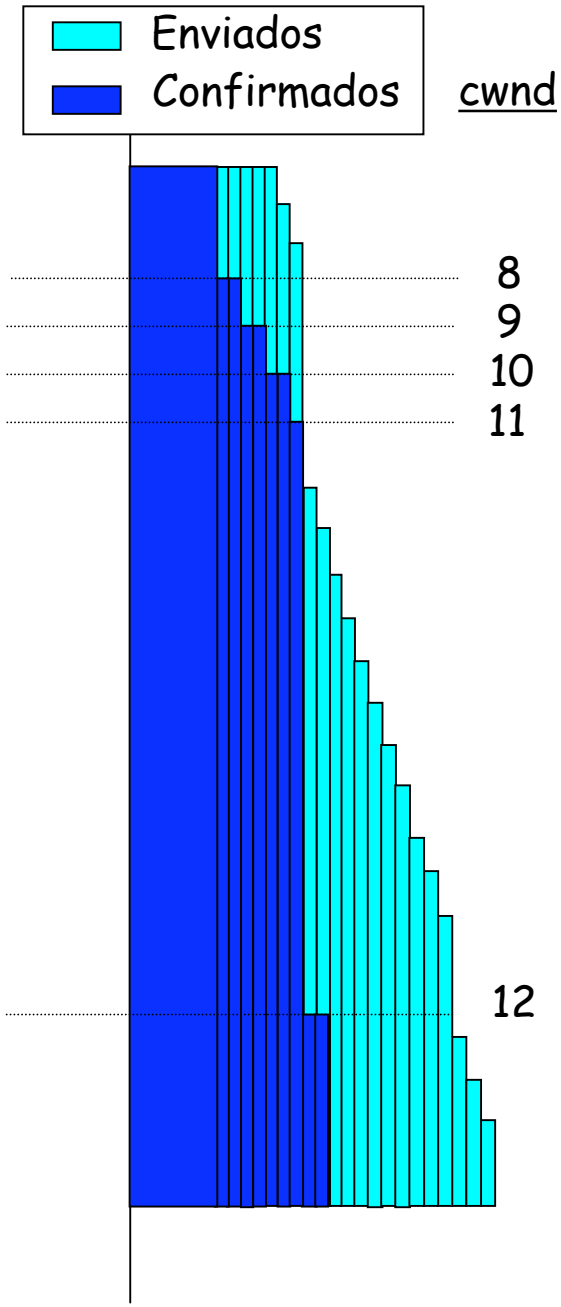
Ejemplo de *slow start*

```
1.1.1.11.2823 > 1.1.1.12.1509: S 0:0(0) win 32120 <mss 1460>
1.1.1.12.1509 > 1.1.1.11.2823: S 0:0(0) ack 1 win 32120  <mss 1460>
1.1.1.11.2823 > 1.1.1.12.1509: . 1:1(0) ack 1 win 32120

1.1.1.12.1509 > 1.1.1.11.2823: P 1:1449(1448) ack 1 win 32120
1.1.1.12.1509 > 1.1.1.11.2823: P 1449:2897(1448) ack 1 win 32120

1.1.1.11.2823 > 1.1.1.12.1509: . 1:1(0) ack 1449 win 31856
1.1.1.12.1509 > 1.1.1.11.2823: P 2897:4345(1448) ack 1 win 32120
1.1.1.12.1509 > 1.1.1.11.2823: P 4345:5793(1448) ack 1 win 32120

1.1.1.11.2823 > 1.1.1.12.1509: . 1:1(0) ack 2897 win 31856
1.1.1.12.1509 > 1.1.1.11.2823: P 5793:7241(1448) ack 1 win 32120
1.1.1.12.1509 > 1.1.1.11.2823: P 7241:8689(1448) ack 1 win 32120

1.1.1.11.2823 > 1.1.1.12.1509: . 1:1(0) ack 5793 win 31856
1.1.1.12.1509 > 1.1.1.11.2823: P 8689:10137(1448) ack 1 win 32120
1.1.1.12.1509 > 1.1.1.11.2823: P 10137:11585(1448) ack 1 win 32120
1.1.1.12.1509 > 1.1.1.11.2823: P 11585:13033(1448) ack 1 win 32120

1.1.1.11.2823 > 1.1.1.12.1509: . 1:1(0) ack 8689 win 31856
1.1.1.12.1509 > 1.1.1.11.2823: P 13033:14481(1448) ack 1 win 32120
1.1.1.12.1509 > 1.1.1.11.2823: P 14481:15929(1448) ack 1 win 32120
1.1.1.12.1509 > 1.1.1.11.2823: P 15929:17377(1448) ack 1 win 32120

1.1.1.11.2823 > 1.1.1.12.1509: . 1:1(0) ack 10137 win 31856
```



Ejemplo de *slow start*

```
1.1.1.12.1509 > 1.1.1.11.2823: P 17377:18825(1448) ack 1 win 32120
1.1.1.12.1509 > 1.1.1.11.2823: P 18825:20001(1176) ack 1 win 32120
1.1.1.11.2823 > 1.1.1.12.1509: . 1:1(0) ack 13033 win 31856
1.1.1.11.2823 > 1.1.1.12.1509: . 1:1(0) ack 15929 win 31856
1.1.1.11.2823 > 1.1.1.12.1509: . 1:1(0) ack 18825 win 31856
1.1.1.11.2823 > 1.1.1.12.1509: . 1:1(0) ack 20001 win 31856

1.1.1.12.1509 > 1.1.1.11.2823: P 20001:21449(1448) ack 1 win 32120
1.1.1.12.1509 > 1.1.1.11.2823: P 21449:22897(1448) ack 1 win 32120
1.1.1.12.1509 > 1.1.1.11.2823: P 22897:24345(1448) ack 1 win 32120
1.1.1.12.1509 > 1.1.1.11.2823: P 24345:25793(1448) ack 1 win 32120
1.1.1.12.1509 > 1.1.1.11.2823: P 25793:27241(1448) ack 1 win 32120
1.1.1.12.1509 > 1.1.1.11.2823: P 27241:28689(1448) ack 1 win 32120
1.1.1.12.1509 > 1.1.1.11.2823: P 28689:30137(1448) ack 1 win 32120
1.1.1.12.1509 > 1.1.1.11.2823: P 30137:31585(1448) ack 1 win 32120
1.1.1.12.1509 > 1.1.1.11.2823: P 31585:33033(1448) ack 1 win 32120
1.1.1.12.1509 > 1.1.1.11.2823: P 33033:34481(1448) ack 1 win 32120
1.1.1.12.1509 > 1.1.1.11.2823: P 34481:35929(1448) ack 1 win 32120

1.1.1.11.2823 > 1.1.1.12.1509: . 1:1(0) ack 22897 win 31856
1.1.1.12.1509 > 1.1.1.11.2823: . 35929:37377(1448) ack 1 win 32120
1.1.1.12.1509 > 1.1.1.11.2823: . 37377:38825(1448) ack 1 win 32120
1.1.1.12.1509 > 1.1.1.11.2823: . 38825:40273(1448) ack 1 win 32120
```



Control de Congestión

"Demasiado" tráfico en un enlace hace que la "calidad" se "degrade": Congestión

Procedimiento:

- No hay congestión ⇒ Aumentar tasa
- Congestión ⇒ Disminuir tasa

¿Cómo detectar la congestión?

- Descarte por desbordamiento de *buffer* ⇒ Congestión
- Detección de pérdidas
  - *Timeout*
  - ACKs duplicados

¿Cómo aumentar la tasa?

- *Slow Start*
- Al comenzar cwnd = 1 segmento del MSS
  - La ventana es mín{win, cwnd}
  - Al principio solo se puede enviar un segmento
  - En realidad (RFC 2581) debe ser: $cwnd_{inicial} < 2xMSS$
- Por cada ACK (que confirme nuevos datos) que se reciba se incrementa cwnd en 1 segmento del MSS

¿Cómo reducir la tasa?

Congestion avoidance

- La conexión comienza con cwnd=1
- *Slow Start* incrementa exponencialmente el tamaño de cwnd
- Ante detección de congestión mediante pérdidas:
  - Reducir la tasa de envío reduciendo la ventana de congestión
  - Diferente reducción ante detección por *timeout* o por ACKs duplicados
- Algoritmo...

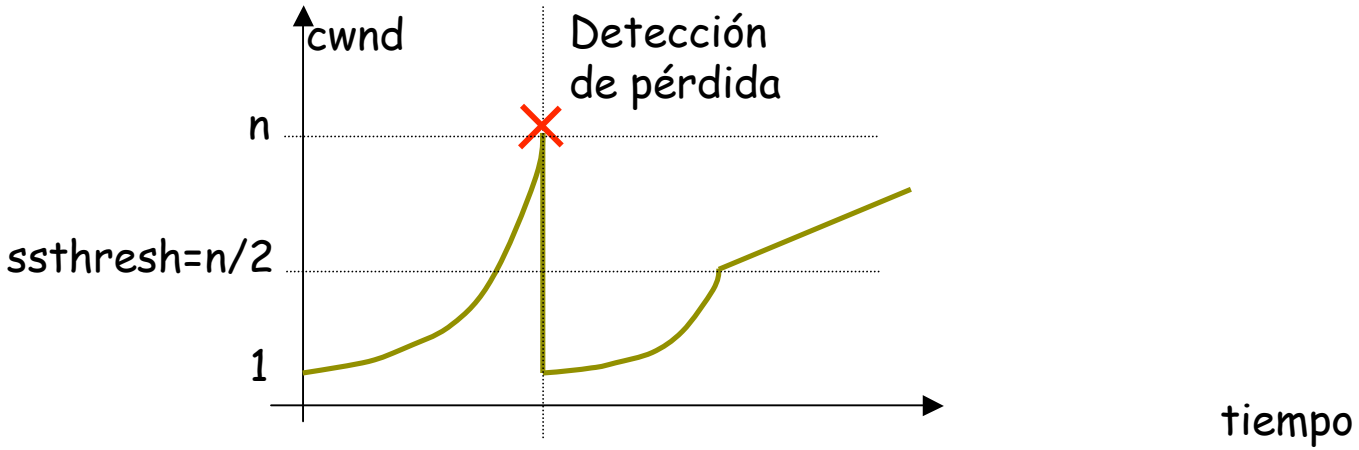
Congestion avoidance

TCP Tahoe

- Inicialización: ***cwnd***=1, ***ssthresh***=65535
- *Slow Start* (…)
- Si se detecta una pérdida (…):
  - *ssthresh* = mín{ cwnd, win }/2
    [RFC2581] ssthresh = máx {FlightSize/2, 2} (…)

  Por ***Timeout***:
  - cwnd=1, comienza *Slow-start* de nuevo (…)

  Nuevos datos son confirmados:
  - Si cwnd<=ssthresh : *Slow-start*, cwnd+=1 (crecimiento exp.) (…)
  - Si cwnd>ssthresh : ***Congest. avoidance***, cwnd+=1/cwnd (crecimiento lineal) (…)



Ejemplo (I)

```
753.246362 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: S 0:0(0) win 32120
753.246536 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: S 0:0(0) ack 1

753.468594 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 1:205(204) ack 1
753.468750 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 205:409(204) ack 1

754.291021 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 205
754.291137 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 409:613(204) ack 1
754.291257 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 613:817(204) ack 1
754.746127 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 409
754.746234 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 817:1021(204) ack 1
754.746353 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 1021:1225(204) ack 1
755.832827 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 817
755.832948 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 1225:1429(204) ack 1
755.833066 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 1429:1633(204) ack 1
755.833182 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 1633:1837(204) ack 1
756.327987 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 1225
756.328105 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 1837:2041(204) ack 1
756.328220 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 2041:2245(204) ack 1
756.328333 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 2245:2449(204) ack 1

760.697798 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 1225:1429(204) ack 1

761.796804 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 1429
```

cwnd	ssthresh
2	
3	
4	
5	
6	
1	3
2	

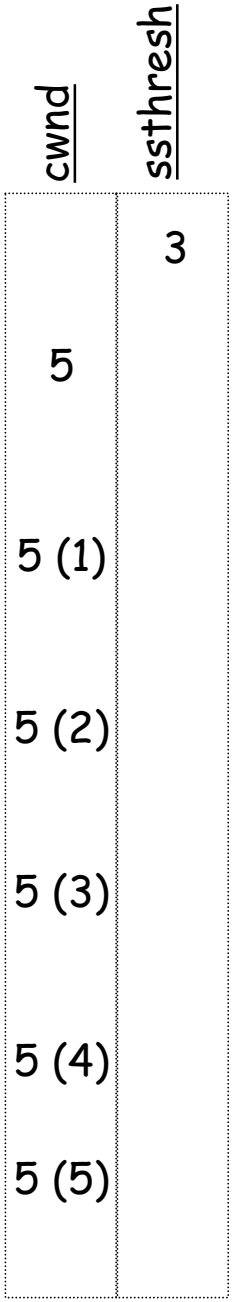
Ejemplo (II)

```
761.796932 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 1429:1633(204) ack 1
761.797054 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 1633:1837(204) ack 1
762.838579 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 1837

762.838691 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 1837:2041(204) ack 1
762.838807 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 2041:2245(204) ack 1
762.838923 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 2245:2449(204) ack 1

764.101234 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 2245

764.827096 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 2449

764.827200 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 2449:2653(204) ack 1
764.827315 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 2653:2857(204) ack 1
764.827428 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 2857:3061(204) ack 1
764.827541 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 3061:3265(204) ack 1
766.320277 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 2857

766.320406 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 3265:3469(204) ack 1
766.320523 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 3469:3673(204) ack 1
766.758919 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 3265

766.759024 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 3673:3877(204) ack 1
766.759141 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 3877:4081(204) ack 1
767.859805 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 3673
```

cwnd	ssthresh
	3
3	
4	
4 (1)	
4 (2)	
4 (3)	
4 (4)	

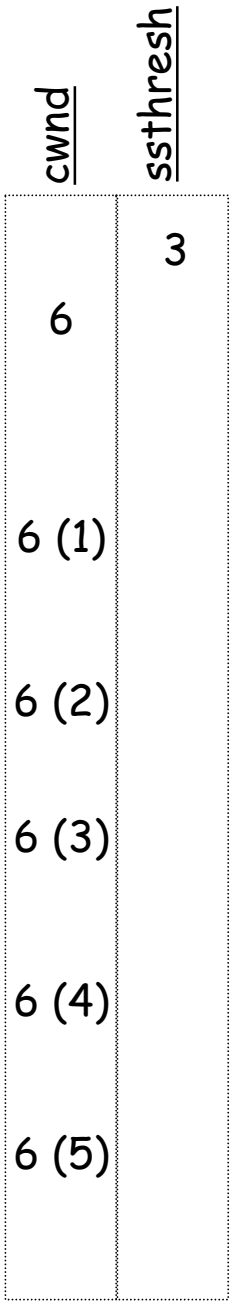
Ejemplo (III)

```
767.859926 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 4081:4285(204) ack 1
767.860043 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 4285:4489(204) ack 1
768.359065 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 4081

768.359188 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 4489:4693(204) ack 1
768.359305 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 4693:4897(204) ack 1
768.359418 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 4897:5101(204) ack 1
768.899165 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 4489

768.899311 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 5101:5305(204) ack 1
768.899429 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 5305:5509(204) ack 1
770.122698 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 4897

770.122858 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 5509:5713(204) ack 1
770.122975 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 5713:5917(204) ack 1
770.614382 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 5305

770.614528 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 5917:6121(204) ack 1
770.614644 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 6121:6325(204) ack 1
771.347724 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 5509

771.347856 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 6325:6529(204) ack 1
771.659393 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 5917
```



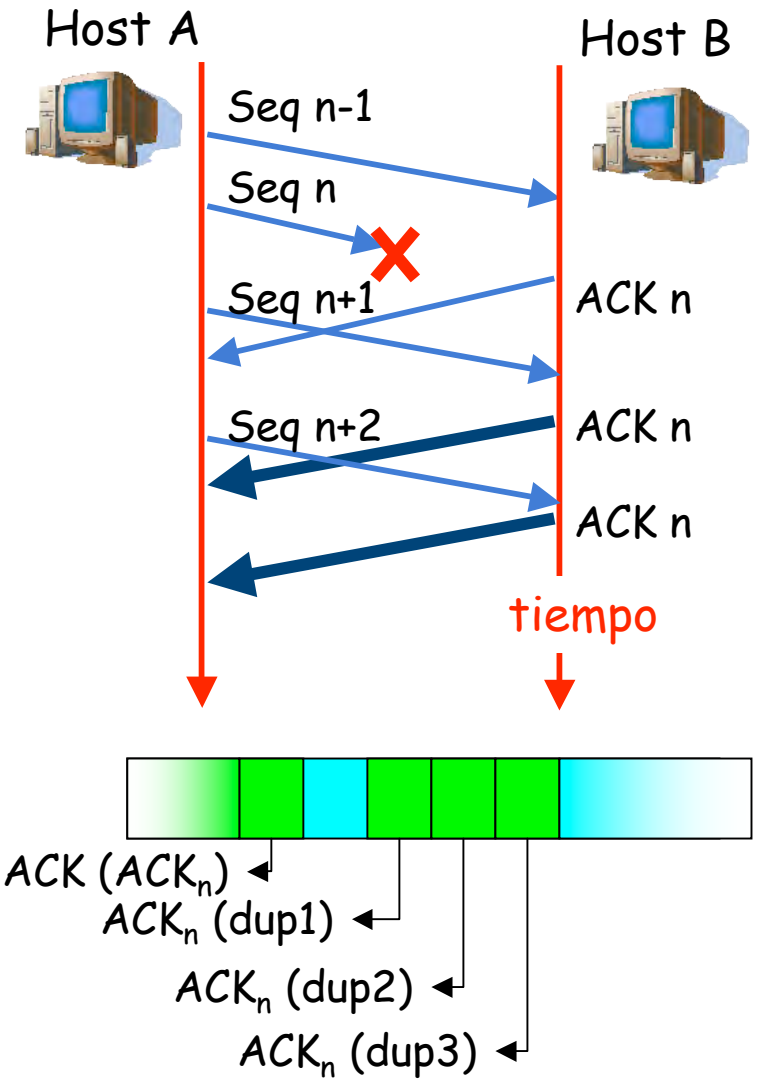
Ejemplo (y IV)

```
771.659497 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 6529:6733(204) ack 1
771.659613 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 6733:6937(204) ack 1
772.159445 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 6325

772.159599 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 6937:7141(204) ack 1
772.159715 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 7141:7345(204) ack 1
772.159829 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 7345:7549(204) ack 1
772.699491 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 6733

772.699660 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 7549:7753(204) ack 1
772.699781 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 7753:7957(204) ack 1
1773.397932 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 6937

773.398089 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 7957:8161(204) ack 1
773.919587 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 7345

773.919734 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 8161:8365(204) ack 1
773.919851 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 8365:8569(204) ack 1
774.409619 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 7753

774.409770 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 8569:8773(204) ack 1
774.409886 eth0 > 1.1.1.61.3014 > 10.1.11.1.1069: P 8773:8977(204) ack 1
774.909675 eth0 < 10.1.11.1.1069 > 1.1.1.61.3014: . 1:1(0) ack 8161
```



Fast recovery, Fast retransmit

- [RFC 2581] "*Out-of-order data segments should be acknowledged inmediately in order to accelerate loss recovery*"
- Los ACKs duplicados:
  - Por desordenamiento o pérdidas
  - Desordenamiento breve, no más de 2 ACKs duplicados
  - Si recibimos 3 o más supondremos ⇒ pérdida
  - Cada uno indica que un segmento ha llegado
- [RFC 2581] "*...The receiver should send an inmediate ACK when it receives a data segment that fills in all or part od a gap in the sequence space*"



Fast recovery, Fast retransmit

TCP Reno

- **TCP *Reno***
- Inicialización: *cwnd*=1, *ssthresh*=65535
- *Slow Start*
- Si se detecta una pérdida:
  - *ssthresh* = mín{ cwnd, win }/2
    [RFC2581] ssthresh = máx {FlightSize/2, 2}

  Por **Timeout:**
  - cwnd=1, comienza *Slow-start* de nuevo

  Si se detecta mediante un 3º ACK duplicado
  - Retransmitir el segmento que parece faltar (***Fast Retransmit***)
  - A continuación ***Fast Recovery*** en vez de *Slow-start*
  - Porque siguen llegando segmentos al otro extremo
  - **cwnd = ssthresh + 3** (los 3 duplicados indican que 3 han llegado)
  - Por cada ACK duplicado adicional cnwd+=1 y transmitir si lo permite *cwnd*
  - ACK que confirme nuevos datos (se ha completado el hueco): cwnd=ssthresh y pasa a hacer *Congestion avoidance*

Ejemplo (I)

```
1.1.1.15.1069 > 10.1.8.251.5000: S 0:0(0) win 32120
10.1.8.251.5000 > 1.1.1.15.1069: S 0:0(0) ack 1 win 31680 (= 33MSS)
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 1 win 32120


10.1.8.251.5000 > 1.1.1.15.1069: P 1:949(948) ack 1 win 32232
10.1.8.251.5000 > 1.1.1.15.1069: P 949:1897(948) ack 1 win 32232
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 949 win 31284

10.1.8.251.5000 > 1.1.1.15.1069: P 1897:2845(948) ack 1 win 32232
10.1.8.251.5000 > 1.1.1.15.1069: P 2845:3793(948) ack 1 win 32232
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 1897 win 32232

10.1.8.251.5000 > 1.1.1.15.1069: P 3793:4741(948) ack 1 win 32232
10.1.8.251.5000 > 1.1.1.15.1069: P 4741:5689(948) ack 1 win 32232
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 3793 win 31284

10.1.8.251.5000 > 1.1.1.15.1069: P 5689:6637(948) ack 1 win 32232
10.1.8.251.5000 > 1.1.1.15.1069: P 6637:7585(948) ack 1 win 32232
10.1.8.251.5000 > 1.1.1.15.1069: P 7585:8533(948) ack 1 win 32232
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 5689 win 31284
```

...

cwnd	ssthresh
2	
3	
4	
5	
6	

Ejemplo (II)

		cwnd	ssthresh
10.1.8.251.5000 > 1.1.1.15.1069: P 75841:76789(948) ack 1 win 32232			
10.1.8.251.5000 > 1.1.1.15.1069: . 76789:77737(948) ack 1 win 32232			
10.1.8.251.5000 > 1.1.1.15.1069: . 77737:78685(948) ack 1 win 32232			
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284		31	
10.1.8.251.5000 > 1.1.1.15.1069: . 78685:79633(948) ack 1 win 32232			
10.1.8.251.5000 > 1.1.1.15.1069: . 79633:80581(948) ack 1 win 32232			
10.1.8.251.5000 > 1.1.1.15.1069: . 80581:81529(948) ack 1 win 32232			
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284	DUP 1		
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284	DUP 2		
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284	**DUP 3**		15
10.1.8.251.5000 > 1.1.1.15.1069: P 52141:53089(948) ack 1 win 32232			
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284		18	
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284		19	
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284		20	
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284		21	
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284		22	
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284		23	
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284		24	
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284		25	
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284		26	
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284		27	

Ejemplo (III)

Trace	cwnd	ssthresh
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284	28	15
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284	29	
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284	30	
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284	31	
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284	32	
10.1.8.251.5000 > 1.1.1.15.1069: . 81529:82477(948) ack 1 win 32232		
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284	33	
10.1.8.251.5000 > 1.1.1.15.1069: . 82477:83425(948) ack 1 win 32232		
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284	34	
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284	35	
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284	36	
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284	37	
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284	38	
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284	39	
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284	40	
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284	41	
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284	42	
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284	43	
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 52141 win 31284	44	
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 81529 win 8532	**15**	

win = 33

win = 9

Ejemplo (y IV)

```
10.1.8.251.5000 > 1.1.1.15.1069: . 83425:84373(948) ack 1 win 32232
10.1.8.251.5000 > 1.1.1.15.1069: . 84373:85321(948) ack 1 win 32232
10.1.8.251.5000 > 1.1.1.15.1069: . 85321:86269(948) ack 1 win 32232
10.1.8.251.5000 > 1.1.1.15.1069: . 86269:87217(948) ack 1 win 32232
10.1.8.251.5000 > 1.1.1.15.1069: . 87217:88165(948) ack 1 win 32232
10.1.8.251.5000 > 1.1.1.15.1069: . 88165:89113(948) ack 1 win 32232
10.1.8.251.5000 > 1.1.1.15.1069: . 89113:90061(948) ack 1 win 32232

1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 81529 win 24648

10.1.8.251.5000 > 1.1.1.15.1069: . 90061:91009(948) ack 1 win 32232
10.1.8.251.5000 > 1.1.1.15.1069: . 91009:91957(948) ack 1 win 32232
10.1.8.251.5000 > 1.1.1.15.1069: . 91957:92905(948) ack 1 win 32232
10.1.8.251.5000 > 1.1.1.15.1069: . 92905:93853(948) ack 1 win 32232
10.1.8.251.5000 > 1.1.1.15.1069: . 93853:94801(948) ack 1 win 32232
10.1.8.251.5000 > 1.1.1.15.1069: . 94801:95749(948) ack 1 win 32232
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 83425 win 31284

10.1.8.251.5000 > 1.1.1.15.1069: . 95749:96697(948) ack 1 win 32232
10.1.8.251.5000 > 1.1.1.15.1069: . 96697:97645(948) ack 1 win 32232
10.1.8.251.5000 > 1.1.1.15.1069: . 97645:98593(948) ack 1 win 32232
1.1.1.15.1069 > 10.1.8.251.5000: . 1:1(0) ack 85321 win 31284
```

…