



URI

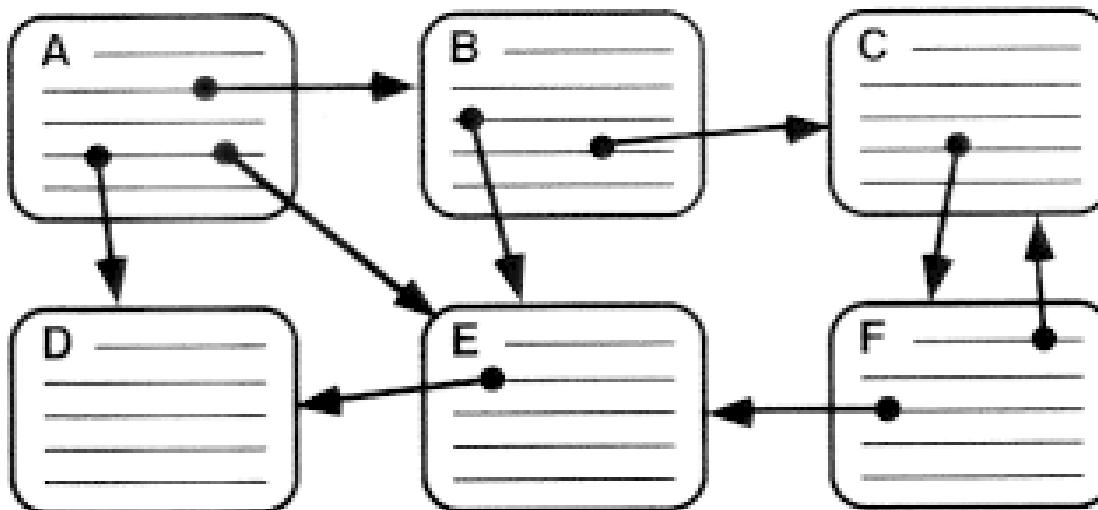
HTTP 1, 2

Proxies

Cookies

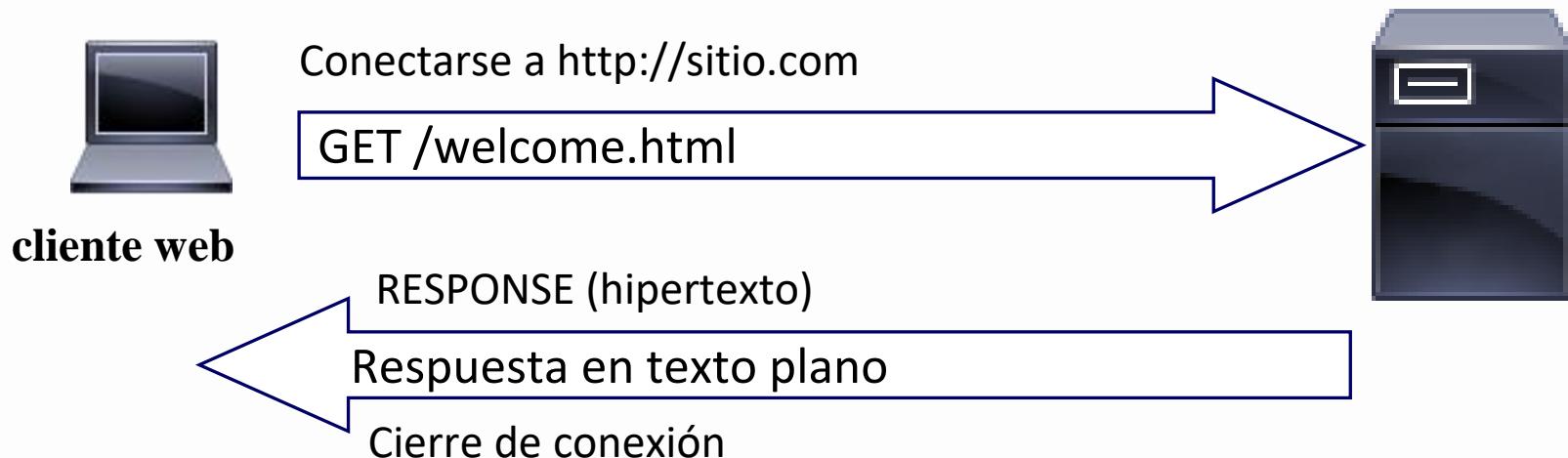
# Previo a HTTP

- Vía FTP se obtenía un documento o texto
- En base a las referencias se debía acceder a otro sitio FTP y descargar el/los textos deseados
- Surge la necesidad de aplicaciones basadas en "hipertextos"



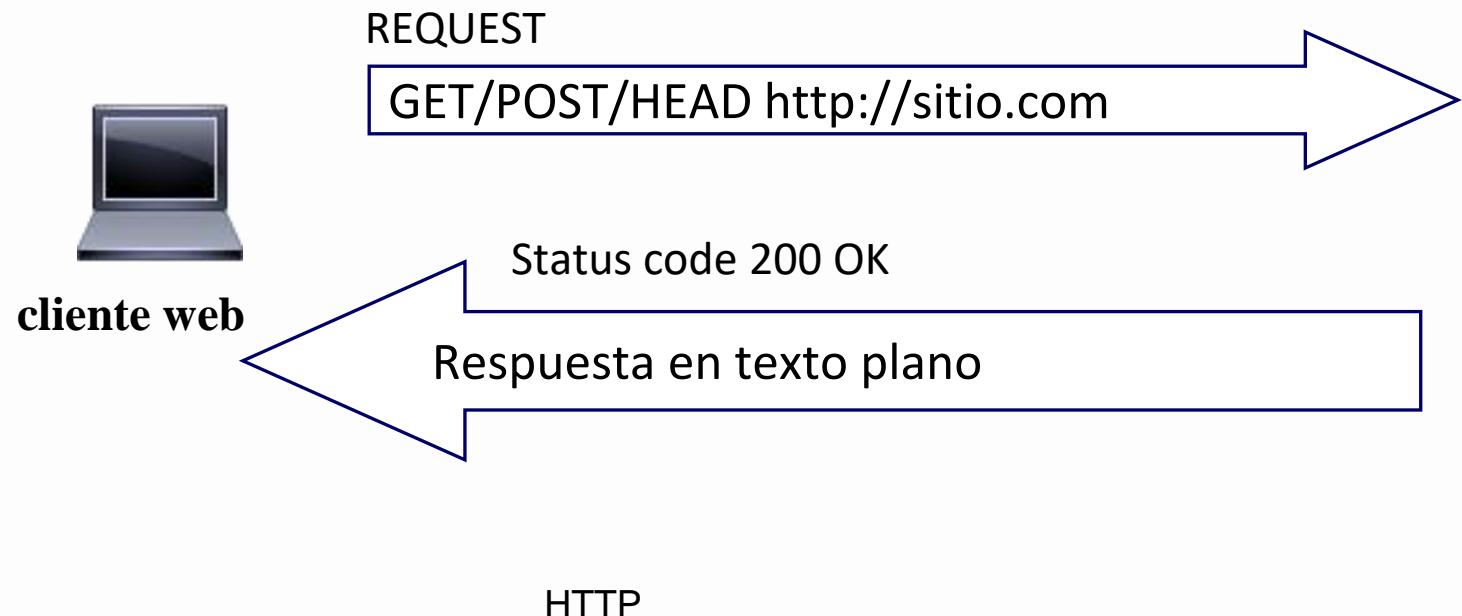
# HTTP 0.9 (1991)

- ◆ Protocolo basado en texto de recuperación de información
- ◆ Mecanismo *request – response*
- ◆ Contenido estático
- ◆ No mantiene estado (no establece una sesión)
- ◆ Orientado a "línea de comando"



# HTTP 1.0 (1996)

- ◆ "Browser friendly"
- ◆ Comandos como GET, POST, HEAD
- ◆ Códigos de estado
- ◆ No solo hipertexto

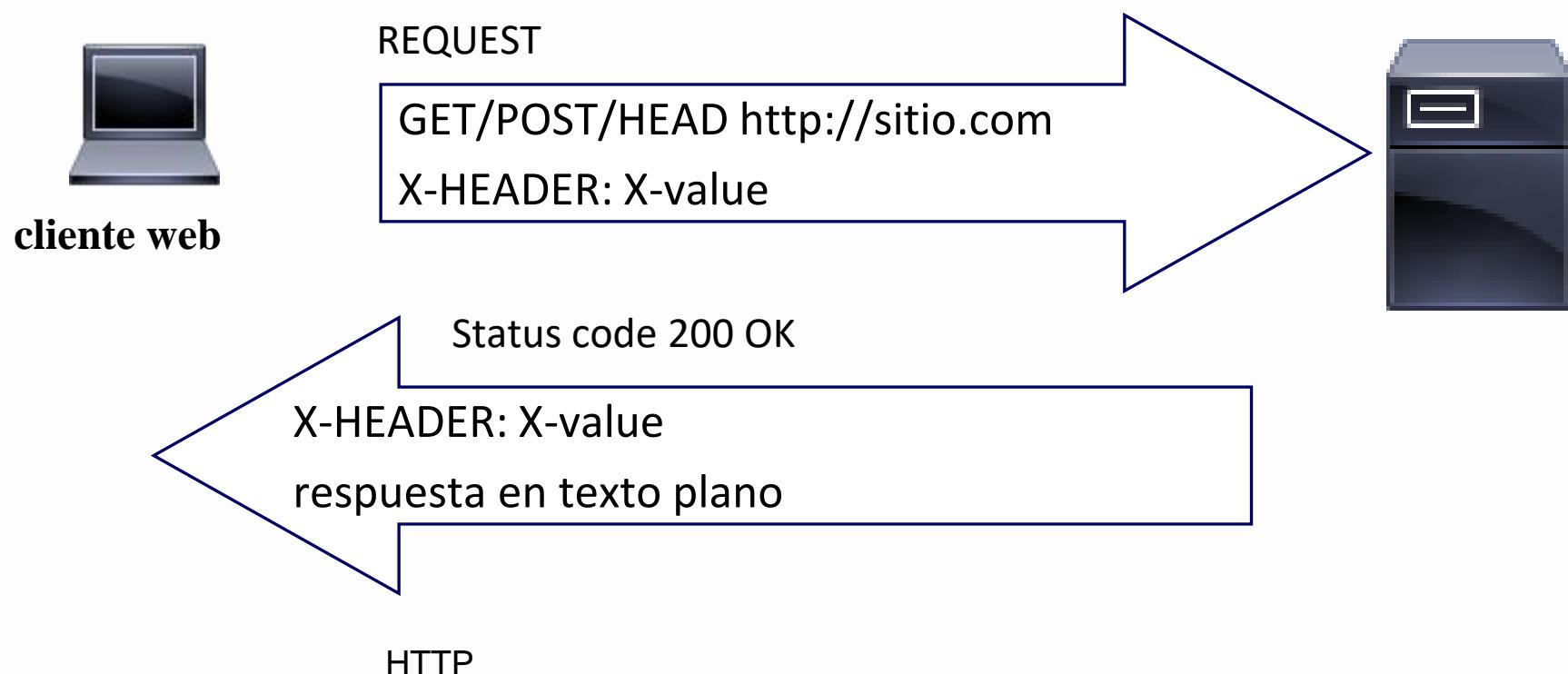


Un request por  
CADA recurso (no  
confundir con  
"página web")

# HTTP 1.1 (1999)

- ◆ Cabeceras en las peticiones
- ◆ Métodos PUT, DELETE, TRACE, OPTIONS
- ◆ Negociación de contenido
- ◆ Soporte de cache
- ◆ Conexiones persistentes

Cada recurso se identifica con una URI (Uniform Resource Identifier )



# HTTP/2

- ◆ Protocolo binario
- ◆ Compresión de headers
- ◆ Multiplexación de recursos
- ◆ Server push
- ◆ ...

<http://www.http2demo.io/>

# HTTP: recursos



- ◆ Un recurso es un bloque de información identificado por su URI
- ◆ Puede ser un archivo (físico) o generado por un programa (abstracto)
- ◆ URL: Uniform Resource Locator
- ◆ URN: Uniform Resource Name
- ◆ RFC 1630, 2396, 2718, 3305, 3986
- ◆ Internationalized Resource Identifiers (IRIs): RFC 3987

# URI (ver RFC 3986 3.3)



**<scheme>://<authority><path>?<query>**

- ◆ El path termina con el primer "?" o "#" o si no hay más caracteres
- ◆ Puede ser relativo o absoluto
- ◆ Si representa una aplicación puede recibir parámetros

- `http:www.example.org/path/name?param1;param2;param3`
- `/relative_path/name?user="..";pwd="..."`
- `.../../logo.png`

# URL:sintaxis

- ◆ Identifica un recurso por su ubicación (*location*)

Algunos sitios o recursos requieren autenticación

puerto (opcional si es 80)

<scheme>://<user>:<password>@<host>:<port>/<path>?<query>

Protocolo a utilizar: por ejemplo **HTTP, FTP, FILE**

**http://soyyo:miclave@www.unsitio.com:90**

**http://192.168.0.100**

**http://localhost:8080**

# URL: sintaxis



**Identificación del recurso  
dentro del servidor**

**Parámetros**

<scheme>://<user>:<password>@<host>:<port>/<path>?<query>

**http://soyyo:mclave@www.unsitio.com:90/index.html**

**http://soyyo:mclave@www.unsitio.com:90/pagZZ.html?width=1024&lang=es**

# URL: sintaxis

- ◆ Puede incluir al final un "fragmento"

<http://www.unsitio.com/intro.html#chapter1>

# URN

- ◆ Identifica un recurso por su nombre
- ◆ No implica que el recurso exista o cómo acceder a él

`urn:isbn:0132856204`

`urn:isbn:0000-0002-3C36-0000-Y-0000-0000-9`

`urn:uuid:6e8bc430-9c3a-11d9-9669-0800200c9a66`

`urn:www.apache.org:`

*<http://www.iana.org/assignments/urn-namespaces/urn-namespaces.xhtml>*

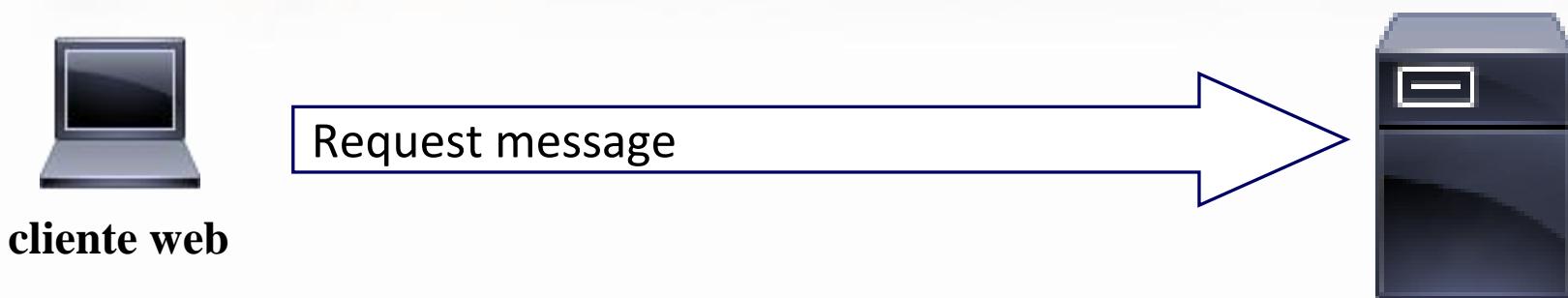
# URI Scheme

- ◆ URIs pueden ser usadas para acceder a recursos, ya sea por medio de URL o URN
- ◆ Ejemplo en html

```
<a href="/img/logo.png">  
<a href="urn:isbn:0453457513">
```

# Mensajes HTTP

- ♦ El cliente y el servidor intercambian mensajes HTTP
  - ♦ *Request message* (cliente → servidor)
  - ♦ *Response message* (servidor → cliente)



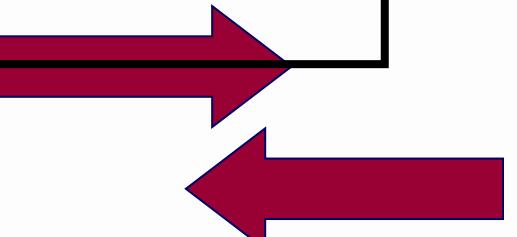
- ♦ Solicitudes (*Request*)
  - ♦ **GET**: Sigue una solicitud de un recurso al servidor
  - ♦ **HEAD**: Sigue una solicitud solo los headers del recurso.
  - ♦ **POST**: Envía información al servidor para ser procesada

# Ejemplo: HTTP - GET



cliente web

```
GET / HTTP/1.1
Host: www.clarin.com
User-Agent: Mozilla/5.0
Accept: text/html,application/xhtml+xml
Accept-Language: es-es
Accept-Encoding: gzip,deflate
Keep-Alive: 300
Connection: keep-alive
```



Start line

Headers

Sólo US-ASCII



Start line

Headers

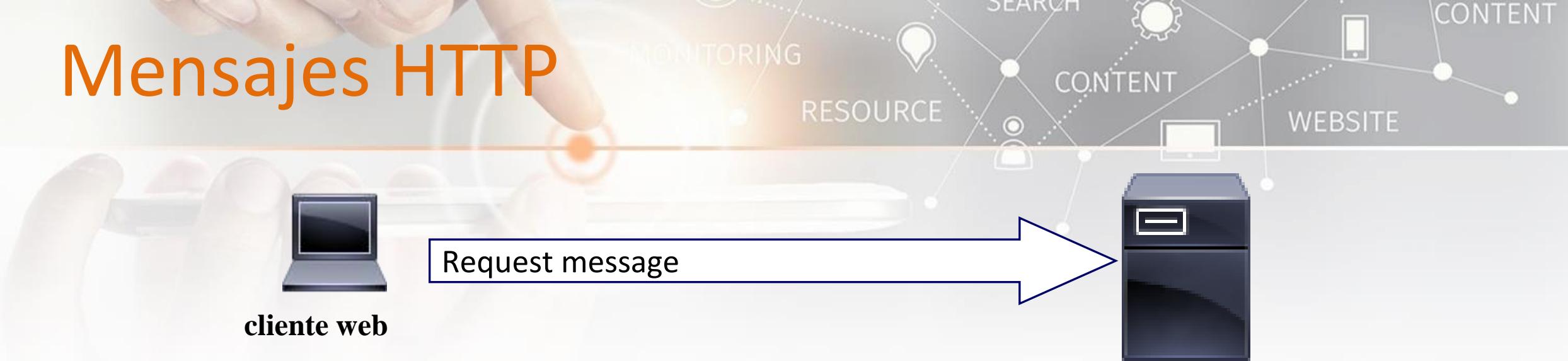
Body

```
HTTP/1.1 200 OK
Date: Mon, 15 Feb 2010 12:47:26 GMT
Server: Apache/2
Content-Length: 24495
Content-Type: text/html; charset=ISO-8859-1

<html><head>
<title>Alerta meteorológico en Capital</title>
.....
```

Sólo US-ASCII

# Mensajes HTTP



- ◆ **Solicitudes (*Request* ) menos utilizados**
  - ◆ **PUT**: Envía un recurso al servidor
  - ◆ **TRACE**: Analiza el recorrido de la solicitud.
  - ◆ **OPTIONS**: Consulta los métodos disponibles en el servidor.
  - ◆ **DELETE**: Elimina un recurso del servidor.

# Mensajes HTTP



## ◆ Respuestas (*Response* )

◆ **Start Line:** Versión, código de respuesta y mensaje.

1XX	Información – El proceso continúa
2XX	Éxito - Acción recibida, comprendida y aceptada
3XX	Redirección – Se requiere nueva acción
4XX	Error en cliente – Sintaxis y/o Semántica inválida
5XX	Error en servidor – Falló pedido correcto

# Tipos de información

Si HTTP sólo transmite texto, ¿cómo puedo transferir imágenes, video, etc.?

- ◆ Utiliza MIME para describir contenido multimedia
- ◆ Cada objeto es etiquetado por el web server
  - ◆ text/html
  - ◆ text/plain
  - ◆ video/mp4
  - ◆ image/jpeg
  - ◆ application/pdf
  - ◆ ...

# Ejemplo: HTTP - POST



cliente web

Body

```
POST /itbaV/mynav.asp HTTP/1.1
Host: iol.itba.edu.ar
User-Agent: Mozilla/5.0
Accept: text/html,application/xhtml+xml
Referer: http://iol.itba.edu.ar/itbaV/welcome.asp
Content-Type: application/x-www-form-urlencoded
Content-Length: 76

secretNr=378924198&txtdni=12345678&txtpwd=mipass&Submit=Conectar&cmd=login
```

```
HTTP/1.1 200 OK
Connection: close
Date: Mon, 15 Feb 2019 17:05:22 GMT
Server: Microsoft-IIS/6.0
Content-Length: 11276
Content-Type: text/html
```

```
<html><head>
<title>ITBA OnLine</title>
.....
```



# Get vs Post

- ◆ GET requests
  - ◆ pueden ser "cacheados"
  - ◆ el browser los mantiene en el historial
  - ◆ pueden ser "bookmarked"
  - ◆ tienen longitud acotada
  - ◆ sólo para pedir datos
- ◆ POST requests
  - ◆ nunca son "cacheados"
  - ◆ no se mantienen en el historial del browser
  - ◆ no pueden ser "bookmarked"
  - ◆ no tienen restricción de longitud de datos

# Ejemplo: HTTP - OPTIONS



cliente web

```
OPTIONS * HTTP/1.1  
User-Agent: Mozilla/4.0 (compatible;  
MSIE5.01; Windows NT)
```



servidor web

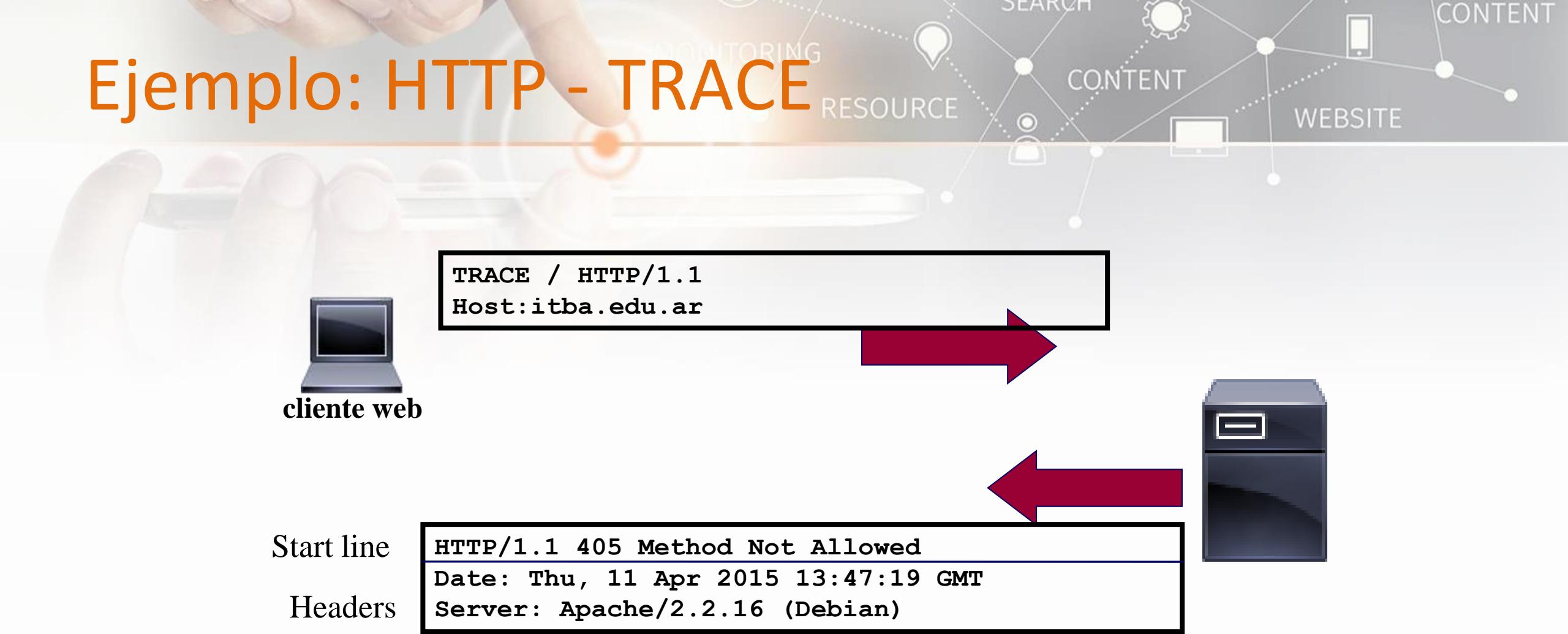


Start line

```
HTTP/1.1 200 OK  
Date: Mon, 01 Aug 2019 12:28:53 GMT  
Server: Apache/2.2.14 (Win32)  
Allow: GET,HEAD,POST,OPTIONS,TRACE  
Content-Type: httpd/unix-directory
```

Headers

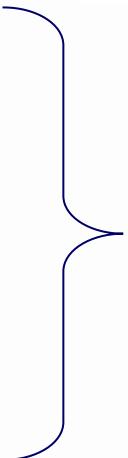
# Ejemplo: HTTP - TRACE



<https://tools.ietf.org/html/rfc7231#section-4.3.8>

# Headers HTTP

- ◆ Estructura: <nombre>: <valor asociado>
- ◆ Finalizan con una línea en blanco.
- ◆ Tipos:
  - ◆ Generales
  - ◆ De solicitudes
  - ◆ De respuestas
  - ◆ De contenido



Usualmente  
se envían en  
este orden

# Headers generales

- ◆ Cache-control: Directivas para cache
- ◆ Connection: Se definen opciones de conexión
- ◆ Date: Fecha de creación del mensaje
- ◆ Transfer-Encoding: Indica el encoding de transferencia
- ◆ Via: Muestra la lista de intermediarios por los que pasó el mensaje.

# Headers de solicitud

- ◆ Accept : Tipo de contenido aceptado por el cliente
- ◆ Accept-Charset: Charset (ISO-xxxx, UTF-8) aceptado por el cliente
- ◆ Accept-Encoding: Encoding (gzip, compress) aceptado por el cliente
- ◆ Expect: Comportamiento esperado del server frente al request.
- ◆ From: Email del usuario de la aplicación que generó el request.
- ◆ Host: Servidor y puerto destino del request.
- ◆ If-Modified-Since: Condiciona al request a la fecha indicada.
- ◆ Referer: URL del documento que generó el request
- ◆ User-Agent: Aplicación que generó el request
- ◆ Upgrade: solicita que use otro protocolo
- ◆ Range: solicita un rango (en bytes) del recurso

# Headers de respuesta

- ◆ Age: Estimación en segundos del tiempo que fue generada la respuesta en el server.
- ◆ Connection: close, keep-alive
- ◆ Location: URI a redireccionar
- ◆ Retry-After: Tiempo de delay para reintento
- ◆ Server: Descripción del software del server.
- ◆ Authorization: indica que el recurso necesita autorización

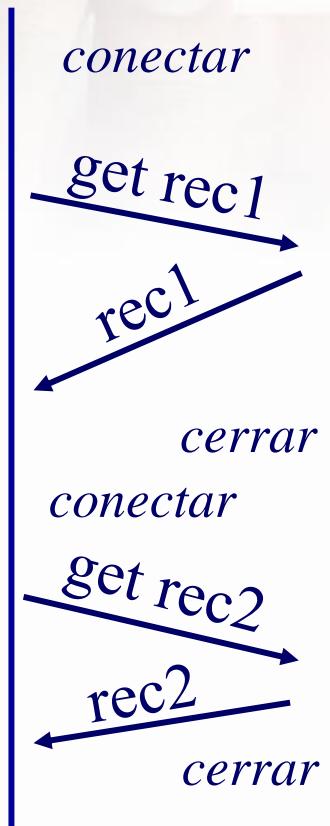
# Headers de contenido

- ◆ Allow: Métodos aplicables al recurso.
- ◆ Content-Encoding
- ◆ Content-Length
- ◆ Content-Location
- ◆ Content-MD5
- ◆ Content-Type
- ◆ Expires: Fecha de expiración del recurso.
- ◆ Last-Modified: Fecha de modificación del recurso

# Conexiones HTTP

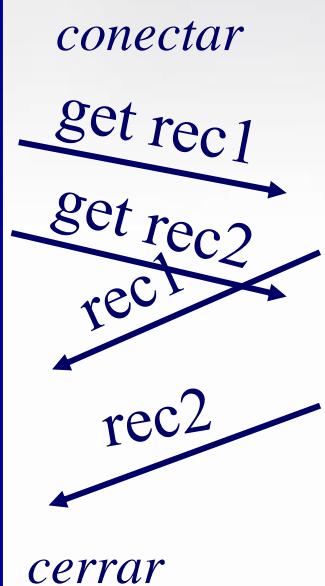


Origen      Destino



No persistente

Origen      Destino



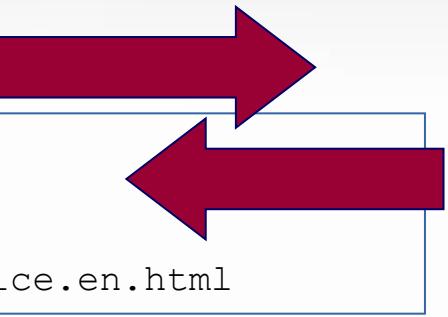
Persistente

# Negociación de contenido



**cliente web**

```
GET /people/alice HTTP/1.1  
Host: www.example.com  
Accept: text/html, application/xhtml+xml  
Accept-Language: en, de
```



```
HTTP/1.1 200 OK  
Content-Type: text/html  
Content-Language: en  
Content-Location: http://www.example.com/alice.en.html
```

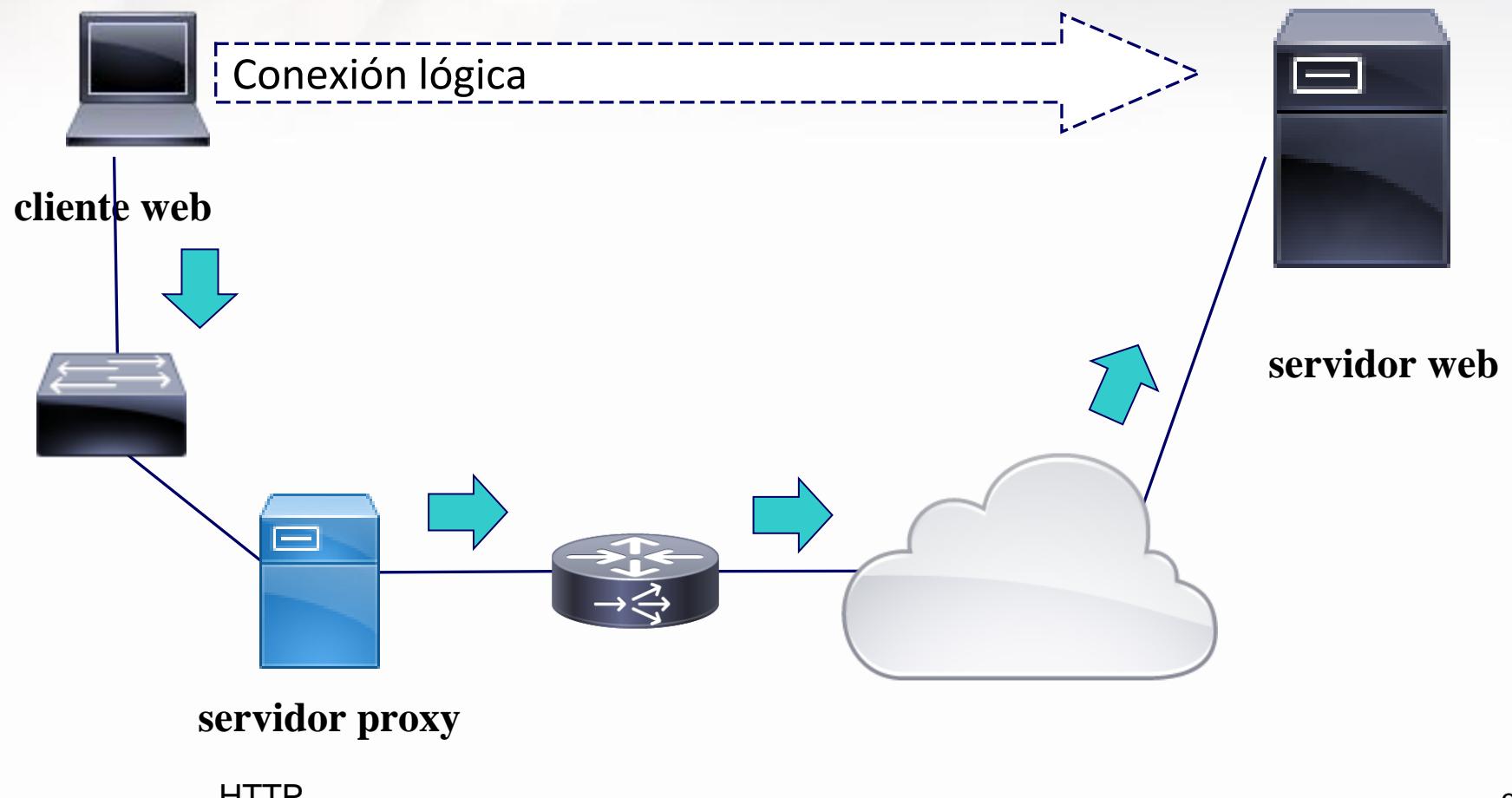
alternativa:



```
HTTP/1.1 302 Found  
Location: http://www.example.com/people/alice.en.html
```

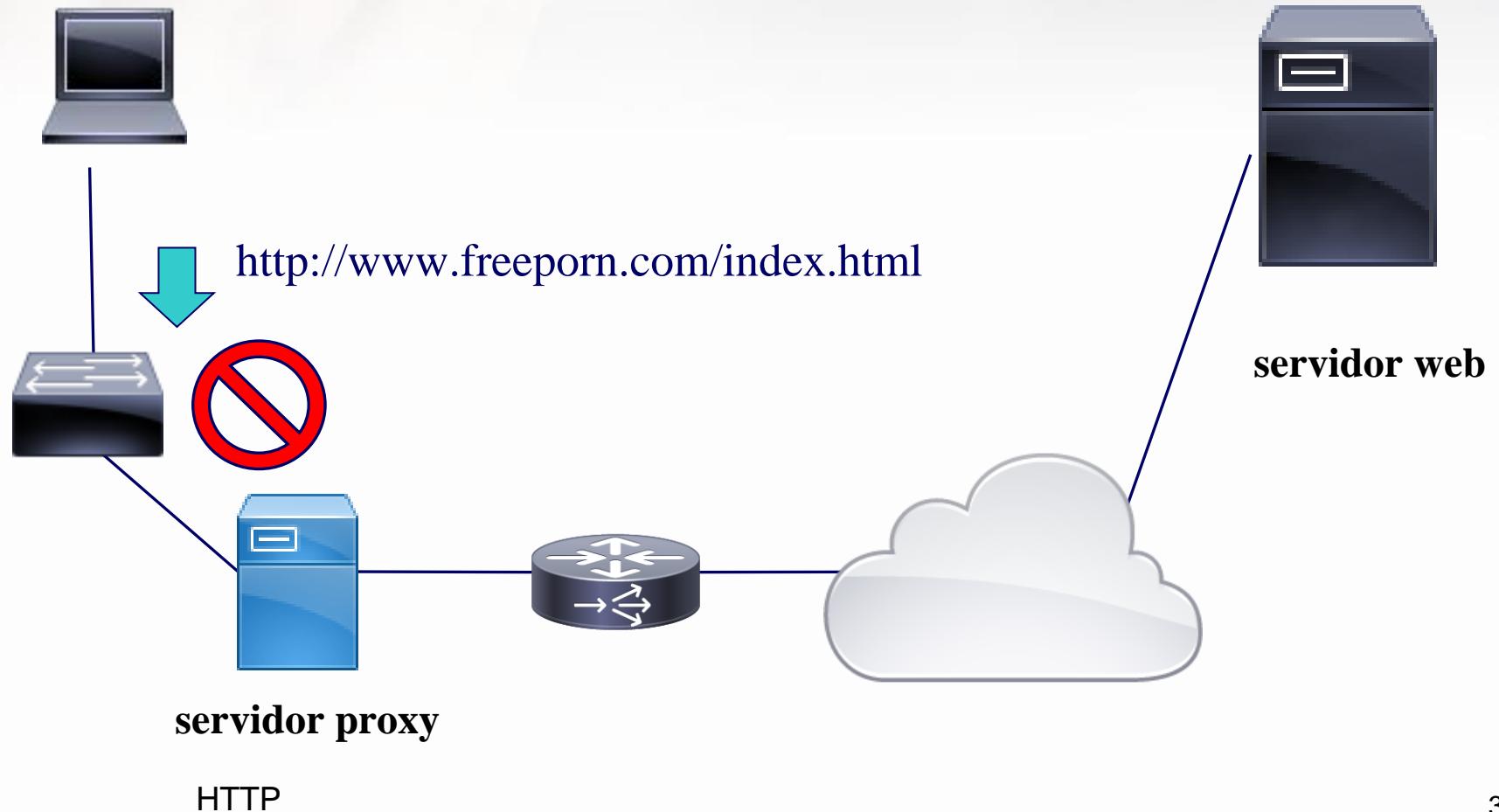
# Proxy servers

Un *proxy server* actúa como intermediario entre una aplicación cliente y un web server. Puede ser **explícito** o **transparente**.



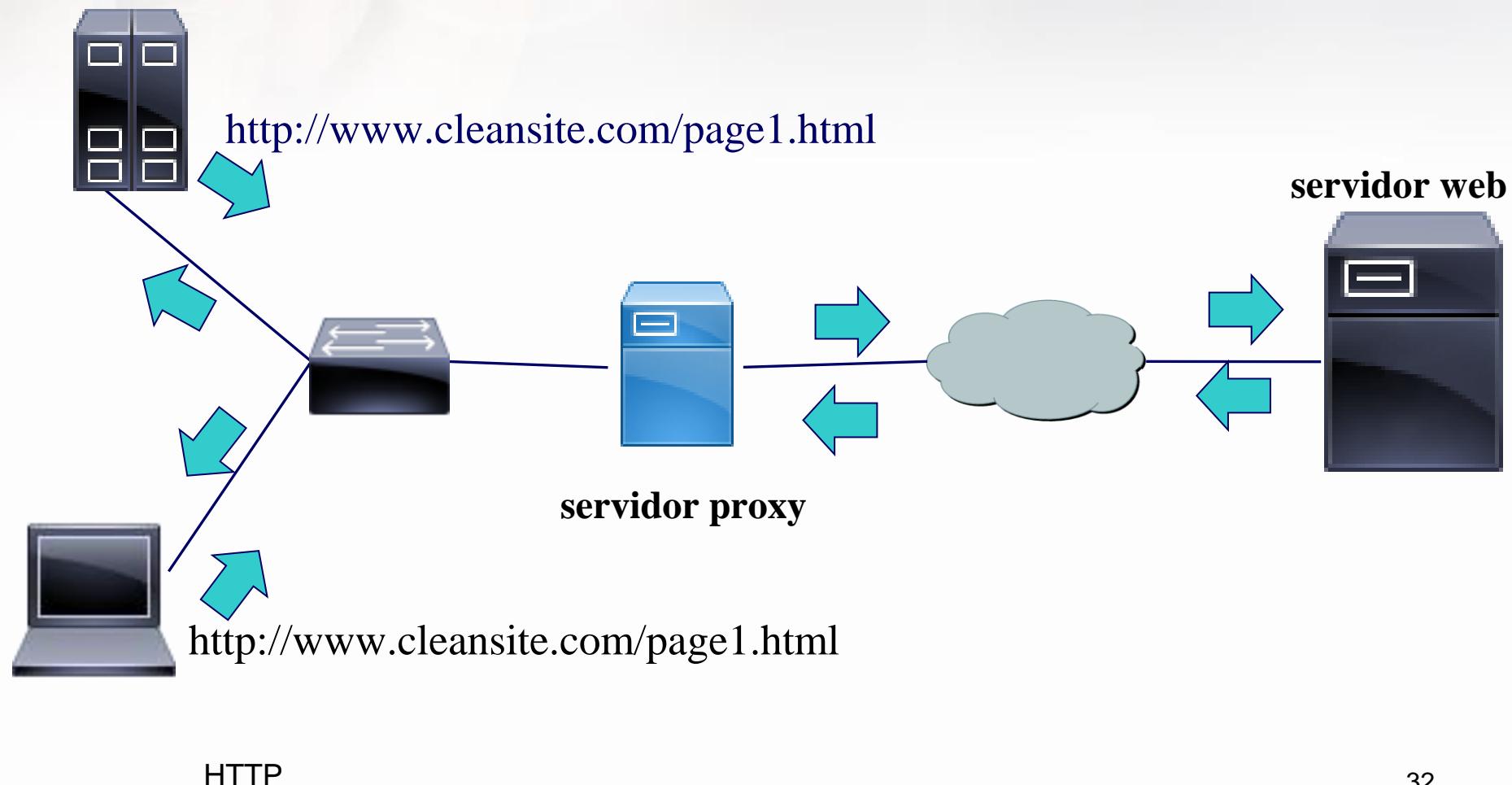
# Proxy servers

Un *proxy server* permite controlar el acceso a determinados sitios o páginas, y almacenar en caché recientes consultas.

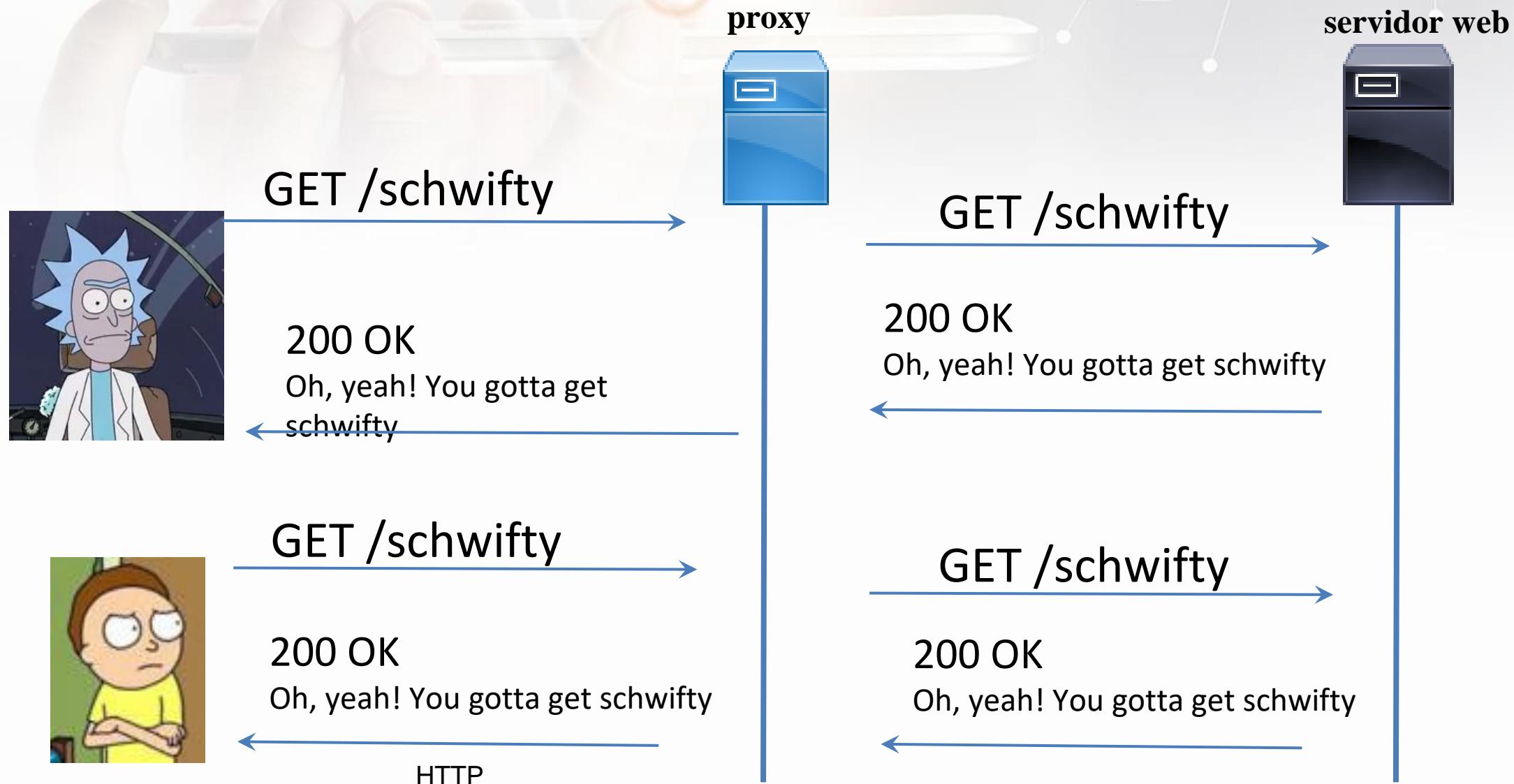


# Proxy servers

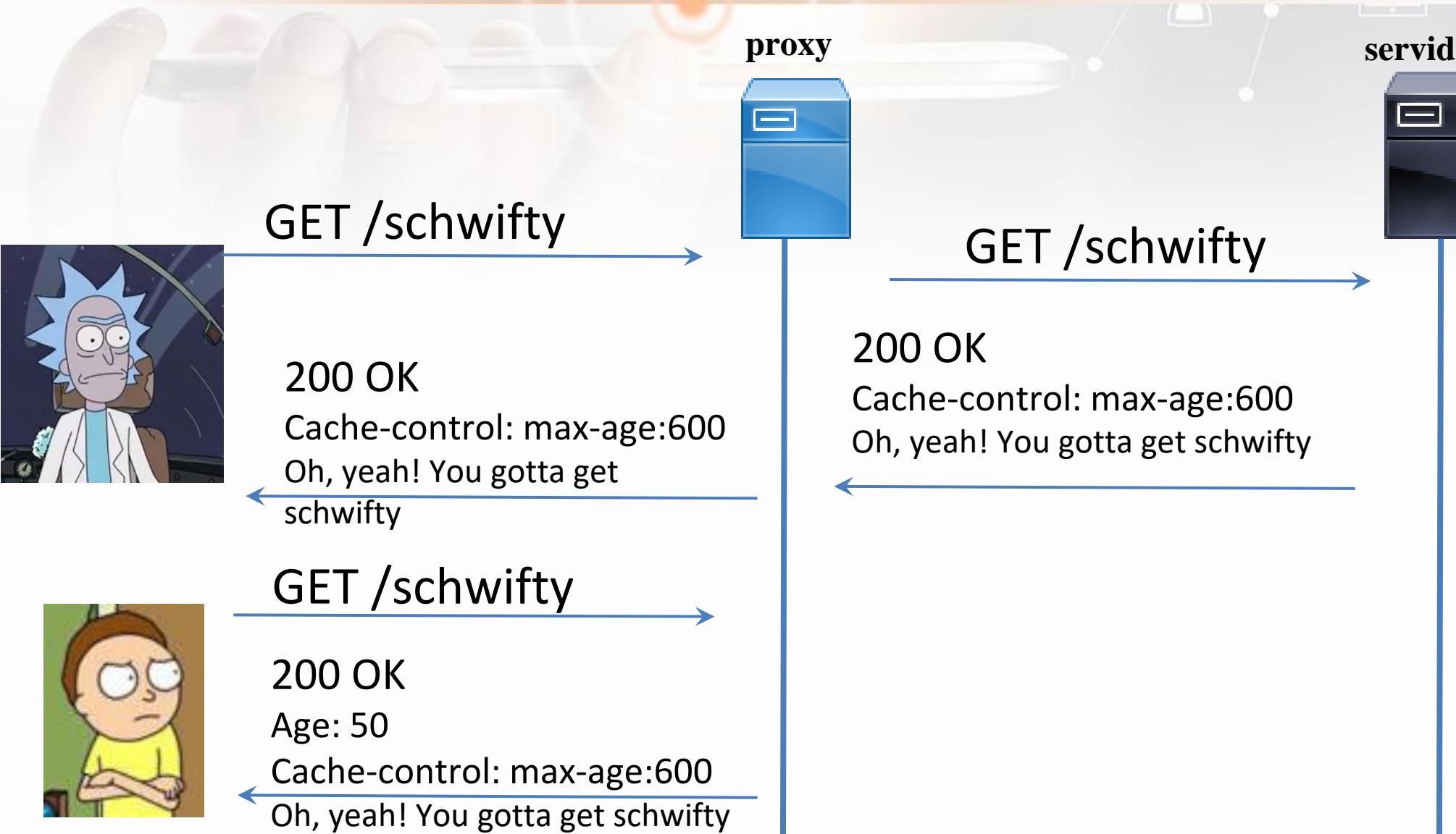
Un *proxy server* permite controlar el acceso a determinados sitios o páginas, y almacenar en caché recientes consultas.



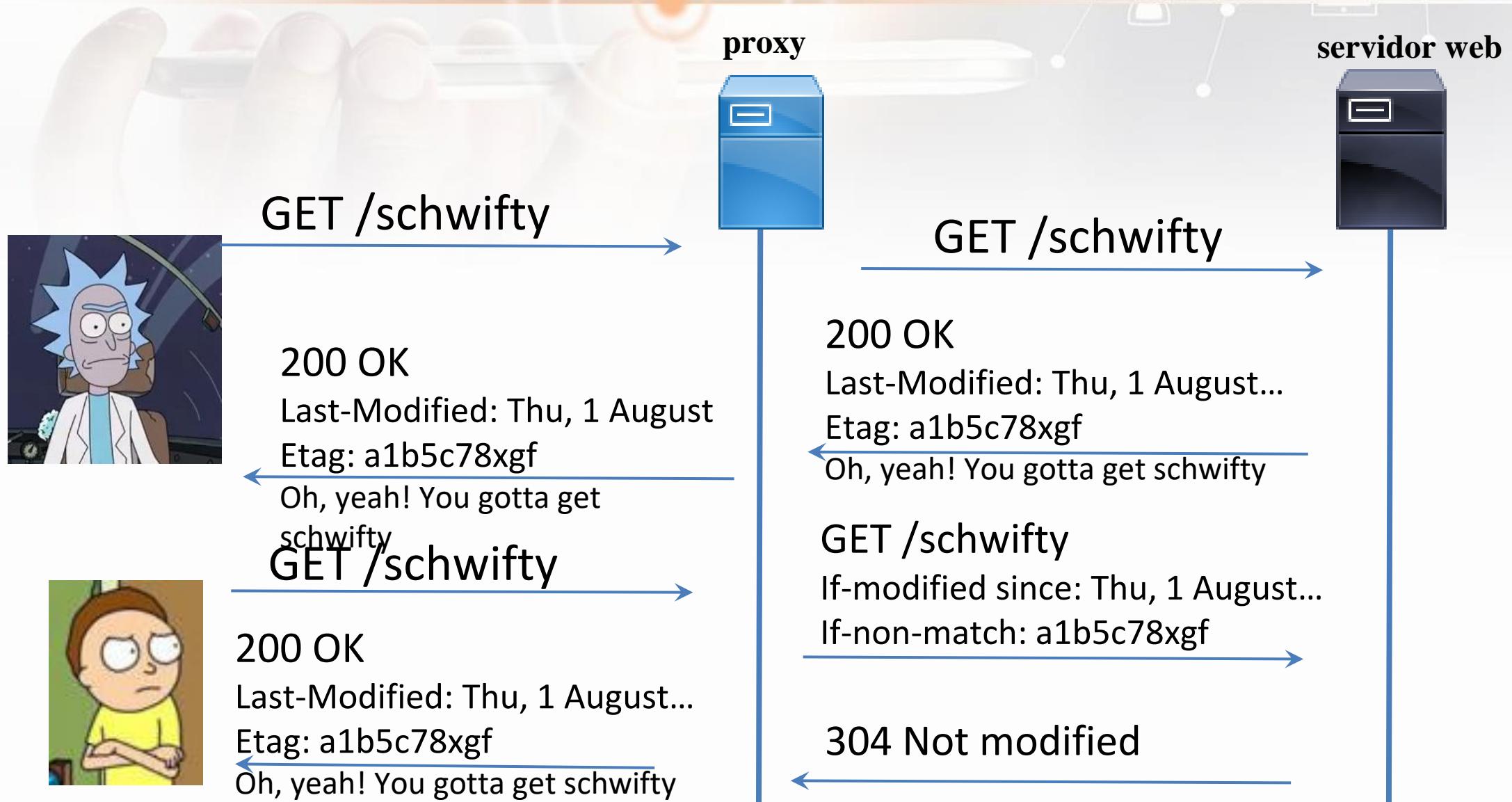
# Cache: sin directivas



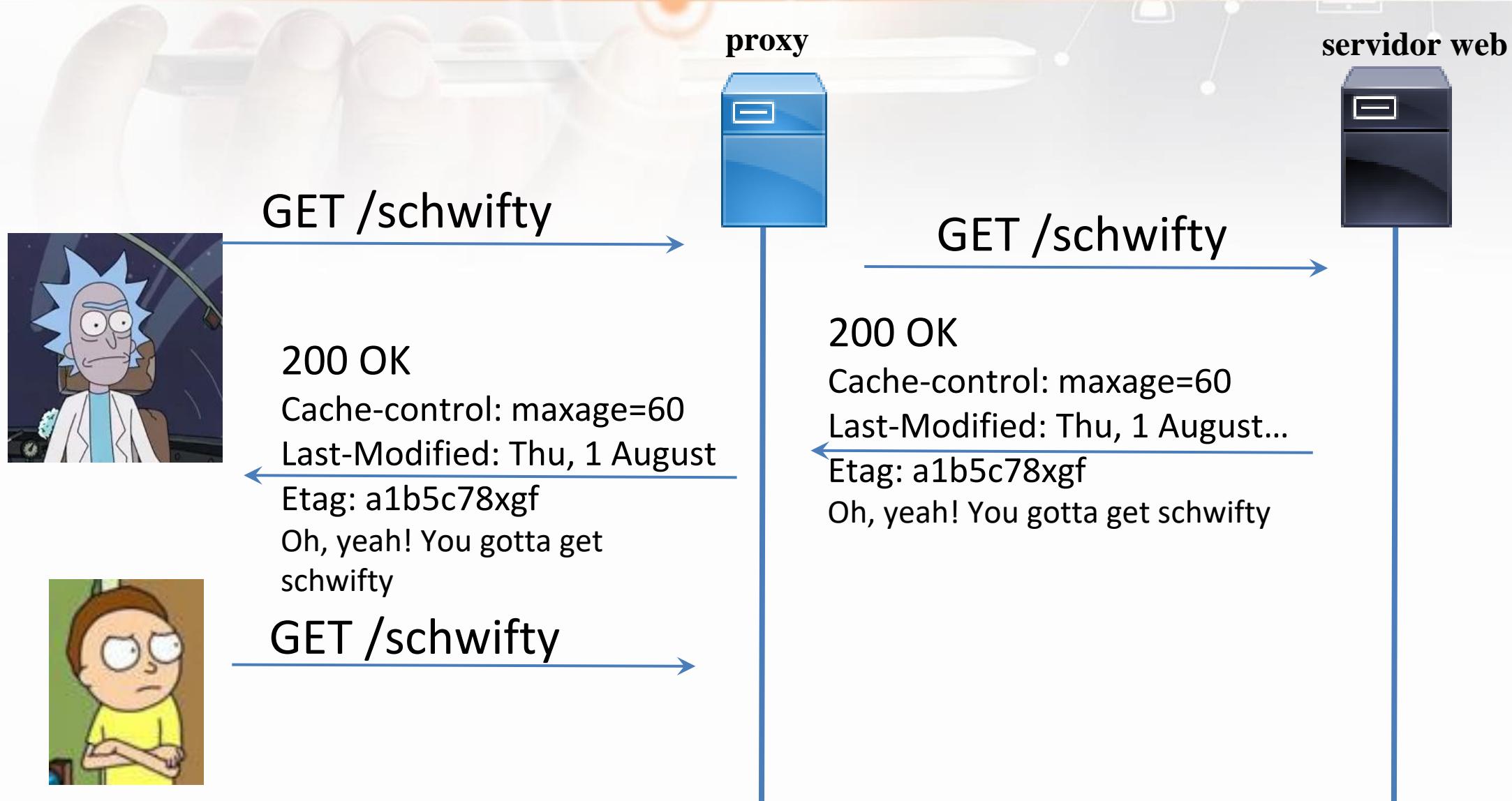
# Cache: max-age



# Cache: last-modified



# Cache: max-age y last-modified



# Cookies

- ◆ Pequeño texto de información enviada por el servidor y almacenada por el browser
- ◆ Usadas para mantener un estado entre el cliente y el servidor
  - ◆ Servidor envía cookie a cliente HTTP ( «*set-cookie*» *response header*)
  - ◆ Cliente HTTP retorna cookie al servidor ( «*cookie*» *request header*)
- ◆ Persistencia
  - ◆ Session cookie
  - ◆ Persistent cookie
- ◆ Third-party cookie
- ◆ Secure cookie

# Cookies



cliente web

```
POST /login.html HTTP/1.1  
username=jPerez  
pwd=123456
```



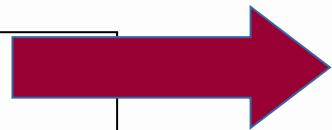
Valida usuario y clave.  
Crea un sessionId

```
HTTP/1.1 200 OK  
Content-Type: text/html  
Set-Cookie: sessionId=123xyz; Expires=Wed, 04 ...  
Set-Cookie: lastSession=2019/06/08  
...
```



```
GET /menu.html HTTP/1.1  
Host: www.example.com  
Cookie: sessionId=123xyz; lastSession=2019/08/08  
...
```

¿La sesión es válida?



```
HTTP/1.1 200 OK  
Content-Type: text/html
```



# cURL

- ◆ Herramienta de línea de comandos para transferir recursos en base a URLs
  - ◆ HTTP ( POST, PUT)
  - ◆ HTTPS
  - ◆ FTP ( rfc 959)
  - ◆ DICT (rfc 2229)
  - ◆ POP3 (rfc 1996)
  - ◆ SMTP (rfc 5321)
  - ◆ etc.

# cURL: ejemplos

- ◆ Incorrectos
  - ◆ curl https:asite.com
  - ◆ curl https:asite.com -head
- ◆ Correctos
  - ◆ curl http://google.com/humans.txt
  - ◆ curl https://www.google.com/humans.txt -i
  - ◆ curl -X POST -F 'locale=en' *url*

# wget

- ◆ Pensado para descargar archivos a disco

- ◆ wget <https://wordpress.org/latest.zip>

- ◆ wget -i files.txt

files.txt es un archivo de texto con una URL por línea

- ◆ wget --limit-rate=500k ...

# netcat

Como HTTP es un protocolo de texto, podemos "armar" los datos a enviar manualmente. Para ello necesitamos una aplicación que simplemente se conecte al servidor, nos pida los datos, los envíe y nos muestre la respuesta

netcat permite esto y mucho más

- ◆ `netcat www.google.com 80`
- ◆ `netcat -l -p 8080 -v`

# Material de lectura

- Capítulo 2.2 de la bibliografía
- Códigos de status HTTP  
[HTTP response status codes](#)
- GET vs POST: [HTTP Methods GET vs POST](#)



# RESOLUCIÓN DE NOMBRES

Nombres de dominio

DNS

Utilidades DNS

DNS spoofing

# Resolución de nombres

## Nombre de dominio (domain name)

Un **dominio** es una colección de computadoras que pueden ser accedidas usando un nombre en común.

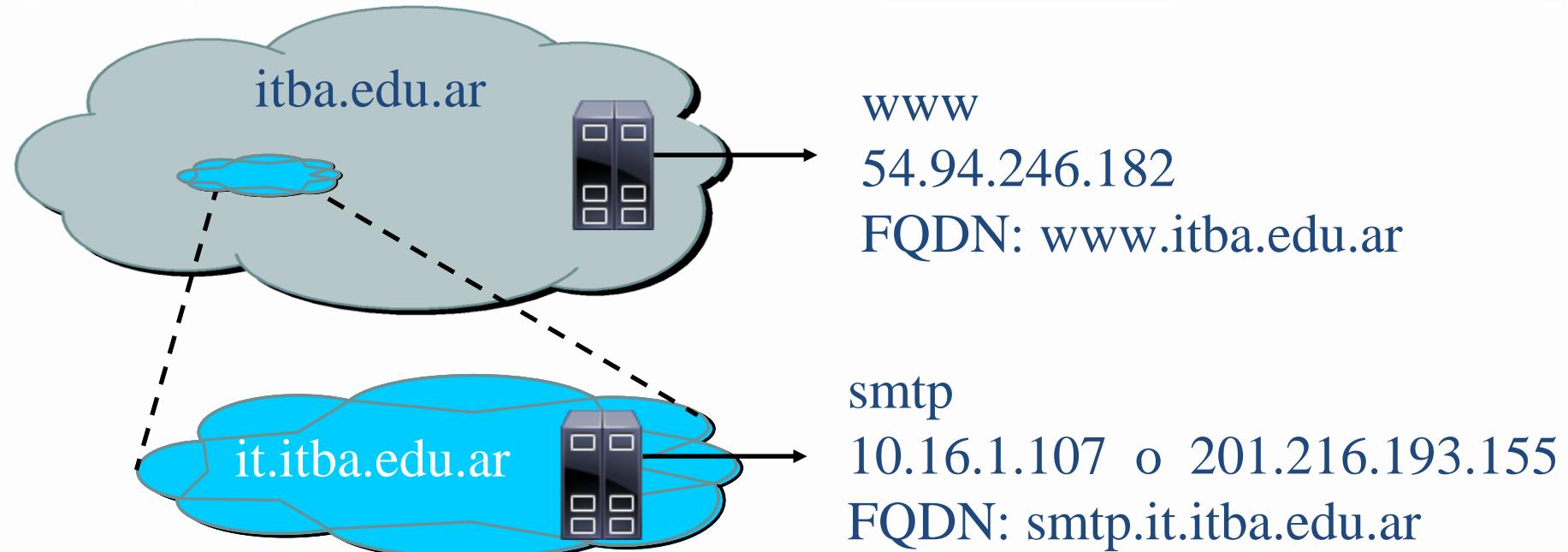
Un **nombre de dominio** hace referencia al nombre de múltiples hosts que son referenciados colectivamente ( `itba.edu.ar`, `ibm.com`, etc.)

Los dominios tienen distintos niveles, siendo **.com** el dominio *top-level* más conocido de ellos. Ver RFC 1591 y RFC 3071.

# Resolución de nombres

Dentro de un *top-level domain*, una organización tiene su propio dominio o dominios. A su vez dentro del dominio puede haber sub-dominios. Y cada host tiene su nombre propio (*hostname*).

Uniendo el nombre del host con el dominio al cual pertenece se forma el **FQDN** (*fully qualified domain name*).

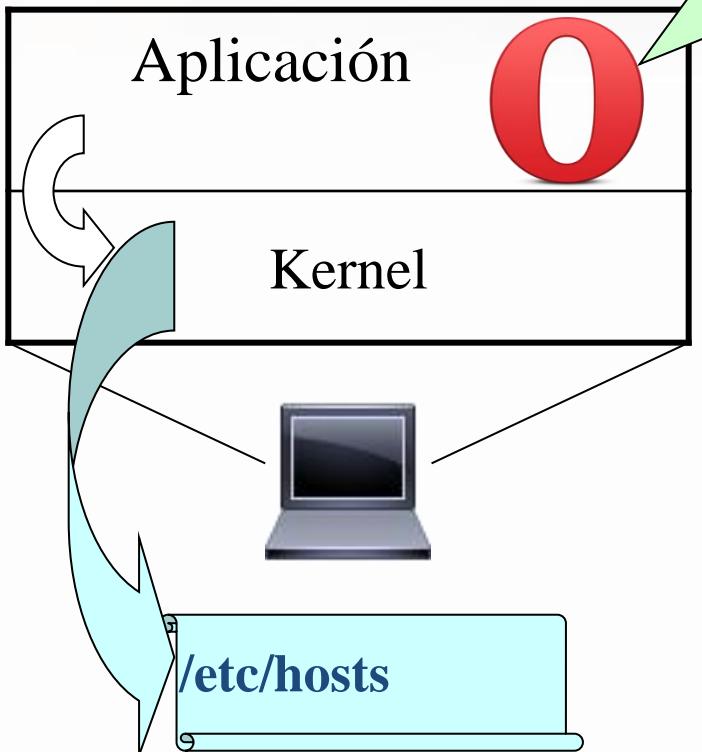


# Resolución de nombres

¿Cómo se obtiene, dado un FQDN, su número de IP?

Opción 1:

¿ IP de www.itba.edu.ar ?



# Archivo /etc/hosts

- ◆ Contiene una línea por cada número de IP y el o los nombres asociados a dicho IP.
- ◆ Por defecto se consulta primero este archivo y luego DNS
- ◆ Se puede cambiar el orden en /etc/nsswitch.conf

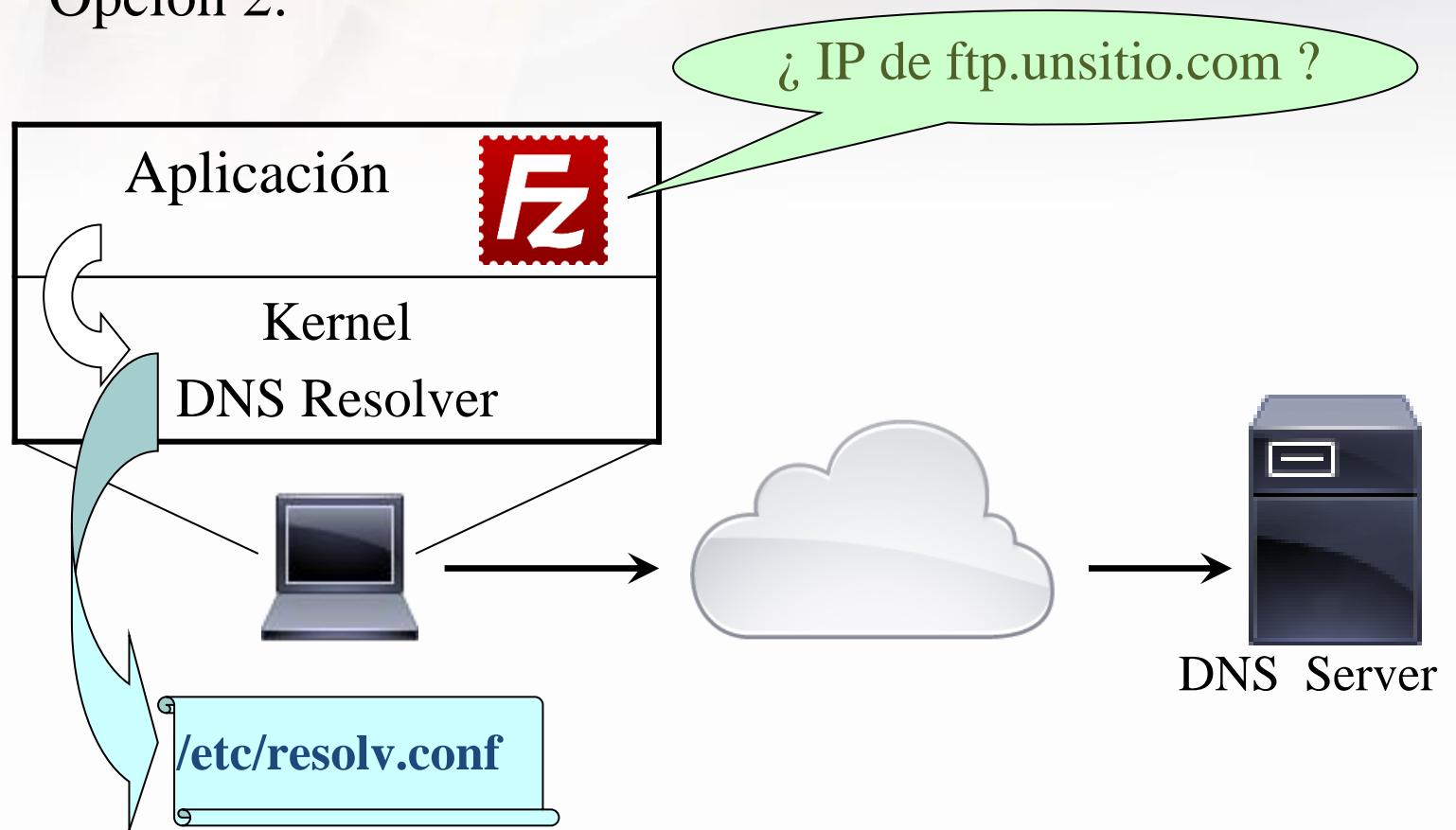
```
# Do not remove the following line, or various programs
# that require network functionality will fail.
127.0.0.1      localhost mihost mihost.com
::1            localhost
192.168.2.2    printserver
127.0.0.1      pagina12.com.ar
127.0.0.1      clarin.com

# End of hosts.
```

# Resolución de nombres

¿Cómo se obtiene, dado un FQDN, su número de IP?

Opción 2:



# Resolución de nombres

Las opciones de configuración de `/etc/resolv.conf` son:

## ***nameserver direcciones***

Números IP de los servidores de nombres que consultará el resolver (hasta 3). Si no hay ningún nameserver utilizará al propio host como servidor de nombres.

## ***domain nombre***

Dominio local por defecto. El resolver agrega este nombre a todo hostname que no contenga un punto antes de resolverlo.

## ***search dominios***

Similar al anterior pero pueden ser varios dominios (no se usan ambas opciones en simultáneo)

# Resolución de nombres

## **sortlist red[/mascara]**

En caso de recibir múltiples direcciones IP para un nombre, reordena las direcciones en base a las redes listadas en esta opción.

## **options opción**

Usada para seteos opcionales. Las posibles opciones son:

- debug
- ndots:*n*
- timeout:*n*
- attempts:*n*
- rotate
- no-check-names
- inet6

# Resolución de nombres

Ejemplo de `/etc/resolv.conf` del host 192.168.2.20

```
domain      itba.edu.ar
nameserver  192.168.2.1
nameserver  8.8.8.8
nameserver  200.49.159.69
```

Linux incluye las utilidades **dig** y **host** para resolver nombres, ya sea para obtener el IP en base a un nombre o el nombre en base a un IP (*reverse DNS*).

# Resolución de nombres

Por defecto se consulta primero el archivo `/etc/hosts` y luego –de ser necesario- al DNS resolver. Esto puede cambiarse editando el archivo `/etc/host.conf`, el que típicamente contiene una sola línea

```
order hosts, bind
```

En caso de querer que se consulte siempre al DNS resolver sólo hay que cambiar el orden, como el ejemplo siguiente:

```
order bind, hosts
```

Para más opciones consultar *man host.conf*

**Un host mantiene en su cache los nombres que ha podido resolver**

# DNS

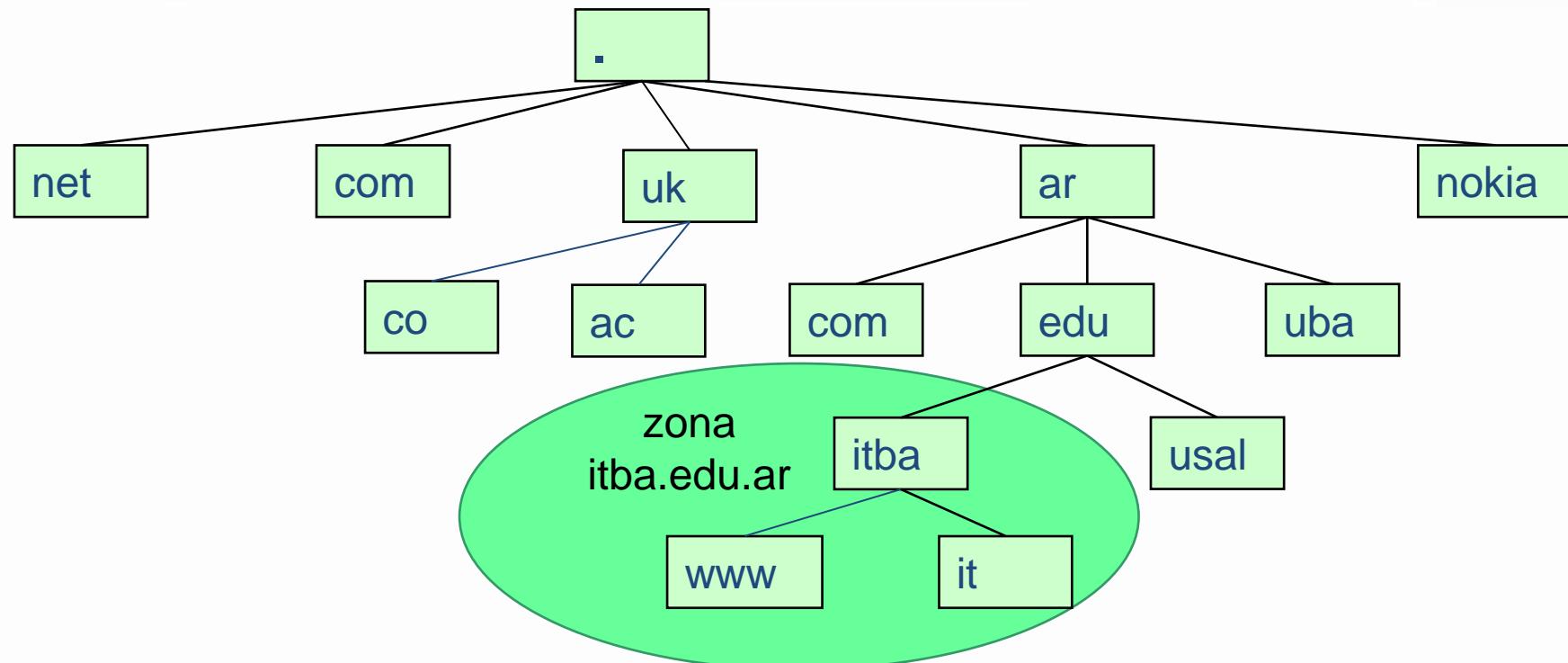


Algunos nombres de dominio a resolver se pueden encontrar en /etc/hosts, pero en la mayoría de los casos deberán ser resueltos preguntando a un **servidor DNS**.

- Base de datos distribuida implementada en una jerarquía de servidores de nombre
- Protocolo de aplicación que permite consultar dicha base

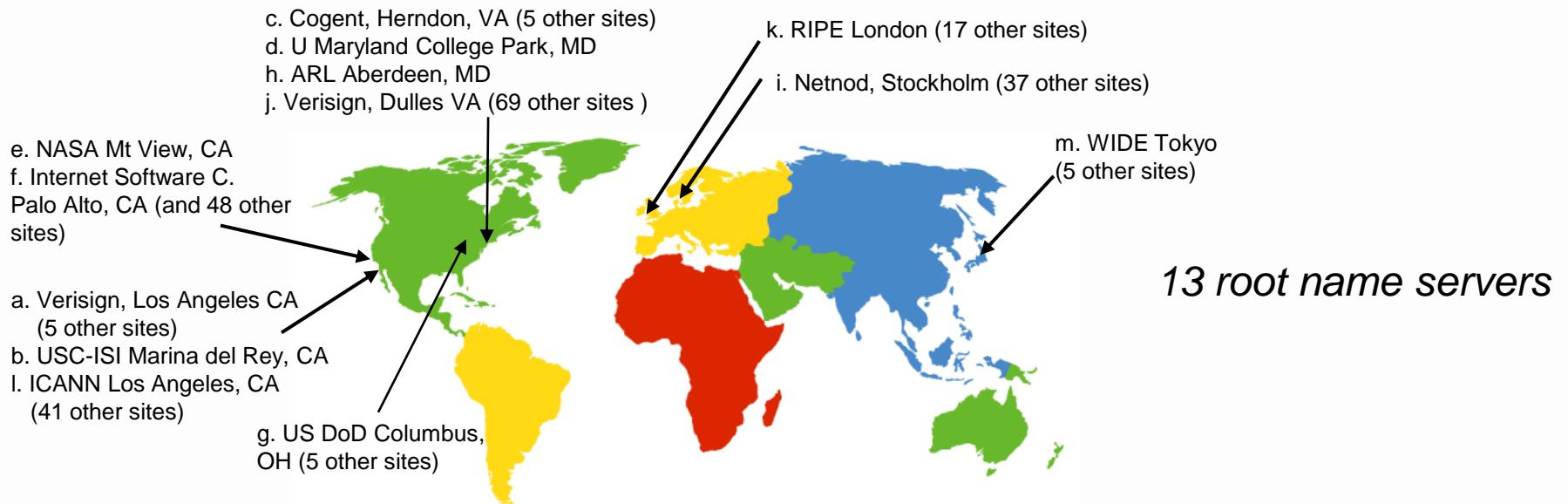
# DNS

DNS es un sistema jerárquico distribuido para traducir nombres de hosts a direcciones IP. La información en DNS puede ser vista como un árbol invertido donde la raíz hace referencia a servidores de nombres raíz (*root name servers*).



# DNS: root name servers

- usados cuando el servidor de nombres local no puede resolver un nombre
- root name server:
  - contacta name server autorizado si no conoce la respuesta
  - almacena
  - retorna respuesta al servidor de nombres local



# TLD, servidores autorizados

## *top-level domain (TLD) servers:*

- responsables de com, org, net, edu, aero, jobs, museums, y todos los países (ar, uk, uy, tv, etc.)
- Verisign para .com TLD, Educause para .edu TLD

## *authoritative DNS servers:*

- Cada organización mantiene su propio servidor(es) DNS
- Puede ser mantenido por la misma organización o un proveedor

Para un listado completo de TLD y sus responsables ver <https://www.iana.org/domains/root/db>

# Servidor DNS local

- No necesariamente pertenecen a la jerarquía
- Cada ISP tiene uno, conocido como “default name server”
- Cuando un host realiza una consulta DNS, es enviada al servidor DNS local
  - Busca en la caché si tiene el par nombre-dirección y no caducó
  - Actúa como un proxy

# Resolución de nombres

## Ejemplos

```
user@server:~$ host www.dc.uba.ar
```

```
www.dc.uba.ar is an alias for www-1.dc.uba.ar.
```

```
www-1.dc.uba.ar is an alias for dc.uba.ar
```

```
dc.uba.ar has address 157.92.27.127
```

```
user@server:~$ host 157.92.27.21
```

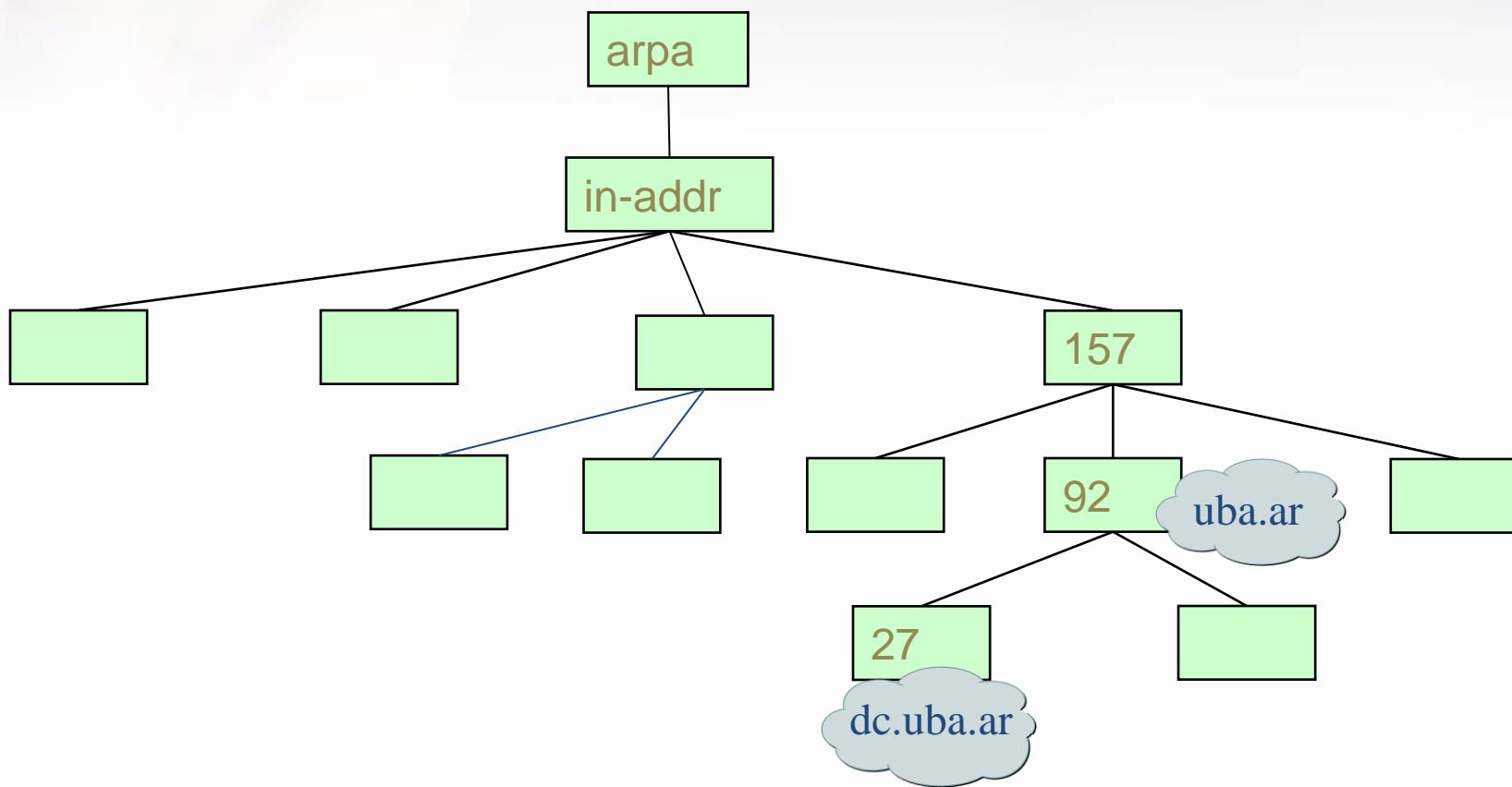
```
21.27.92.157.in-addr.arpa domain name pointer
```

```
dc.uba.ar.
```

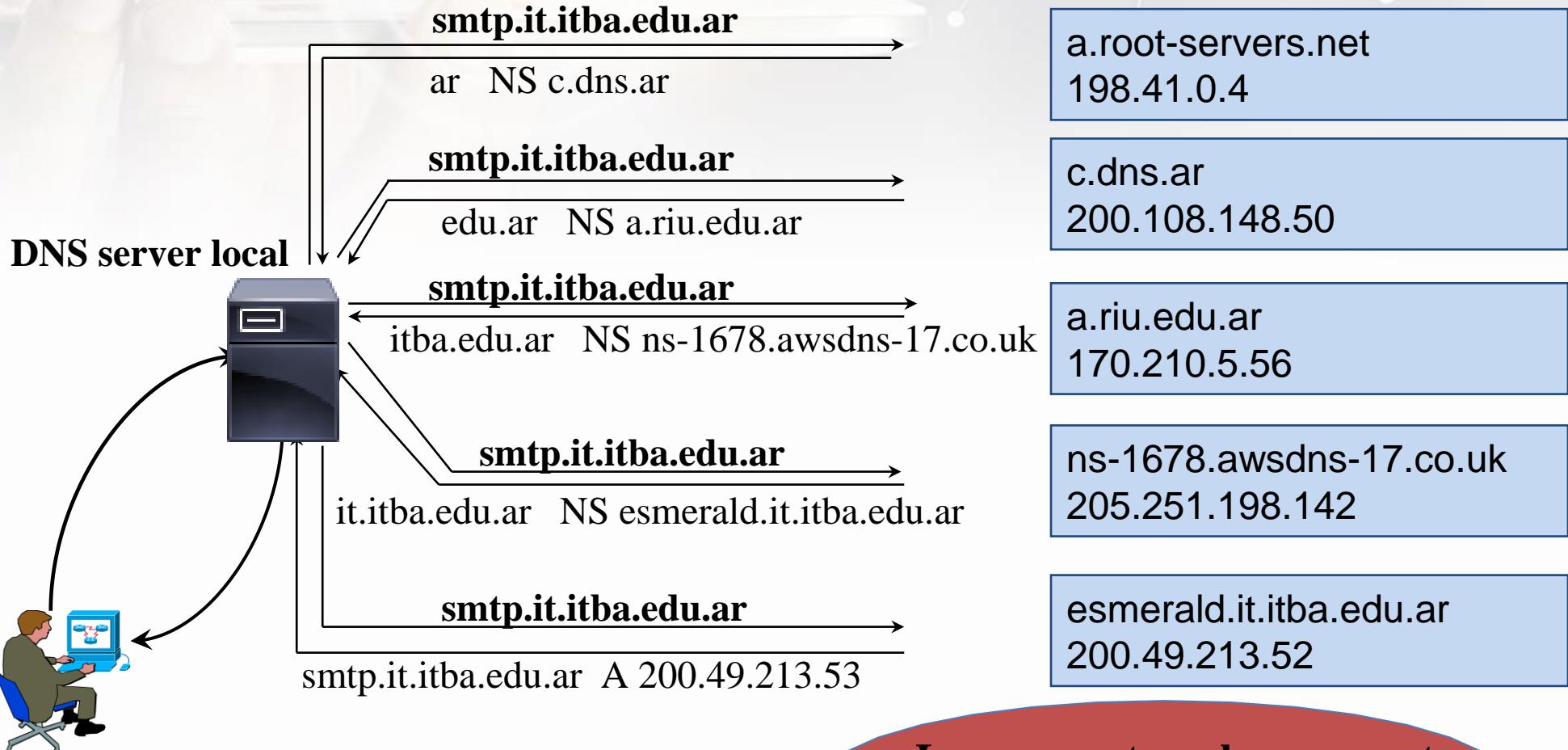
# DNS



Para poder responder un *DNS reverse* se usa un árbol especial en base a los números IP.



# DNS: consultas



¿Cuál es el IP de  
`smtp.it.itba.edu.ar`?

Resolución de nombres

Las preguntas y las respuestas  
utilizan UDP

# DNS: cache

- ◆ Una vez que el servidor de nombres aprende el mapeo, lo almacena en cache
- ◆ Expiran tras un tiempo (TTL)
- ◆ Servidores de nombre generalmente cachean la dirección de los servidores TLD
- ◆ Los Root Servers no suelen visitarse

¿Qué sucede si cambio el IP de un host conocido o migro una aplicación?

# DNS



Los servidores raíz DNS son operados por IETF (*Internet Engineering Task Force*). Cuando alguien obtiene un nombre de dominio es el responsable de mantener, en un servidor DNS **master-primary**, actualizado ese dominio.

En Unix y Linux DNS es implementado generalmente con BIND (*Berkeley Internet Name Domain*).

# Configuración de un servidor DNS

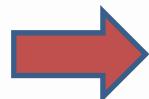
Existen cuatro niveles de configuración:

- Resolver only systems
- Caching-only servers
- ~~Master~~ Primary servers
- ~~Slave~~ Secondary servers

# DNS Caching-only Server

- Aprende las respuestas acerca de los nombres de dominios de otro servidor. Una vez que aprende esa respuesta, la almacena para futuras preguntas.
- Todos los servidores utilizan esta técnica pero un caching-only server es la única técnica que utiliza.
- No es considerado un server autorizado porque su información es de segunda mano.

El programa **dnscache** funciona solo como cache de servidores de nombre, y puede ser usado en caching-only servers.



**DNS Caching-only Server**

Resolución de nombres

# DNS Primary Server



- ◆ Es la fuente autorizada para toda la información de una zona específica.
- ◆ Lee la información sobre el dominio desde un archivo construido por el administrador.
- ◆ Es un server autorizado para un dominio o parte de un dominio pues puede responder con autoridad preguntas sobre ese dominio.

# DNS Secondary Server

- ◆ Transfiere información completa de una zona desde un master server y la almacena en un archivo en el disco local.
- ◆ Mantiene información completa y actualizada de una zona y puede responder preguntas sobre zona en forma autorizada.

# Registros DNS

DNS: base de datos distribuída de "resource records" (RR)

formato RR : (name, value, type, ttl)

type	name	value
A	nombre del host	IP
NS	dominio	nombre del host autorizado a resolver nombres del dominio
CNAME	alias para el nombre canónico	nombre canónico
MX	dominio	nombre del servidor de mails para el dominio

# Registros DNS



DNS: base de datos distribuída de "resource records" (RR)

formato RR : (`name, value, type, ttl`)

<b>type</b>	<b>name</b>	<b>value</b>
AAAA	nombre del host	IPv6
SOA	Start of Authority	Inicio de información autorizada
SRV	Localizador de servicios	Para no crear nuevos registros tipo MX
RP	Persona responsable	Normalmente el mail del responsable

# Registros DNS



DNS: base de datos distribuída de "resource records" (RR)

formato RR : (name, value, type, ttl)

type	value
TXT	un texto libre
SPF	Lista de IPs de hosts autorizados para enviar mails en nombre de este dominio

# DNS: ejemplo parcial de configuración

```
$ORIGIN example.com.  
@ IN SOA ns1.example.com. hostmaster.example.com. (  
                      zone-admin.example.com.          ; address of responsible party  
                      2016072701                ; serial number  
                      3600                  ; refresh period  
                      600                   ; retry period  
                      1w                    ; expire time  
                      3h )                 ; minimum ttl  
                      IN NS ns1.example.com.      ; DNS in the domain  
                      IN NS ns.outsider.com.    ; external to domain  
                      IN NS ns2.example.com.  
                      IN MX 10 mail.example.com.  
                      IN MX 20 smtp.example.com  
ns1   IN A  192.168.0.1  
ns2   IN A  201.13.248.106  
mail  IN A  204.13.248.106  
smtp  CNAME mail.example.com.  
www   IN A  192.168.0.3
```

Ver apunte "Understanding DNS—anatomy of a BIND zone file" en Material Didáctico / Prácticas / 03.-DNS

# reverse-zone



```
$ORIGIN 1.0.10.in-addr.arpa.  
$TTL 86400  
@ IN SOA dns1.example.com. hostmaster.example.com. (  
    2001062501 ; serial  
    21600      ; refresh after 6 hours  
    3600       ; retry after 1 hour  
    604800     ; expire after 1 week  
    86400 )    ; minimum TTL of 1 day  
  
IN NS dns1.example.com.  
IN NS dns2.example.com.  
  
20 IN PTR alice.example.com.  
21 IN PTR betty.example.com.  
22 IN PTR charlie.example.com.  
23 IN PTR doug.example.com.  
24 IN PTR ernest.example.com.  
25 IN PTR fanny.example.com.
```

# Utilidades DNS

Para poder testear si la configuración es correcta se suelen usar 3 utilidades de línea de comandos:

- ◆ host
- ◆ nslookup
- ◆ dig

Para obtener información sobre el responsable de un dominio se puede utilizar whois.

# whois: ejemplo

ITBA.edu.ar - Instituto Tecnológico de Buenos Aires

Dirección Postal: Av. E. Madero 399 (1106) - Capital Federal

Teléfonos: +54-11-4314-7778

Incumbencia Principal: Instituto Universitario

## DATOS DEL CONTACTO TECNICO:

Nombre: Lo Nigro Miguel Martín

Dirección Postal: Av. Eduardo Madero 399

Teléfonos: +54 11 21504800 interno 5926 Fax:

Dirección de e-mail: [mlonigro@itba.edu.ar](mailto:mlonigro@itba.edu.ar)

Horario de disponibilidad para contacto telefónico: 13 - 17 Hs

...

# Utilidades DNS: host

**host** informa, dado un nombre de host, el número IP que le corresponde.

```
user@server:~$ host clarin.com
```

```
www.clarin.com has address 200.42.136.212  
clarin.com mail is handled by 0 clarin-  
com.mail.protection.outlook.com.
```

```
user@server:~$ host upa
```

```
upa.midominio.com has address 172.16.12.4
```

```
user@server:~$ host -n 127.0.0.1
```

```
1.0.0.127.in-addr.arpa domain name pointer localhost.
```

```
user@server:~$ host localhost
```

```
localhost has address 127.0.0.1  
localhost has IPv6 address ::1
```

# Utilidades DNS: dig

**dig** permite ver los registros para un dominio

```
user@server:~$ dig www.itba.edu.ar
; <>> DiG 9.12.0 <>> www.itba.edu.ar
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 499
;;
;; QUESTION SECTION:
;www.itba.edu.ar.           IN      A

;;
;; ANSWER SECTION:
www.itba.edu.ar.        172800  IN      CNAME    AWS-S-BAL04.itba.edu.ar.
AWS-S-BAL04.itba.edu.ar. 172800  IN      CNAME    AWS-S-BAL04-248295998.sa-
east-1.elb.amazonaws.com.
AWS-S-BAL04-248295998.sa-east-1.elb.amazonaws.com. 60 IN A 54.94.221.168
AWS-S-BAL04-248295998.sa-east-1.elb.amazonaws.com. 60 IN A 54.94.246.182
```

# Utilidades DNS: dig

Ejemplo: **dig www.itba.edu.ar (cont.)**

```
;; AUTHORITY SECTION:  
sa-east-1.elb.amazonaws.com. 67 IN NS ns-632.awsdns-15.net.  
sa-east-1.elb.amazonaws.com. 67 IN NS ns-2034.awsdns-62.co.uk.  
sa-east-1.elb.amazonaws.com. 67 IN NS ns-1276.awsdns-31.org.  
sa-east-1.elb.amazonaws.com. 67 IN NS ns-291.awsdns-36.com.  
  
;; ADDITIONAL SECTION:  
ns-291.awsdns-36.com. 12086 IN A 205.251.193.35  
ns-291.awsdns-36.com. 9492 IN AAAA 2600:9000:5301:2300::1  
ns-632.awsdns-15.net. 34682 IN A 205.251.194.120  
ns-632.awsdns-15.net. 84172 IN AAAA 2600:9000:5302:7800::1  
ns-1276.awsdns-31.org. 8938 IN A 205.251.196.252  
ns-1276.awsdns-31.org. 8938 IN AAAA 2600:9000:5304:fc00::1  
ns-2034.awsdns-62.co.uk. 9492 IN AAAA 2600:9000:5307:f200::1
```

# Utilidades DNS: dig

Ejemplo: **dig www.itba.edu.ar (cont.)**

```
; ; Query time: 2 msec
; ; SERVER: 10.1.0.10#53(10.1.0.10)
; ; WHEN: Tue Aug 22 09:31:59 -03 2017
; ; MSG SIZE  rcvd: 462
```

```
~$ dig www.itba.edu.ar +nocomments +noquestion +noauthority +noadditional +nostats
; <>> DiG 9.14.10 <>> www.itba.edu.ar +nocomments +noquestion +noauthority +noadditional
+nostats
;; global options: +cmd
www.itba.edu.ar.      59    IN    CNAME  aws-s-bal04.itba.edu.ar.
aws-s-bal04.itba.edu.ar. 39    IN    CNAME  aws-s-bal04-248295998.sa-east-1.
elb.amazonaws.com.
aws-s-bal04-248295998.sa-east-1.elb.amazonaws.com. 59 IN A 54.233.177.225
aws-s-bal04-248295998.sa-east-1.elb.amazonaws.com. 59 IN A 52.67.10.227
```

# Utilidades DNS: dig

- `dig host +noall +answer`
- `dig host MX +noall +answer`
- `dig -t NS host +noall +answer`
- `dig host ANY +noall +answer`
- `dig host ns +short`
- `dig -x 13.227.69.6`
- `dig host +trace`

# Utilidades DNS: nslookup



**nslookup** puede ser utilizado en forma interactiva o en forma similar a dig.

```
user@server:~$ nslookup www.itba.edu.ar
Server:      10.16.1.102
Address:     10.16.1.102#53
Non-authoritative answer:
www.itba.edu.ar canonical name = aws-s-bal04.itba.edu.ar
aws-s-bal04.itba.edu.ar canonical name = aws-s-bal04-248295998.sa-east-
1.elb.amazonaws.com.
Name:        aws-s-bal04-248295998.sa-east-1.elb.amazonaws.com
Address:    54.233.98.53
Name:        aws-s-bal04-248295998.sa-east-1.elb.amazonaws.com
Address:    54.94.246.182
```

# Utilidades DNS: nslookup

Si se lo ejecuta sin parámetros ingresa al modo interactivo

```
user@server:~$ nslookup  
> clarin.com  
Server:      192.168.2.1  
Address:     192.168.2.1  
  
Non-authoritative answer:  
Name:  clarin.com  
Address: 200.42.136.212  
  
>dns.itba.edu.ar  
  
*** Can't find dns.itba.edu.ar: No answer
```

Intento adivinar el servidor DNS

# Utilidades DNS: nslookup

```
> set type=NS  
> itba.edu.ar
```

Server: 192.168.2.1  
Address: 192.168.2.1

Ahora la consulta será  
por registros NS

**Non-authoritative answer:**

```
itba.edu.ar      nameserver = ns-888.awsdns-47.net.  
itba.edu.ar      nameserver = ns-1287.awsdns-32.org.  
itba.edu.ar      nameserver = ns-90.awsdns-11.com
```

```
> server ns-888.awsdns-47.net
```

Default server: ns-888.awsdns-47.net  
Address: 205.251.198.142#53  
Default server: ns-888.awsdns-47.net  
Address: 2600:9000:5306:8e00::1#53

Le pregunto al server  
autorizado

# Utilidades DNS: nslookup

```
> itba.edu.ar
```

```
Server:      ns-1678.awsdns-17.co.uk  
Address:     205.251.198.142#53
```

```
itba.edu.ar      nameserver = ns-1678.awsdns-17.co.uk.  
itba.edu.ar      nameserver = ns-888.awsdns-47.net.  
itba.edu.ar      nameserver = ns-1287.awsdns-32.org.  
itba.edu.ar      nameserver = ns-90.awsdns-11.com.
```

# DNS: Split-horizon

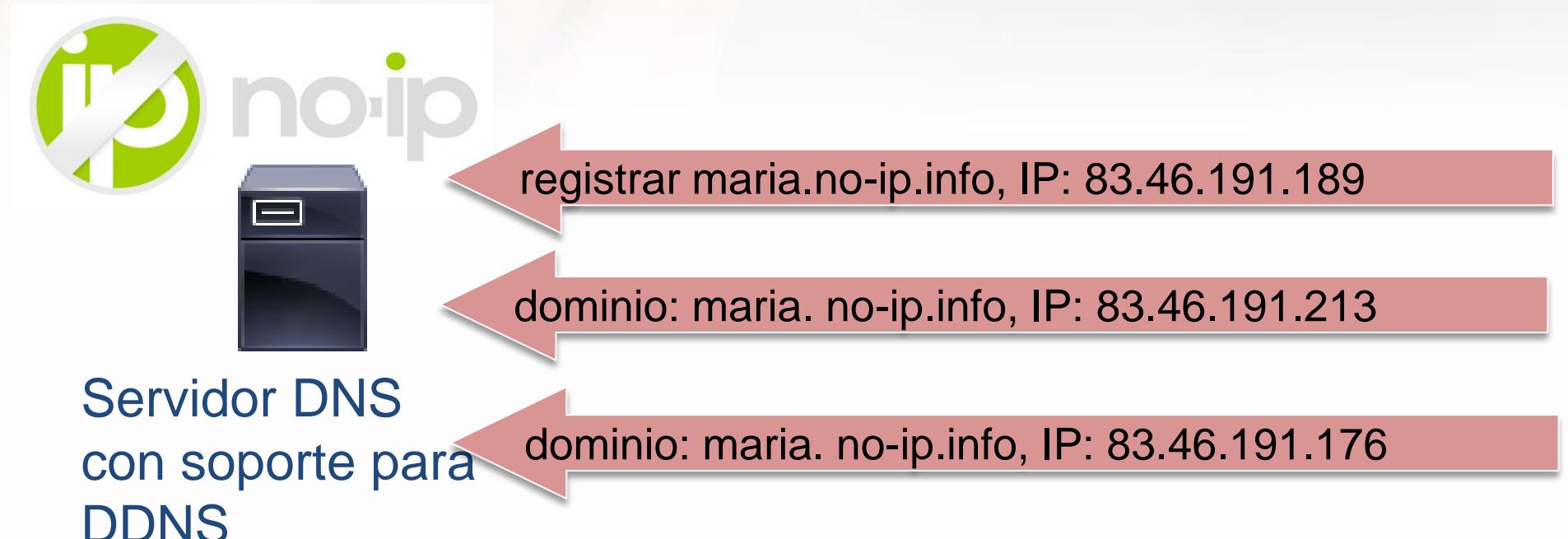
Split-Horizon: un nombre se resuelve con distintas IP dependiendo del origen de la consulta

```
$ host crystal.it.itba.edu.ar  
crystal.it.itba.edu.ar has address 10.16.1.103
```

```
$ host crystal.it.itba.edu.ar  
crystal.it.itba.edu.ar has address 200.5.121.139
```

# DNS dinámico

DDNS (RFC 2136) permite actualizar en forma dinámica un servidor DNS. Introducido en BIND a partir de la versión 8.



Usuario  
con IP  
dinámica

# DNS dinámico

## Ejemplo

```
user@server:~$ dig carlos.no-ip.com

...
;; ANSWER SECTION:
carlos.no-ip.com. 55 IN A 69.208.210.116

;; AUTHORITY SECTION:
no-ip.com. 86330 IN NS nf2.no-ip.com.
no-ip.com. 86330 IN NS nf3.no-ip.com.
no-ip.com. 86330 IN NS nf1.no-ip.com.

;; ADDITIONAL SECTION:
nf1.no-ip.com. 84798 IN A 8.4.112.75
nf2.no-ip.com. 84797 IN A 63.208.74.227
nf3.no-ip.com. 84797 IN A 216.66.37.13
```

# DNS spoofing

Es el arte de lograr que un registro DNS apunte a un IP que no sea el que debería apuntar.



La respuesta estaba en la cache o la aportó otro servidor DNS.

# DNS spoofing



Método: enviar respuesta sin recibir una pregunta



Rta: El IP de campus.itba.edu.ar es  
24.232.138.98



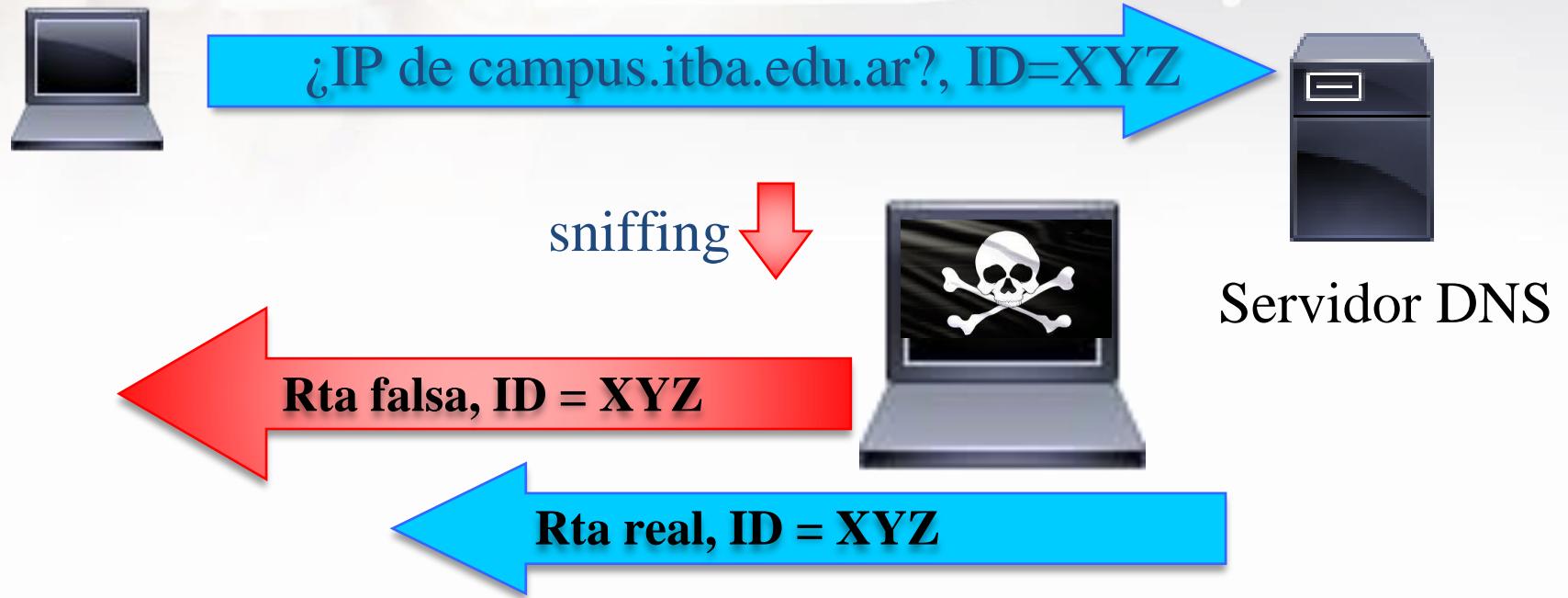
Servidor DNS  
Víctima

Se debe verificar que la respuesta se corresponda con alguna pregunta. DNS establece que los mensajes de pregunta y respuesta tengan un mismo ID que los identifique

# DNS spoofing



Método: *DNS sniffer*

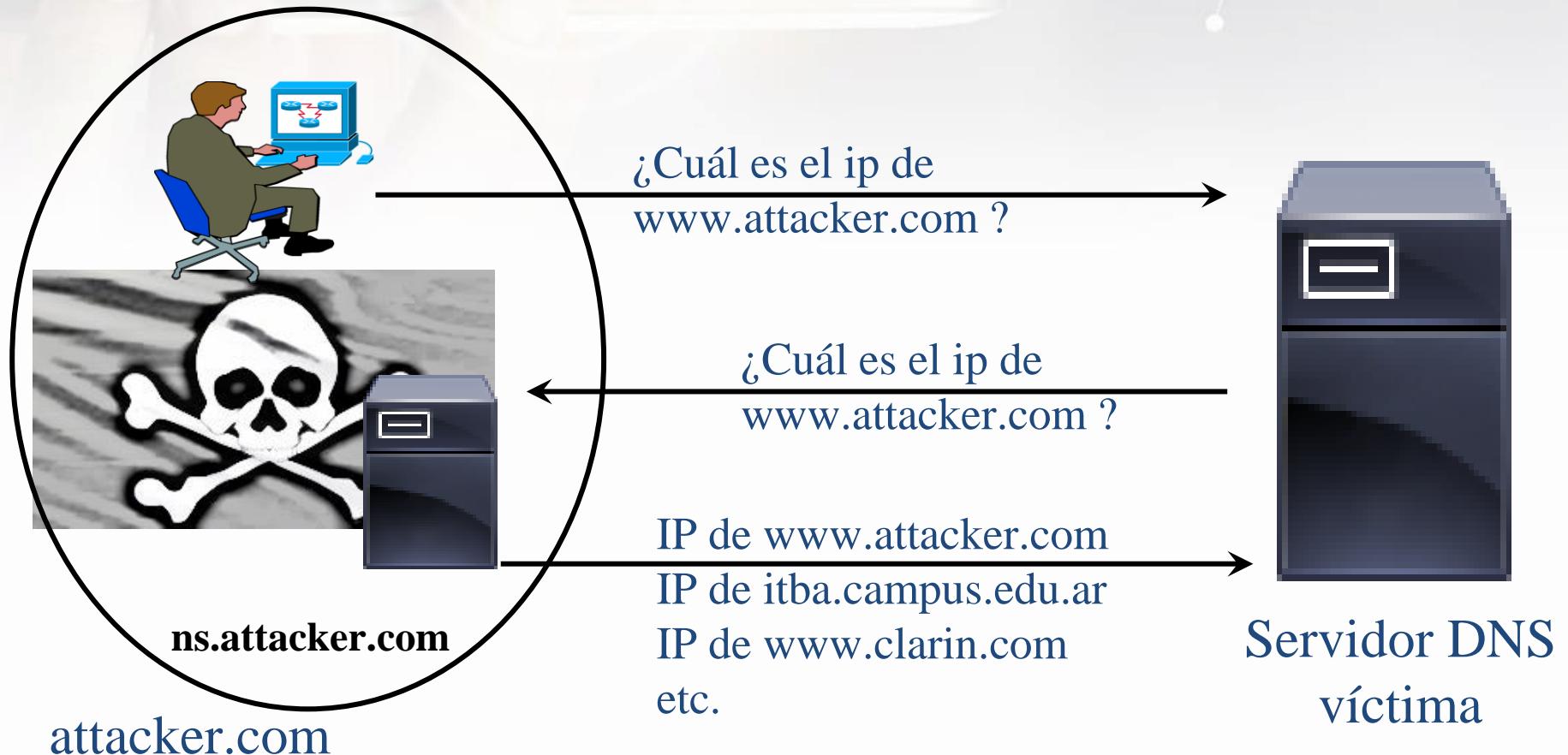


¿Sirve verificar el IP de origen de la respuesta?

# DNS spoofing



Método: *DNS cache poisoning*



# Anexo

Formato de los mensajes

Ver también las capturas publicadas en Campus

# DNS: formato de los mensajes

Message Header:

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1				
REQUEST ID															Q	Opcode		A	T	D	R	Rsvd		Rcode											
Count of Question Records															Count of Answer Records																				
Count of Authority Records															Count of Additional Records																				

## Opcode

- ◆ 0: consulta estándar
- ◆ 1: consulta inversa
- ◆ 2: consulta por “server status”

## Rcode

- ◆ 0: sin error
- ◆ 1: error de formato
- ◆ 2: error en el servidor
- ◆ 3: error en el nombre
- ◆ 4: sin implementar
- ◆ 5: rechazado

# DNS: formato de los mensajes

Message Header:

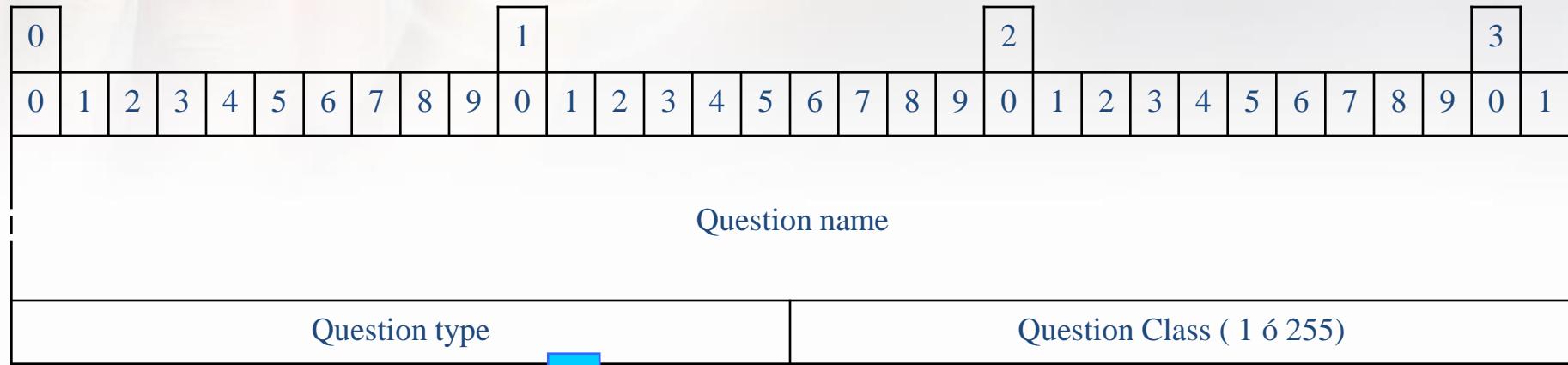
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1				
REQUEST ID															Q	Opcode		A	T	D	R	Rsvd		Rcode											
Count of Question Records															Count of Answer Records																				
Count of Authority Records															Count of Additional Records																				

## Bits

- ◆ Q: 0 (Query) / 1 (Response)
- ◆ A: Respuesta autorizada
- ◆ T: Respuesta truncada
- ◆ D: Pregunta con recursividad
- ◆ R: Recursión disponible

# DNS: formato de los mensajes

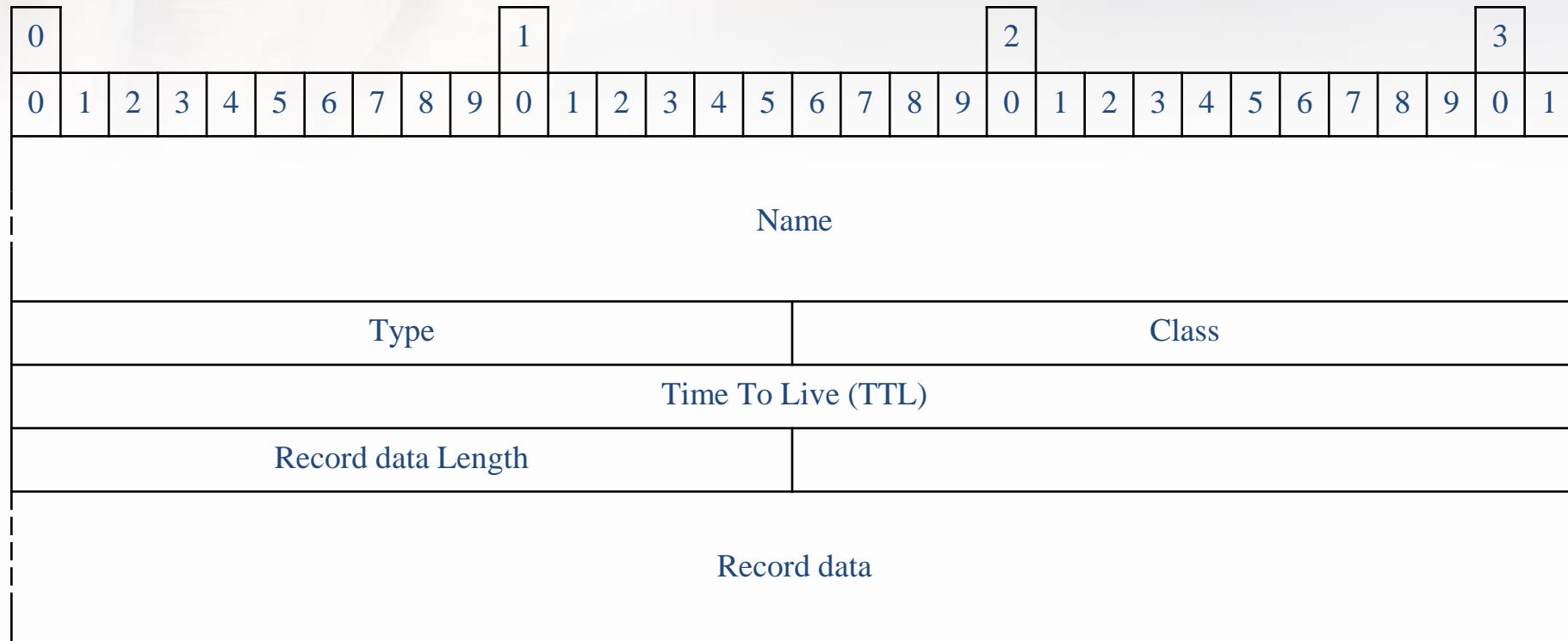
Mensaje de pregunta (Question record):



- ◆ 1: host
- ◆ 2: name server autorizado
- ◆ 5: nombre canónico de un alias
- ◆ 15: mail exchange
- ◆ 252: solicitud de transferencia de zona
- ◆ etc.

# DNS: formato de los mensajes

Mensaje de respuesta (Answer, Authority, Additional record):



# Material de lectura

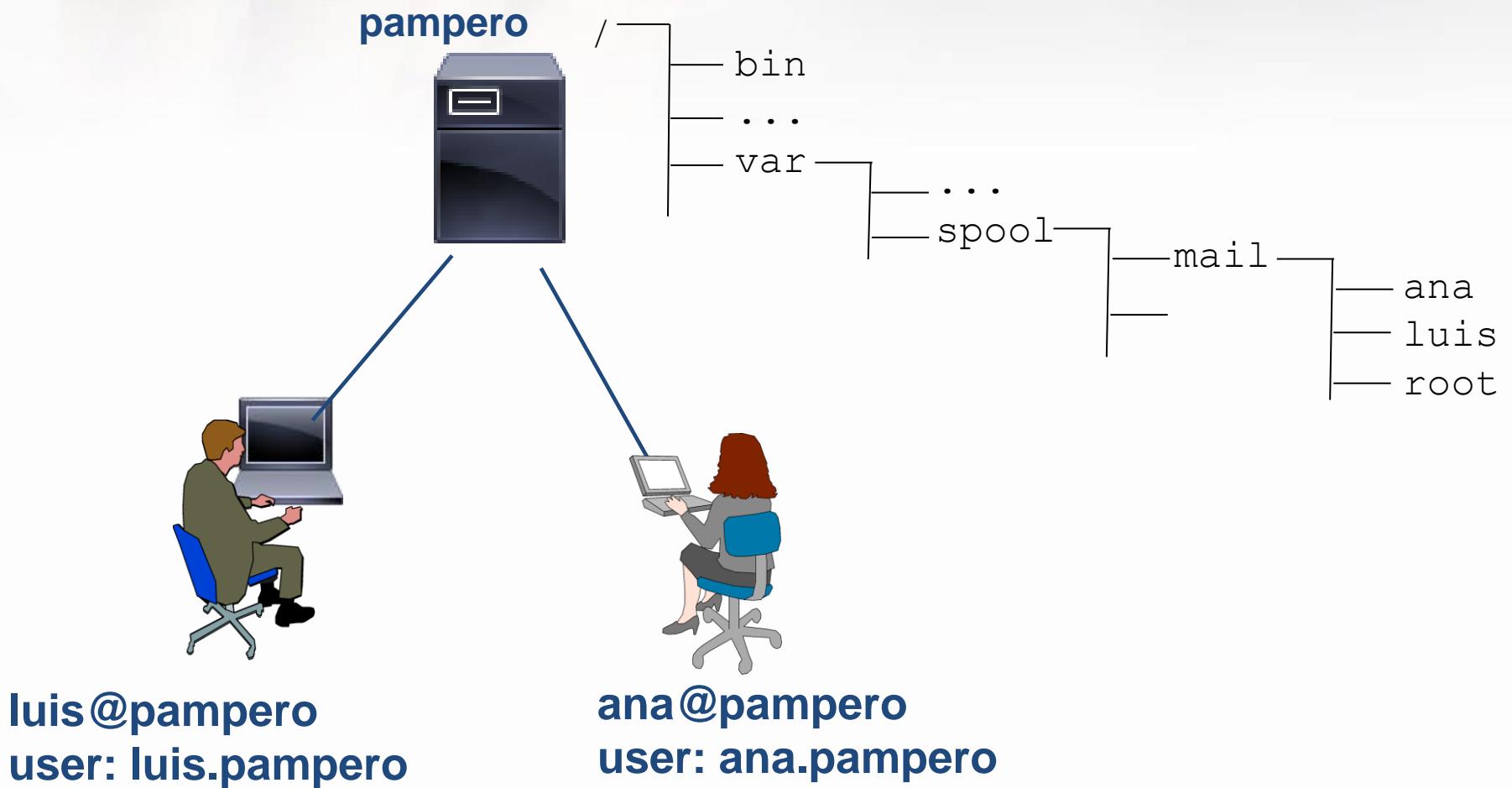
## Capítulo 2.5 de la bibliografía



# Correo electrónico

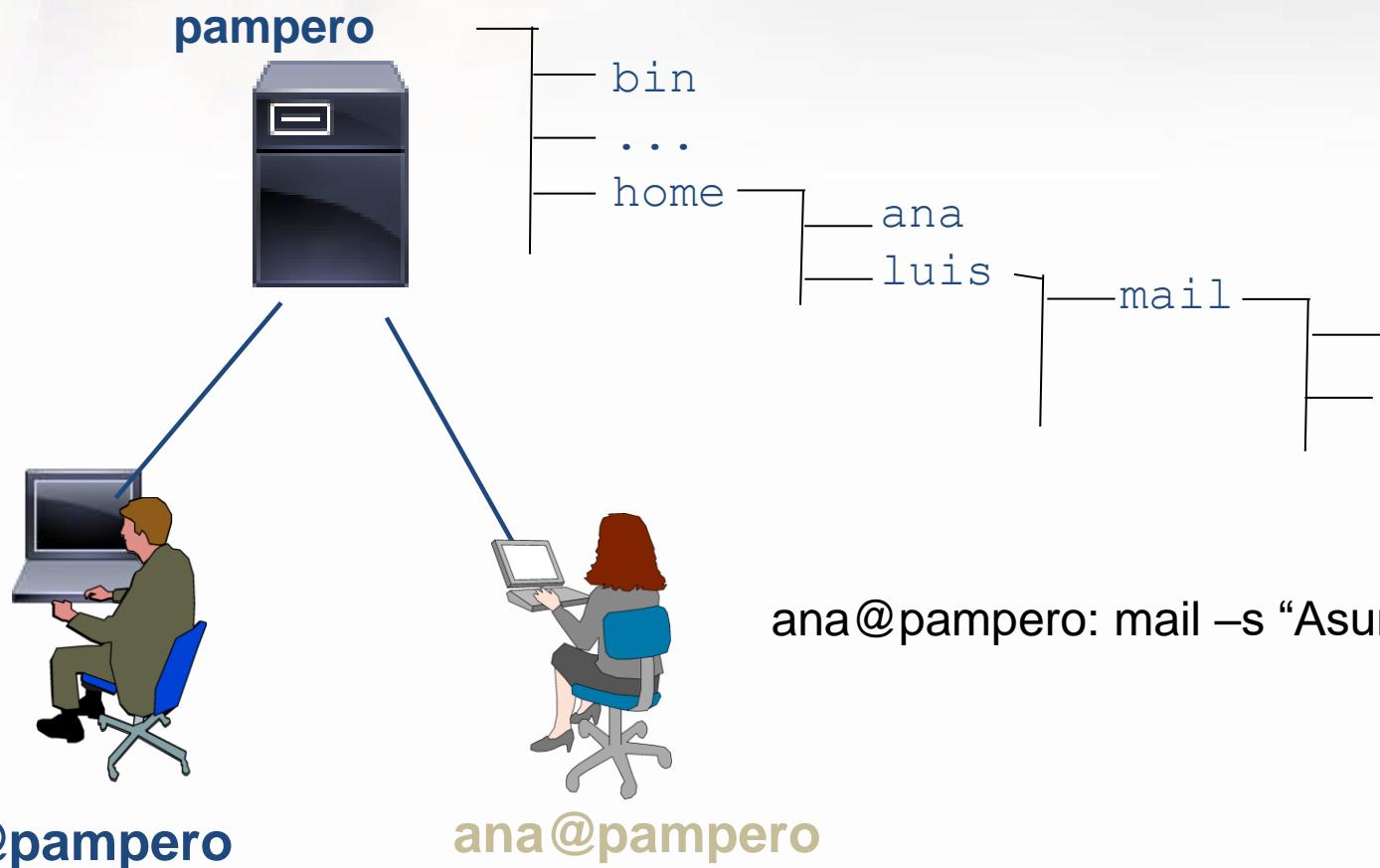
# Email

En un sistema Unix/Linux, toda cuenta de usuario tiene también una cuenta de mail.



# Email

En un sistema Unix/Linux, toda cuenta de usuario tiene también una cuenta de mail.



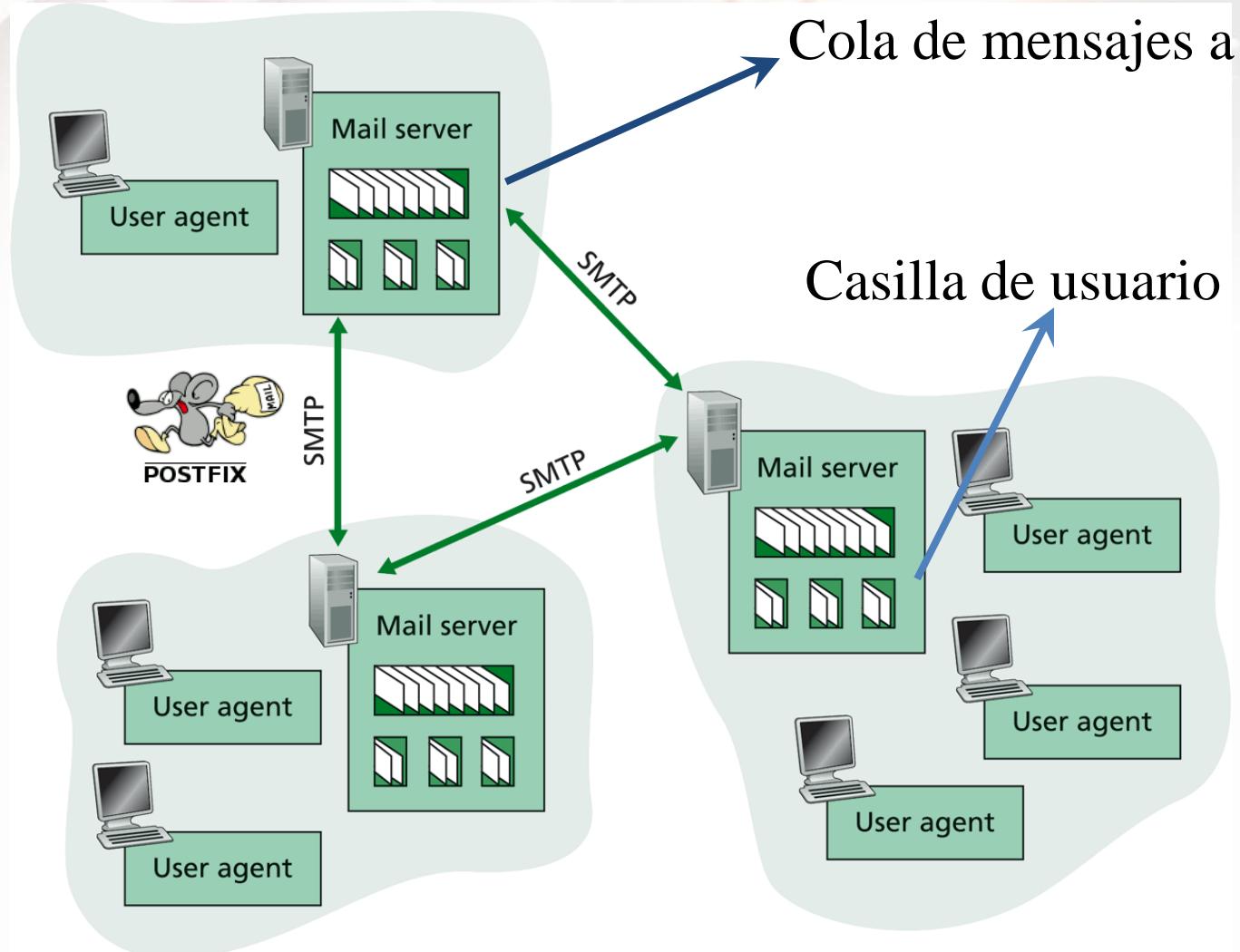
# Email: ejemplos

## Enviar y leer mails por línea de comando

```
$ mail -s "Subject" user  
$ mail -a attachFile -s "Subject" user,user2
```

```
$ mail  
"/var/spool/mail/mgarbe": 333 messages 332 new  
> N 1 .....  
? t 1  
texto del mensaje  
? q
```

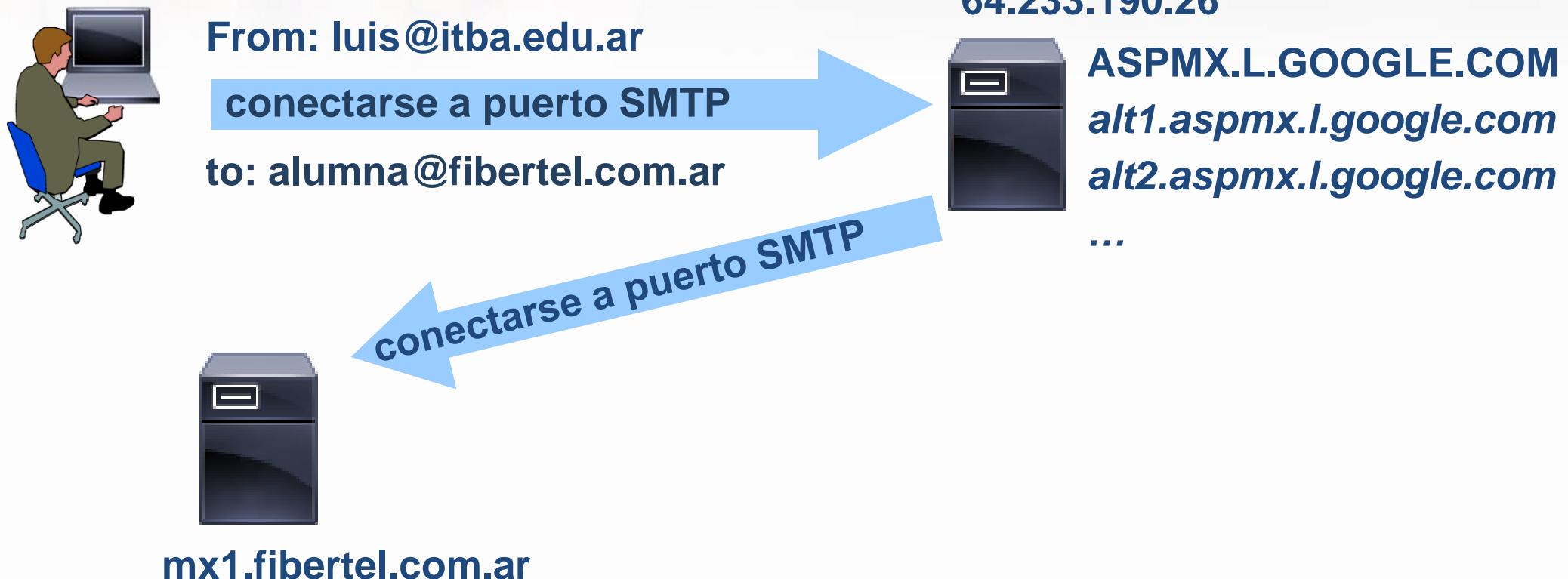
# Cómo enviar mails de un host a otro



- **MTA:** Mail Transfer Agent
  - Transfiere el mensaje entre hosts
- **MUA:** Mail User Agent 
  - permite al usuario leer y enviar mensajes
- **MDA:** Mail Delivery Agent
  - coloca el mensaje en la casilla de correo

# SMTP

El protocolo por el cual se transmiten los mails por Internet es SMTP (Simple Mail Transport Protocol), definido en RFC821(1982) y RFC5321(2008).

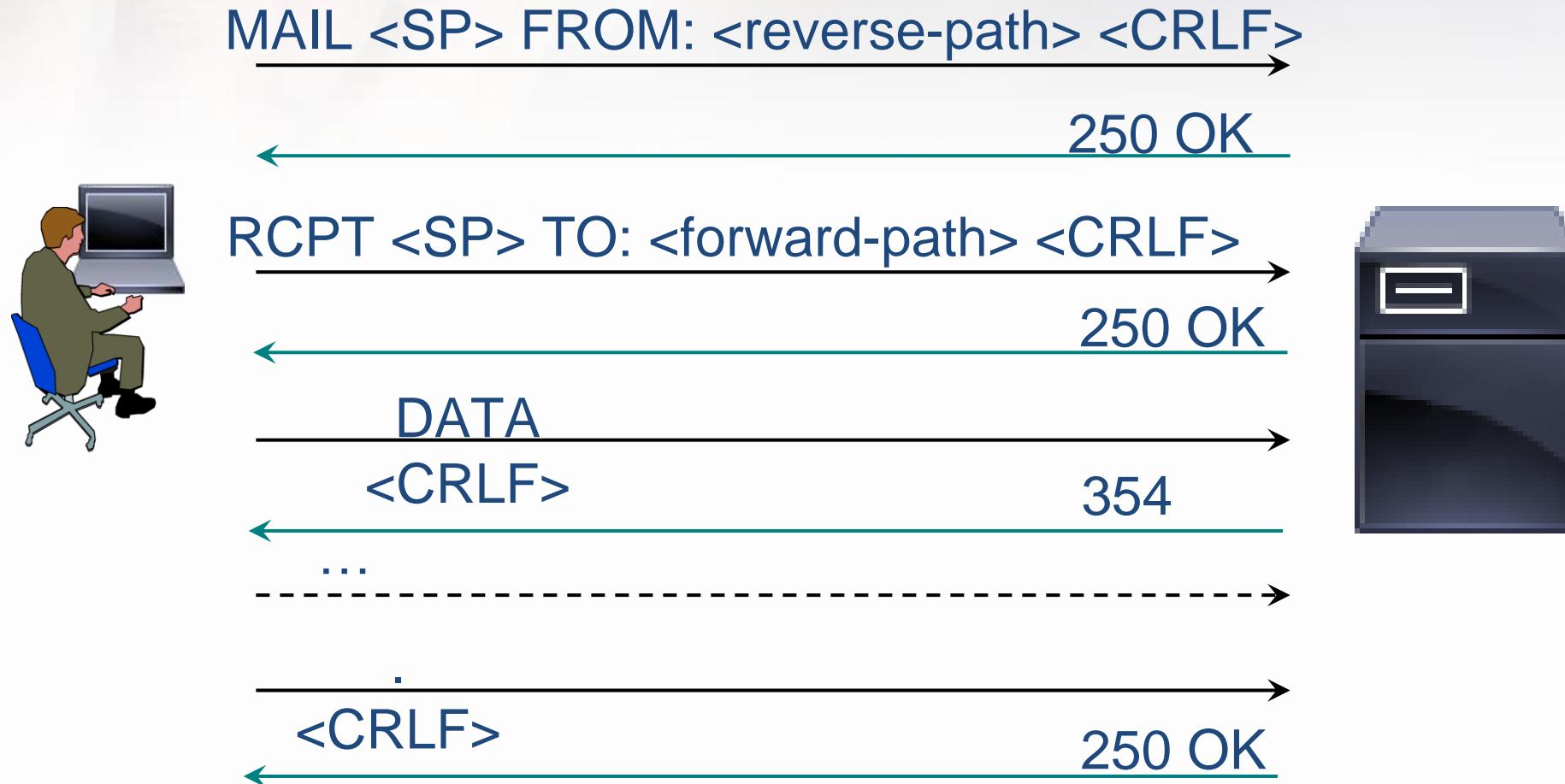


# SMTP

- ◆ Usa TCP para transferir mensajes desde el cliente al servidor (puerto 25)
- ◆ Transferencia directa: desde "sending server" a "receiving server"
- ◆ Tres fases
  - ◆ handshaking
  - ◆ transferencia de mensajes
  - ◆ cierre
- ◆ Interacción comando / respuesta
  - ◆ comando: texto ASCII
  - ◆ respuesta: código de status y comentario
- ◆ Los mensajes son US-ASCII: **ASCII-7 bits**

# SMTP

## Pasos en una transacción SMTP



# SMTP: ejemplo

S: MAIL FROM:<Smith@Alpha.ARPA>

R: 250 OK

S: RCPT TO:<Jones@Beta.ARPA>

R: 250 OK

S: RCPT TO:<Green@Beta.ARPA>

R: 550 *No such user here*

S: RCPT TO:<Brown@Beta.ARPA>

R: 250 OK

S: DATA

R: 354 *Start mail input; end with <CRLF>.<CRLF>*

S: Blah blah blah...

S: <CRLF>.<CRLF>

R: 250 OK

Ver ejemplo real de sesión completa en [SMTP: ejemplo](#)

# Undeliverable mail notification

S: MAIL FROM:<>

R: 250 ok

S: RCPT TO:<@HOSTX.ARPA:JOE@HOSTW.ARPA>

R: 250 ok

S: DATA

R: 354 send the mail data, end with .

S: From: SMTP@HOSTY.ARPA

S: To: JOE@HOSTW.ARPA

S: Subject: Mail System Problem

S: Sorry JOE, your message to SAM@HOSTZ.ARPA lost.

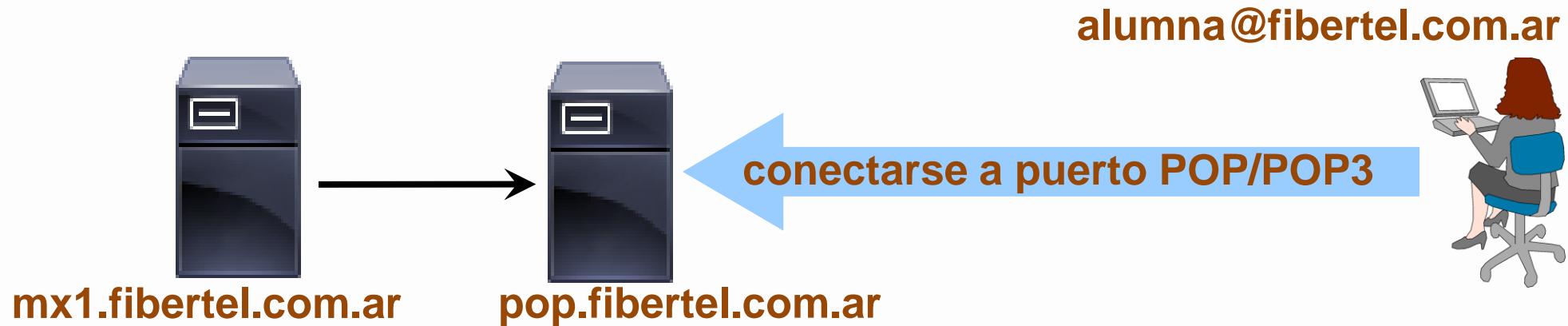
S: HOSTZ.ARPA said this: "550 No Such User"

S: .

R: 250 ok

# Leer mail en forma remota

En caso de querer acceder a los mensajes en forma remota, el sistema tiene que tener habilitado algún **protocolo de entrega final de usuario**, por ejemplo POP2 (puerto 109), POP3 (puerto 110), IMAP (puerto 143) o IMAP3 (puerto 220).



# Protocolos de entrega final a usuario

## ★ POP3 (Post Office Protocol)

- El objetivo de POP es obtener los mensajes del servidor y almacenarlos en el host local.
- Existen versiones que permiten dejar una copia en el servidor.

## ★ IMAP (Interactive Mail Access Protocol)

- Pensado para que un usuario consulte su correo desde varios hosts
- El servidor de correo mantiene un almacenamiento central accesible desde cualquier host.
- El cliente IMAP no copia el correo en el host local
- El cliente IMAP distingue si los mensajes ya han sido leídos
- Permite crear carpetas en el servidor
- El servidor realiza backups de los mails

# POP3



## Fase de autorización

- Comandos del cliente:
  - ❖ user: declara usuario
  - ❖ pass: password
- Respuestas del servidor
  - ❖ +OK
  - ❖ -ERR

## Transacciones, cliente:

- list: lista mensajes
- retr: obtiene mensajes por numero
- dele: borra mensajes
- quit

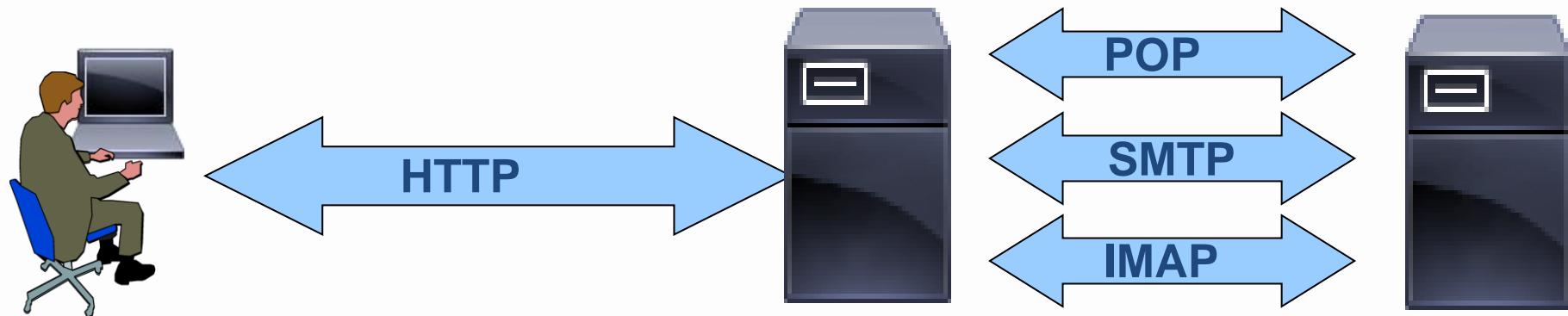
```
S: +OK POP3 server ready
C: user bob
S: +OK
C: pass hungry
S: +OK user successfully logged on
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

# POP3: comandos

Comando	Función
USER [username]	indicar el user name
PASS [password]	password del usuario
QUIT	cierra la conexión
STAT	retornar cantidad de mensajes y bytes
LIST	listar todos los mensajes
RETR <i>msgNumber</i>	pedir un mensaje
DELE <i>msgNumber</i>	marcar para borrar un mensaje
NOOP	
RSET	recuperar los mensajes marcados para borrado
TOP [ <i>msg</i> ] [ <i>lines</i> ]	header y primeras líneas de un mensaje

# Webmail

Es un cliente de correo electrónico (MUA) con una interface web, que permite acceder al mailbox de un usuario.



# IMAP: comandos

Comando	Función
LOGIN <i>user key</i>	Login
CAPABILITY	Lista las funcionalidades
LOGOUT	Cierra la conexión
SELECT <i>mailbox</i>	Selecciona un buzón
EXAMINE <i>mailbox</i>	Abre un mailbox como solo lectura
CREATE <i>mailbox</i>	Crea un nuevo buzón
DELETE <i>mailbox</i>	Elimina un buzón
RENAME <i>mbx1 mbx2</i>	Cambia el nombre de un buzón
CLOSE	Cierra el buzón y elimina los mensajes marcados para borrar
EXPUNGE	Elimina los mensajes marcados para borrar
FETCH <i>n mbx</i>	Muestra un mensaje del mailbox

# Formato de un mensaje

El formato de un mail, según el RFC 822, consta de

- ◆ Una envoltura primitiva (ver RFC 821).
- ◆ Campos de cabecera (From: ..., To: ..., etc.)  
(los usuarios pueden definir cabeceras que comiencen con X)
- ◆ Una línea en blanco
- ◆ Cuerpo del mensaje

El E-mail es probablemente la aplicación TCP/IP más usada.  
Sin embargo, SMTP y RFC 822 se limitan a texto ASCII de 7  
bits con una longitud de línea máxima de 100 caracteres.

# Encoding: Base64

- ◆ Método de codificación simple
- ◆ Cada grupo de 3 bytes es codificado como 4 bytes, cada uno conteniendo sólo 6 bits de datos
- ◆ Estos son enviados como “7-bit ASCII”
- ◆ Base64
  - ◆ 6 bits -> [0,63]
  - ◆ Se asigna un carácter a cada número

# Encoding: Base64

## Alfabeto Base64

0 A	17 R	34 i	51 z
1 B	18 S	35 j	52 0
2 C	19 T	36 k	53 1
3 D	20 U	37 l	54 2
4 E	21 V	38 m	55 3
5 F	22 W	39 n	56 4
6 G	23 X	40 o	57 5
7 H	24 Y	41 p	58 6
8 I	25 Z	42 q	59 7
9 J	26 a	43 r	60 8
10 K	27 b	44 s	61 9
11 L	28 c	45 t	62 +
12 M	29 d	46 u	63 /
13 N	30 e	47 v	
14 O	31 f	48 w	(pad) =
15 P	32 g	49 x	
16 Q	33 h	50 y	

# Base64: ejemplo

3 bytes a codificar: 10101111

11001010

11101010

Stream de 24 bits: 101011111100101011101010

Agrupamos de a 6 bits: 101011 111100 101011 101010

Valor decimal 43 60 43 42

Caracter Base64 r 8 r q

Se transmite: 01110010 00111000 01110010 01110001

# Base64: relleno



¿Qué sucede si la cantidad de bits no es múltiplo de 6?

- Se agregan ceros al final hasta un múltiplo de 6.
- Uno o dos caracteres de relleno ('=') son agregados para que sea múltiplo de 6 bytes.
- En general se agrega un solo carácter de relleno

# Base64: relleno

4 bytes a codificar: 10101111 11001010 11101010 00100011

Stream de 32 bits: 1010111110010101110101000100011

Agrupado de a 6 bits: 101011 111100 101011 101010 001000 110000

Valor decimal	43	60	43	42	08	48
---------------	----	----	----	----	----	----

Caracter Base64	r	8	r	q	I	w
-----------------	---	---	---	---	---	---

Relleno	r	8	r	q	I	w	=	=
---------	---	---	---	---	---	---	---	---

Ver <https://www.base64encode.net>

# MIME (RFC1521 y RFC1522)

- SMTP no puede transmitir objetos binarios.
- SMTP no puede transmitir texto que incluya caracteres nacionales
- Los servidores SMTP pueden rechazar los mensajes que superen un tamaño concreto.
- Algunas implementaciones de SMTP u otros MTAs de Internet no respetan por completo el estándar SMTP. Algunos problemas son:
  - Eliminación de espacios al final de la línea
  - Relleno de todas las líneas de un mensaje para que tengan la misma longitud.
  - Conversión de caracteres TAB a múltiples caracteres SPACE.

# MIME



MIME			NFS	
SMTP	POP	DNS	RPC	TFTP
TCP		UDP		
IP				

# MIME



MIME (*Multipurpose Internet Mail Extensions Encoding*) respeta el RFC 822. Agrega una estructura al cuerpo del mensaje, por lo que es transparente para SMTP. Permite:

- ◆ Normalizar intercambio de diferentes tipos de contenido (texto simple, texto formateado, imágenes, sonido, video y documentos HTML).
- ◆ Solucionar el problema de enviar texto internacional por mail.

# MIME: encabezados

- ◆ MIME-version: 1.0 (*comentario*)
- ◆ Content-type
- ◆ Content-Transfer-Encoding
  - ◆ 7 bits
  - ◆ 8 bits
  - ◆ binario
- ◆ Content-Description
- ◆ Content-ID

# MIME:content-type

([https://www.w3.org/Protocols/rfc1341/0\\_TableOfContents.html](https://www.w3.org/Protocols/rfc1341/0_TableOfContents.html))

Content-Type : *type/subtype ;parameter=value ;parameter=value*

Type	subtype
text	html, plain, richtext
multipart	mixed, alternative, digest
message	Partial, external-body
image	gif, jpeg, tiff
video	mpeg, quicktime
audio	basic, x-aiff, x-wav
application	pdf, rtf, postscript

# MIME: ejemplo parcial

...lo dicho, Industrial and Commercial Bank of China (Argentina) S.A. no resulta responsable por dichas modificaciones ni por los eventuales perjuicios que tales circunstancias puedan ocasionar.

```
-----_NextPart_000_00A5_01C0F442.AAB436D0 Content-Type: application/pdf; name="Adjunto_1_OP.pdf"
Content-Disposition: attachment; filename="Adjunto_1_OP.pdf" Content-Transfer-Encoding: base64
JVBERi0xLjMKJaqrrK0KNCAwIG9iago8PCAvVHlwZSAvSW5mbwovUHJvZHVjZXIgKG51bGwpID4+
CmVuZG9iago1IDAgb2JqCjw8IC9MZW5ndGggMjQ3NiAvRmlsdGVyIC9GbGF0ZURIY29kZSAKID4+
CnN0cmVhbQp4nJ1bW3PaShJ+96/gMalaK3O/5E3G2IdG3sxJy/rfSCg+LBfkBdwqs7++u2RhGYE
uvScpCqR5U9fX6a7p3sEN4uryeLqv1dk9HbFVUJGWjD4VwoC/7qrftb6cfXPq5sQJ+A+pzyRBUJw
AlctOKkVDsg9rl+wLhSkDI1wd+GkwxF6kkvbxQqQiMGVYgtYm9gvd3xELdxa/Lhyyi9WxVN0ROAv
HVECUrQeaSYTy8Vo8XP0r0+Pk9uHp/vPo2umDTzyKZ3fT2aL6Sz1t16SNPk8+vdo8fdQY1I7gKoe
T2FNkxrrKo9rE1z4gJQ+OBkuuEk4A7t1Ik52P81vJzNv4+3EXz+n909fK4O/3NELOsIUoIST7quy
jNbPtKjASMKkHGn4Re37u8n4t7Rdh8nj9GX6NOtVA5ZakzNKChGiE0YYOV+xgdhphqzip1TRpG9d
h3GF2BLWFbL6zFUK0XKkjA6CNFtv87cgSPdv2e642S1bg/SMSDNPNF5ut9nX8R+bbeYfNYS2PykD
Fcb5evOWB0GSH47L7VdKrPE3bz6yXX4I9Nzss0M7twi0CsxpzbLOYI9ejTLLMKvrcW2Ca2OouVis
8e/TRRi358sLESEoaz7DyTWEsWXG6GsRpNH5s5W4cEln+2BV1sGyTndv4P1wOW72H8f80KebNYnR
```

...

# SPAM

(<https://www.youtube.com/watch?v=zLih-WQwBSc>)



# SPAM

- 
1. Obtener direcciones de correo
  2. Envío de mensajes
    - Troyanos, bots
    - Servidores SMTP que aceptan relay
  3. Verificación de la recepción (a veces)

# SPAM: cómo defenderte

- ◆ Evitarlos
  - ◆ Listas negras y listas blancas      *Deny list y Allow list*
  - ◆ GreyListing
- ◆ Clasificarlos
  - ◆ Filtros Bayesianos

# Autenticación de mails

Según Verizon el 30% de los mails de phishing se abren, y en hasta un 12% de ellos se accede a los enlaces o archivos maliciosos

- ◆ Hay dos técnicas principales para autenticar mails
  - ◆ SPF (Sender Policy Framework)
  - ◆ DKIM (DomainKeys Identified Mail)

# SPF

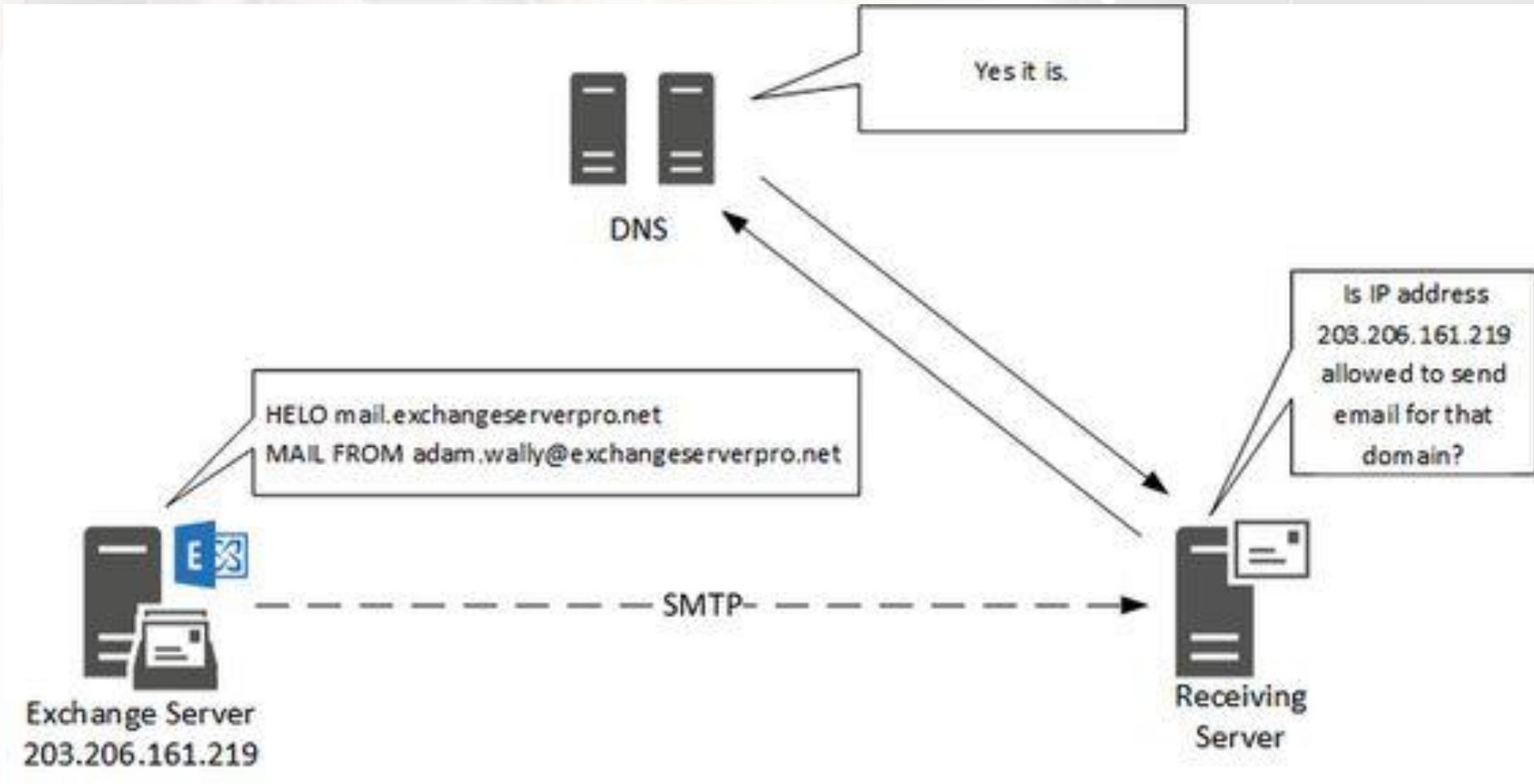


Permite detectar qué servidor o servidores tienen permiso para enviar mails en su nombre.

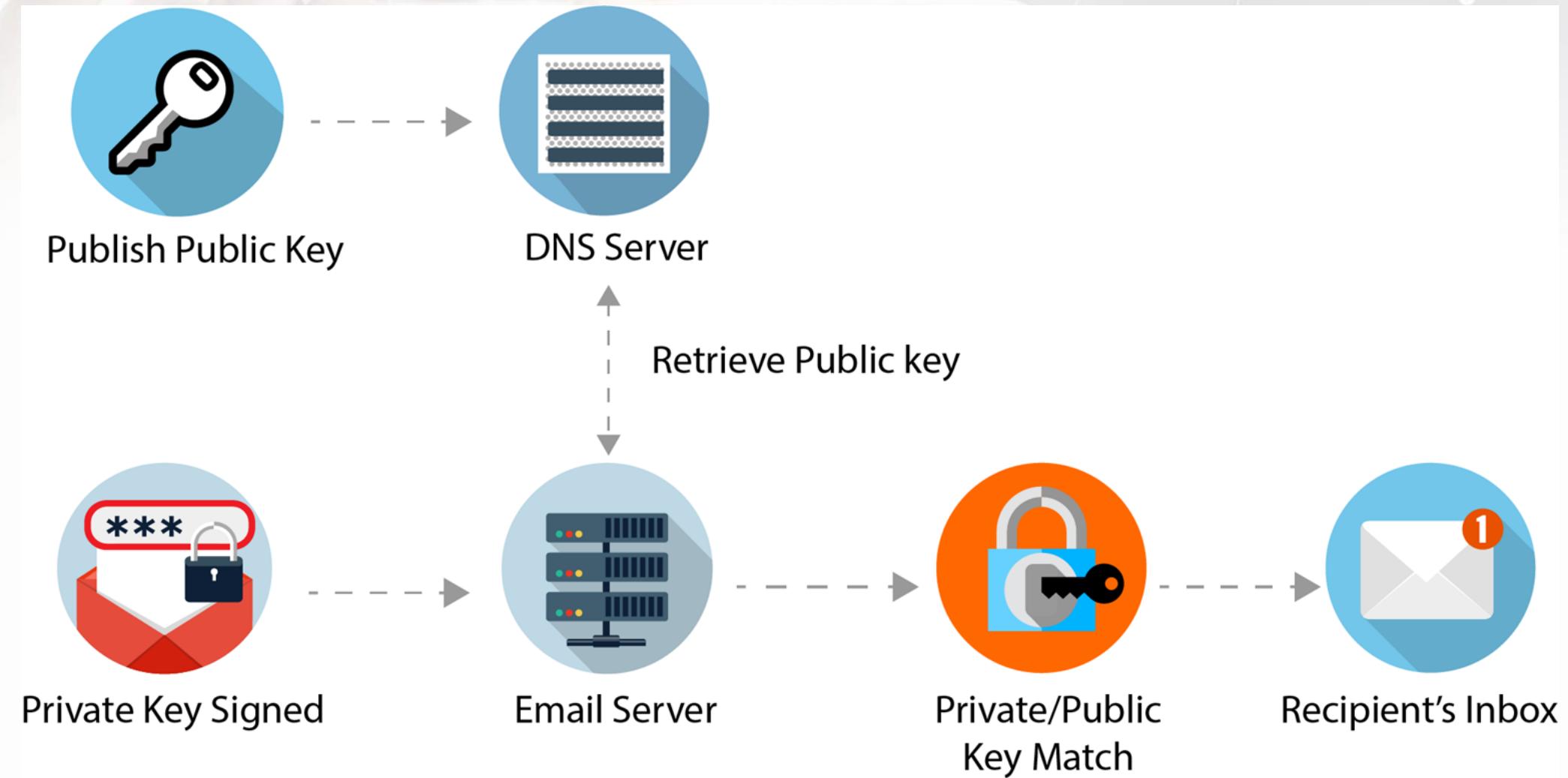
Su función es evitar que un mail sea enviado desde servidores que no estén relacionados con el dominio.

Se utilizan registros TXT y SPF en el servidor DNS

# SPF



# DKIM



# Material de lectura

## Capítulo 2.4 de la bibliografía

<https://postmarkapp.com/guides/dkim>

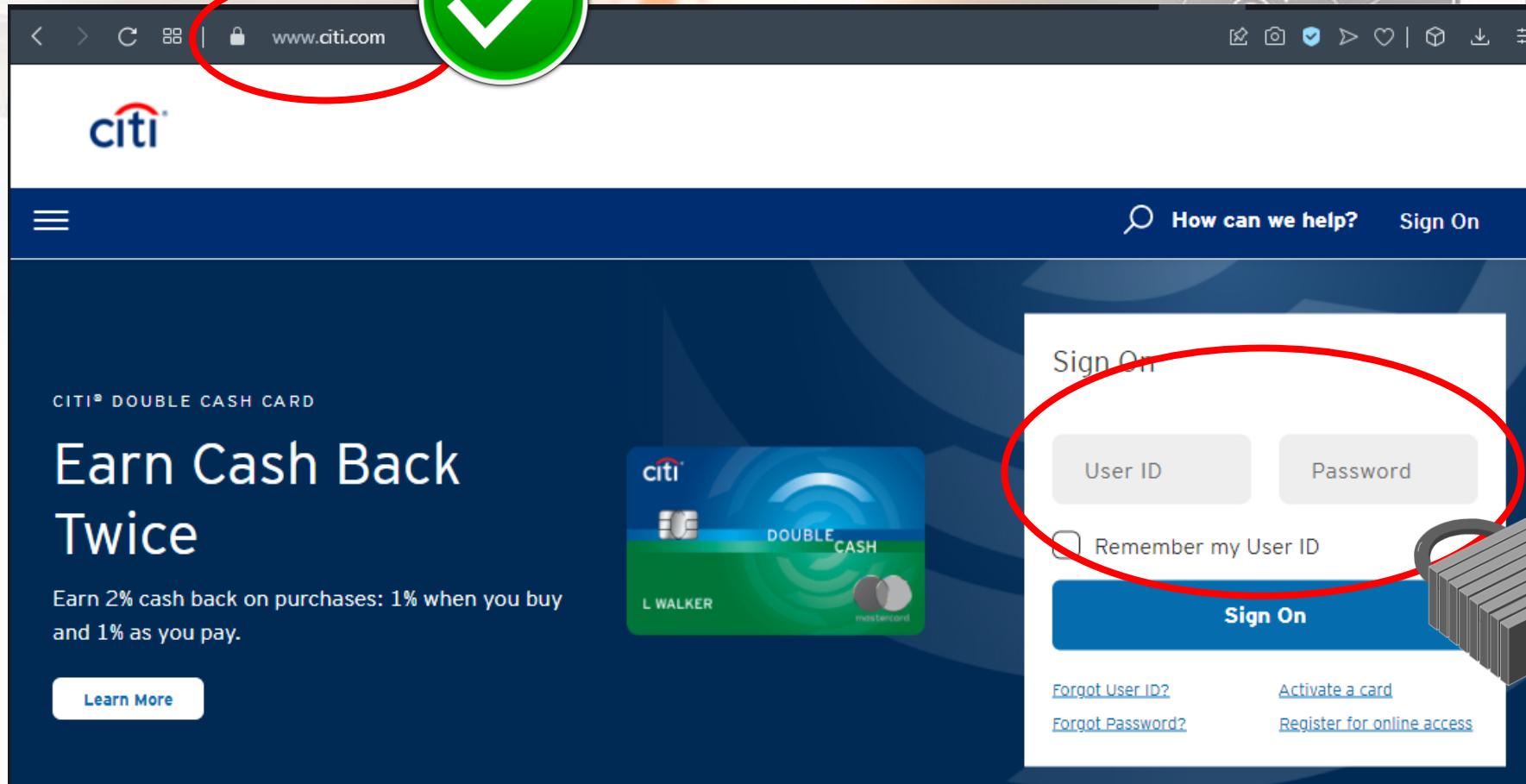
<https://postmarkapp.com/guides/spf>

<https://www.sparkpost.com/resources/email-explained/dmarc-explained/>



**TLS / SSL**

# SSL



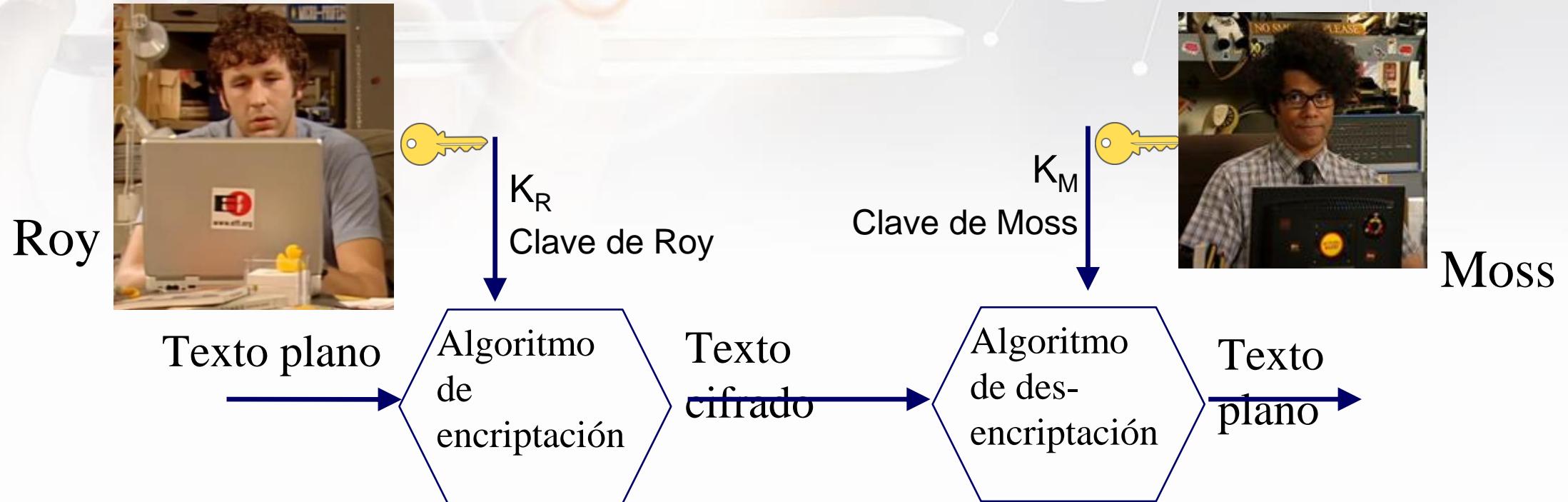
Para encriptar datos desde y hacia el servidor, un Web Server puede usar SSL (Secure Sockets Layer).



## Secure Sockets Layer

- ◆ Desarrollado por Netscape en 1994
- ◆ RFC 6101 define SSL 3.0 (Agosto 2011)
- ◆ Una capa intermedia entre protocolos de aplicación y de transporte (TCP)
- ◆ Permite a un servidor que "hable" SSL autenticarse a sí mismo con un cliente que "hable" SSL
- ◆ Permite a un cliente autenticarse con un servidor
- ◆ Permite encriptar la conversación entre cliente y servidor

# Nociones básicas de criptografía

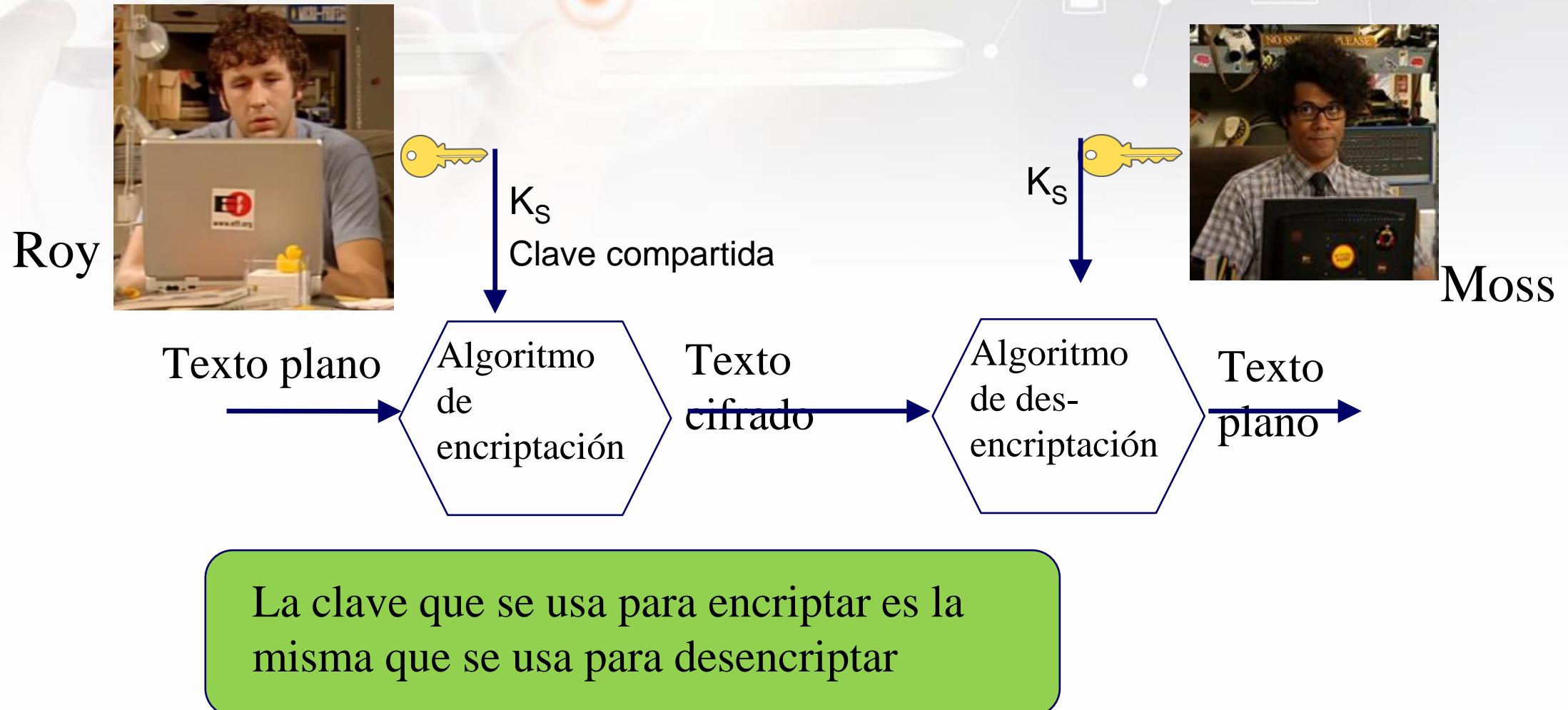


M: mensaje en texto plano

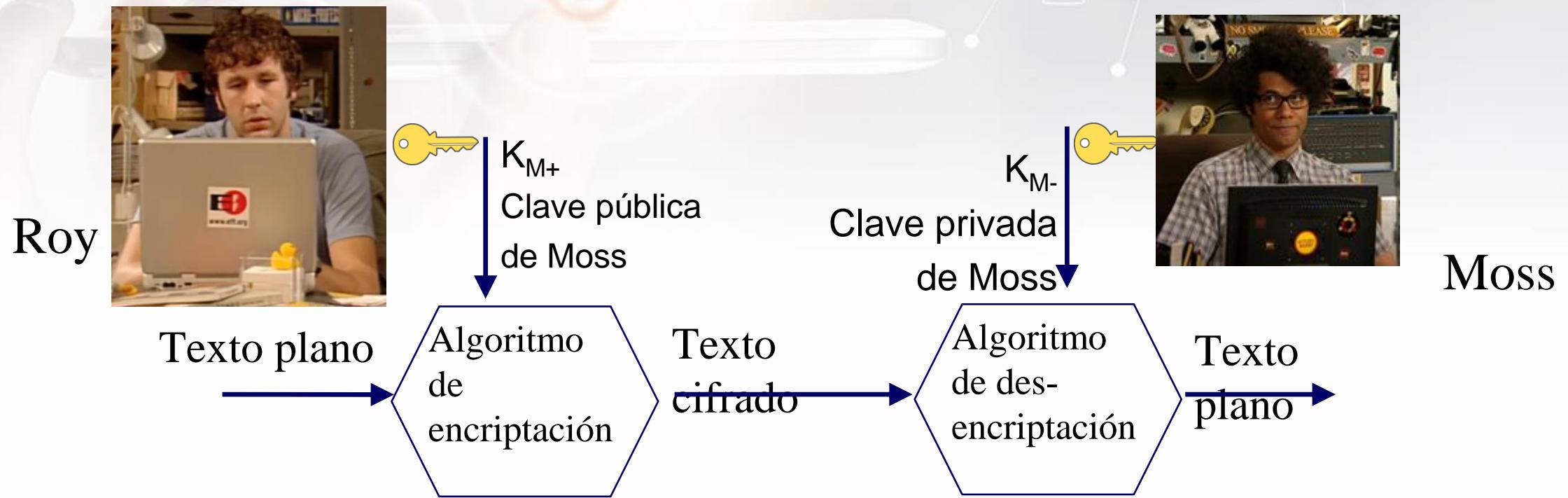
$K_R(M)$ : mensaje cifrado con clave  $K_R$

$$M = K_M(K_R(M))$$

# Clave simétrica

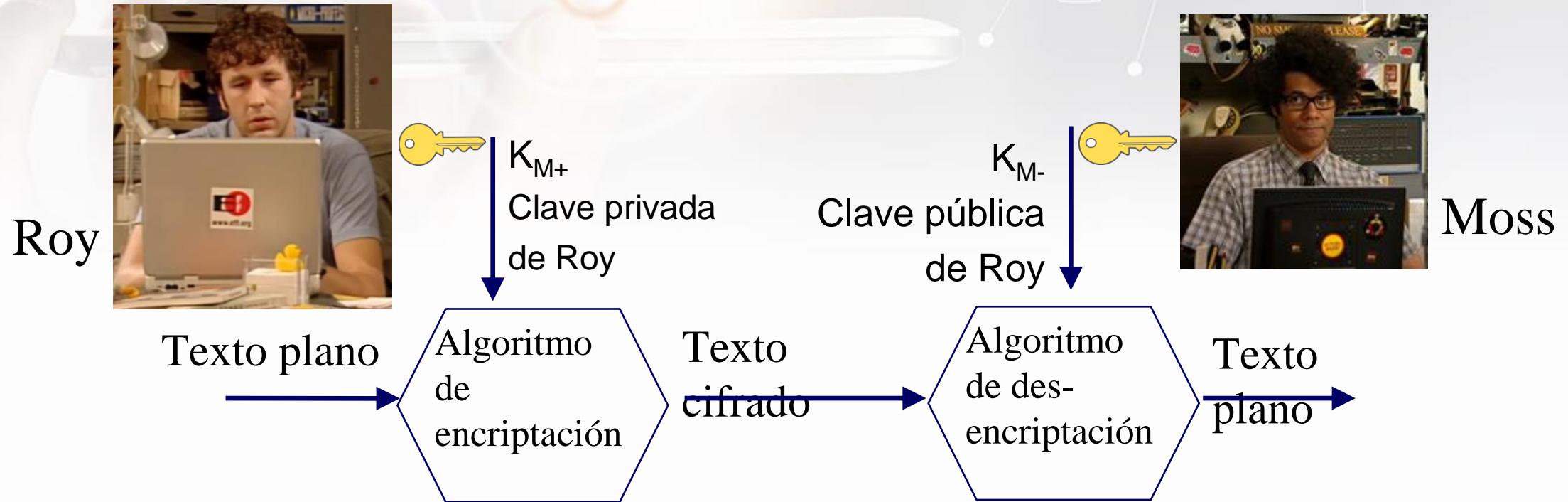


# Claves asimétricas



Un mensaje codificado con la clave pública de Moss sólo podrá desencriptarse con la clave privada de Moss.

# Clave asimétrica



Un mensaje codificado con la clave privada de Roy sólo podrá desencriptarse con la clave pública de Roy.

# Claves públicas

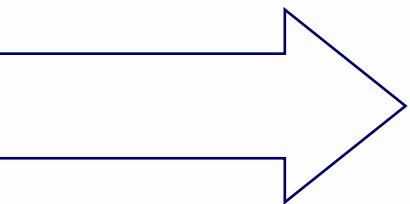


¿Cómo hace Roy para asegurarse que habla con Moss y no con alguien que se hace pasar por él?



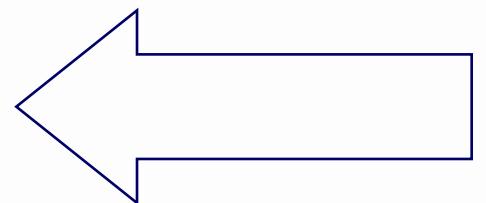
Clave pública de Moss

*PassPhrase*



Clave privada de Moss

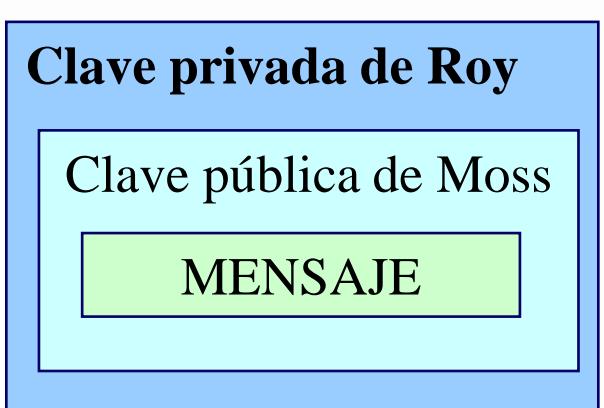
*PassPhrase*



# Claves públicas



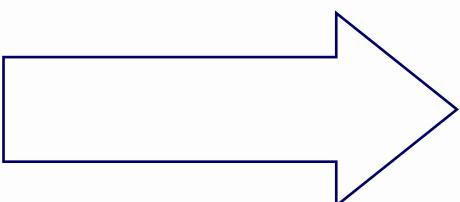
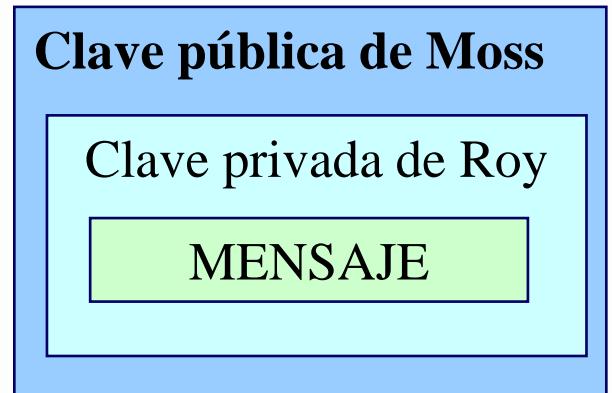
¿Cómo hace Roy para enviar un mensaje de forma tal que sólo lo pueda leer Moss y se asegure que sea de Roy?



# Claves públicas



¿Cómo hace Roy para enviar un mensaje de forma tal que sólo lo pueda leer Moss y se asegure que sea de Roy?





## Creación de certificados

- I. Crear una clave pública y privada
- II. Crear un certificado que incluya la clave del servidor
- III. Firmar el certificado, por un CA reconocido o por uno mismo.

# ¿Por qué aparece a veces?

RESOURCE

SEARCH



CONTENT

CONTENT

WEBSITE



The server's host key is not cached in the registry. You have no guarantee that the server is the computer you think it is.

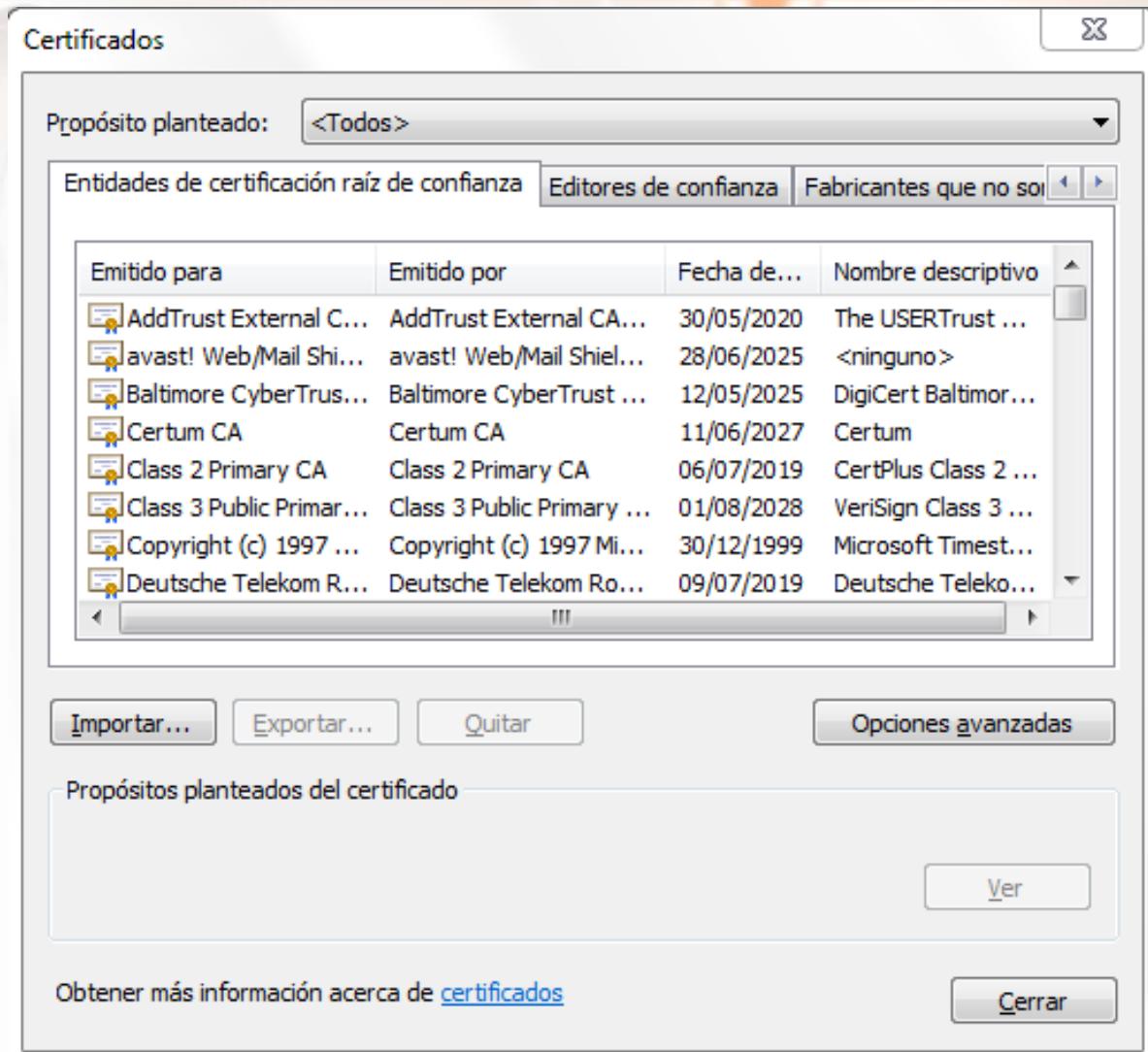
The server's ssh-ed25519 key fingerprint is:

ssh-ed25519 255 2e:76:f8:f6:0e:f8:b0:0e:84:19:35:b1:e3:19:8b:cc

If you trust this host, hit Yes to add the key to PuTTY's cache and carry on connecting.  
If you want to carry on connecting just once, without adding the key to the cache, hit No.

If you do not trust this host, hit Cancel to abandon the connection.

# SSL: Listado de entidades certificantes.



Los certificados incluyen:

- La clave pública de la autoridad
- Los datos de la autoridad (nombre, e-mail)
- Período de vigencia
- Identificación del firmante
- Firma digital del firmante

# SSL handshaking



Solicitud de conexión segura ( https://.....)

Envío de certificado X.509 conteniendo clave pública

*Cliente verifica autenticidad del certificado*

Cliente genera clave simétrica aleatoria y la  
encripta usando clave pública del servidor

Cliente y servidor conocen la clave simétrica y  
encriptan los datos con ella durante la sesión





## Transport Layer Security

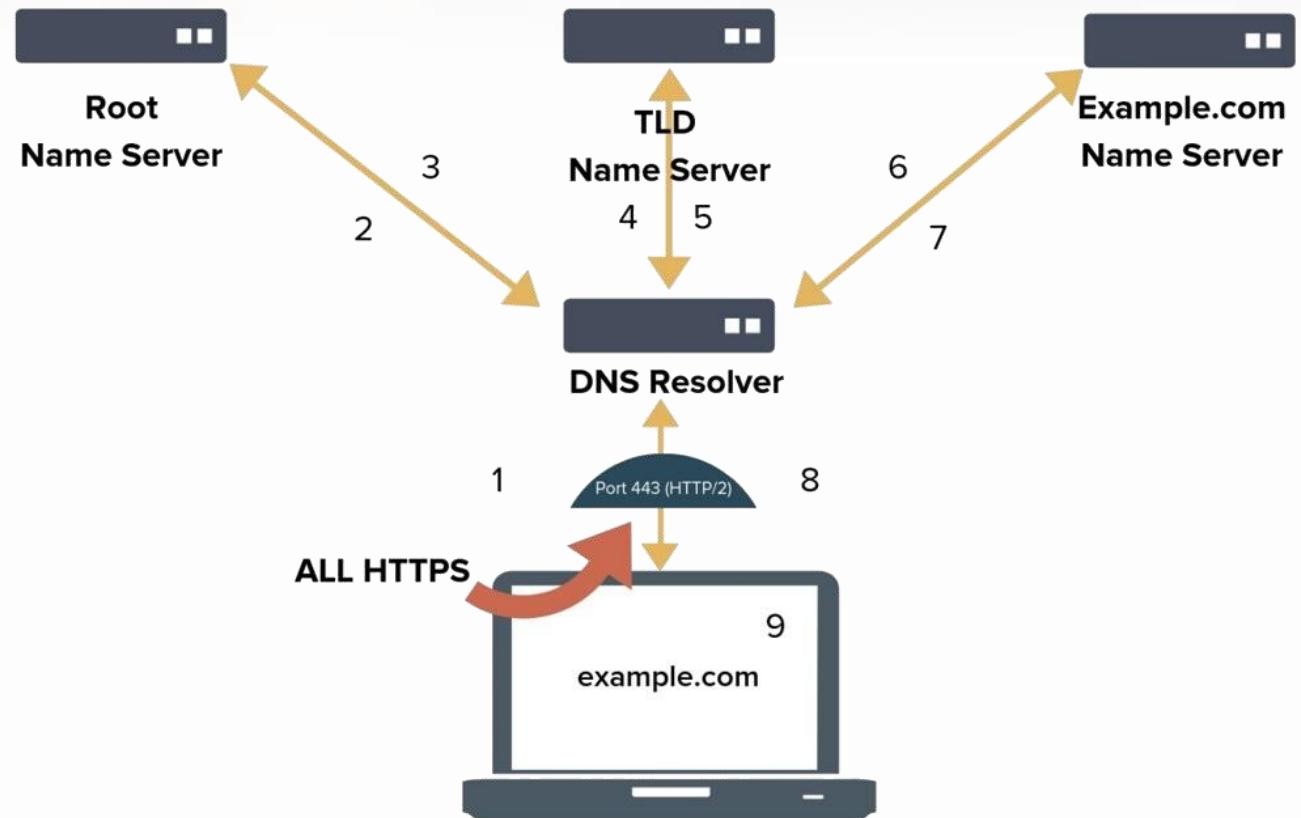
- ◆ TLS 1.0 está basado en SSL 3.0 (a veces TLS es llamado SSL 3.1)
- ◆ RFC 5246 define TLS 1.2
- ◆ Conexión segura por puerto: 443 para "*https*", 993 para "*IMAPs*", 995 para "*POPs*", etc. Usa SSL
- ◆ Conexión segura por protocolo: comienza con un "hello" inseguro y luego cambia a una conexión segura. Ejemplo: comando STARTTLS en SMTP
- ◆ TSL y SSL proporcionan el mismo nivel de encriptación. Difieren en cómo se inicia la conexión segura.

# DNS over TLS (DoT)

- ◆ Por defecto el puerto 853
- ◆ Incorporado en forma nativa en algunos Linux
- ◆ Dos modos:
  - ◆ modo estricto
  - ◆ modo de privacidad oportunista
- ◆ Soportado por algunos servidores:
  - ◆ ver <https://dnsprivacy.org/wiki/display/DP/DNS+Privacy+Public+Resolvers>

# DNS over HTTPS (DoH)

- ◆ Se configura en cada browser que lo soporte (en Firefox está por defecto)
- ◆ RFC 8484
- ◆ El cliente usa GET o POST para enviar una query DNS codificada

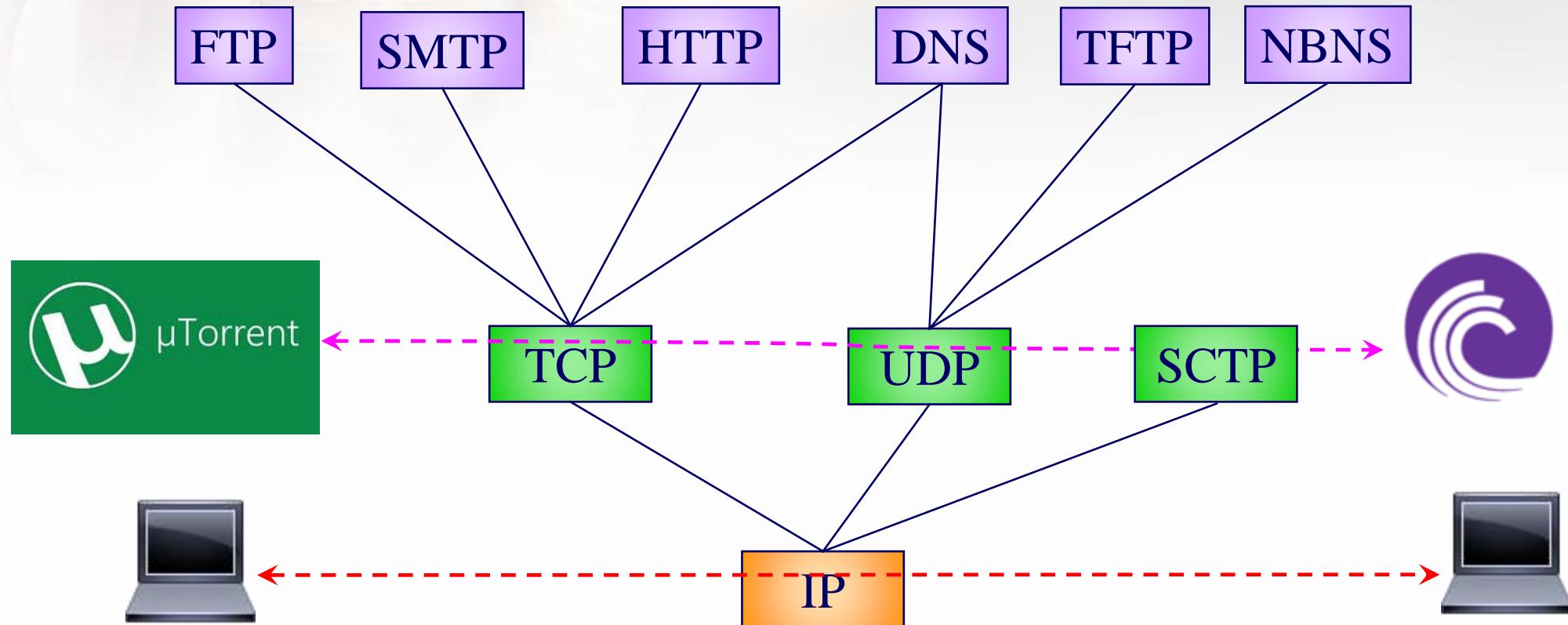




# Protocolos de transporte

- ◆ Servicios
- ◆ Multiplexación
- ◆ UDP
- ◆ TCP

# Protocolos de transporte



# Servicios

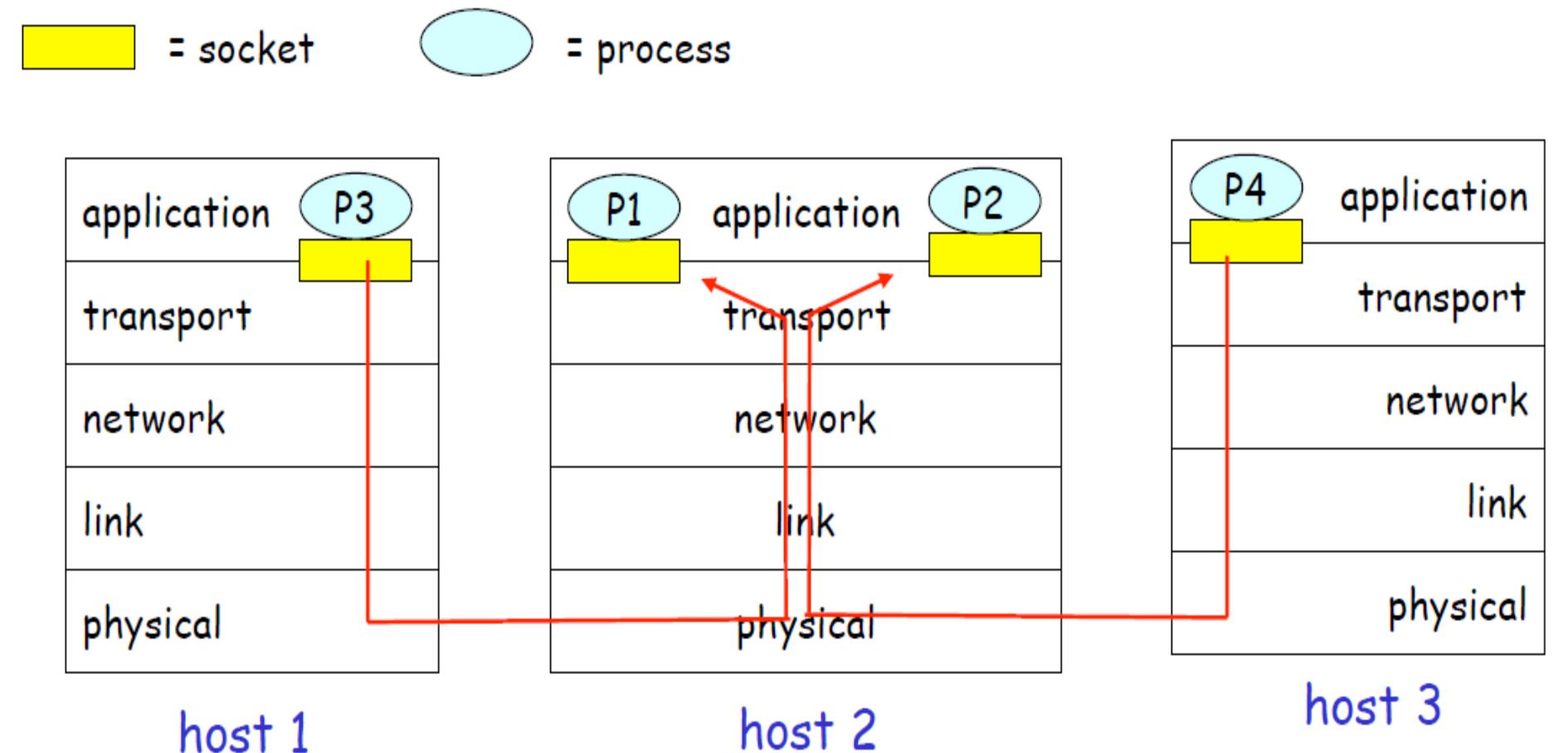


- ◆ Provee la comunicación lógica entre dos procesos que corren en distintos hosts
- ◆ Ejecutan en las «puntas finales»
  - ◆ El que envía: separa los mensajes en segmentos y los entrega al nivel de red
  - ◆ El que recibe: reensambla los segmentos en mensajes y los pasa al nivel de aplicación
- ◆ Protocolos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

# Servicios

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

# Multiplexación / Demultiplexación



# DeMultiplexación



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

# Multiplexación: puertos

Números de puerto (RFC 1700)

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

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

# Multiplexación: puertos

Ejemplo de algunos puertos.

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

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

# Multiplexación: puertos

Ejemplo de algunos puertos.

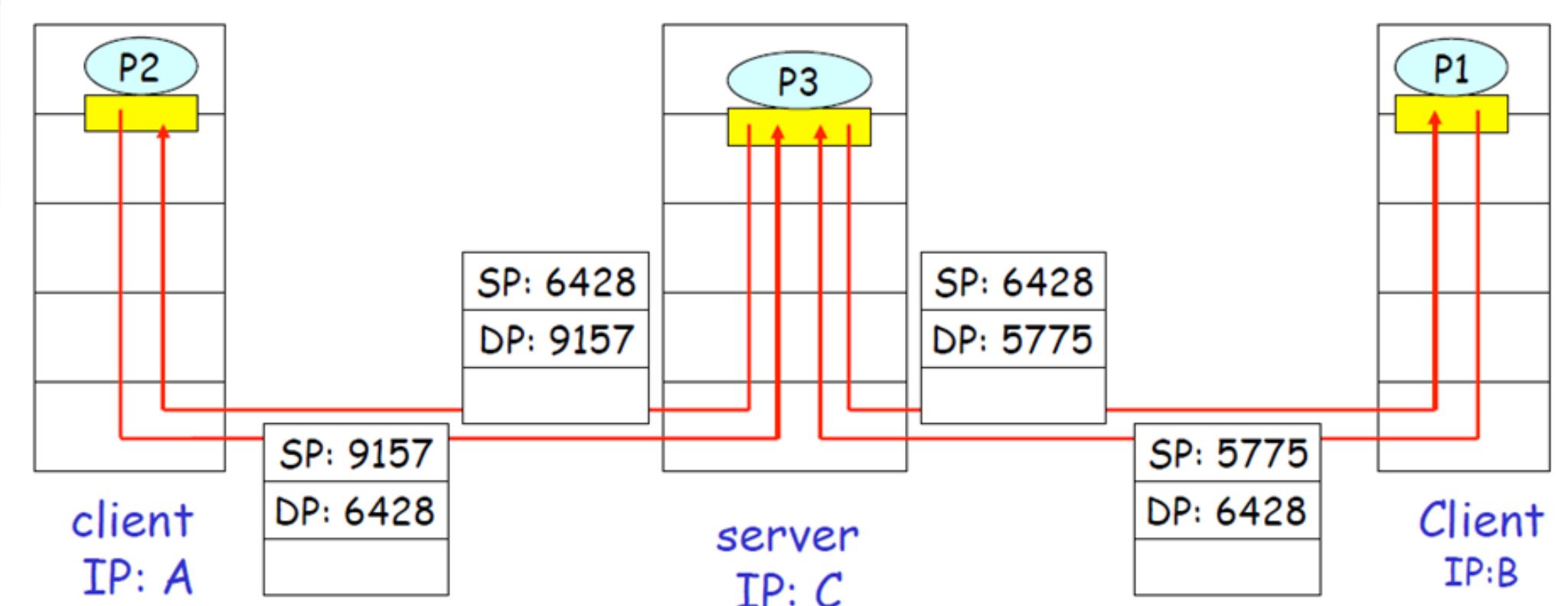
Puerto	Nombre
161	SNMP
162	SNMPTrap
377	NEC corporation
385	IBM application
530	RPC
531	chat
749	kerberos admin.

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

# Demultiplexación sin conexión

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

# Demultiplexación sin conexión





## Formato de un mensaje UDP

0		1		2		3															
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Source Port				Destination Port																	
UDP Len				Checksum																	
Data																					

# UDP



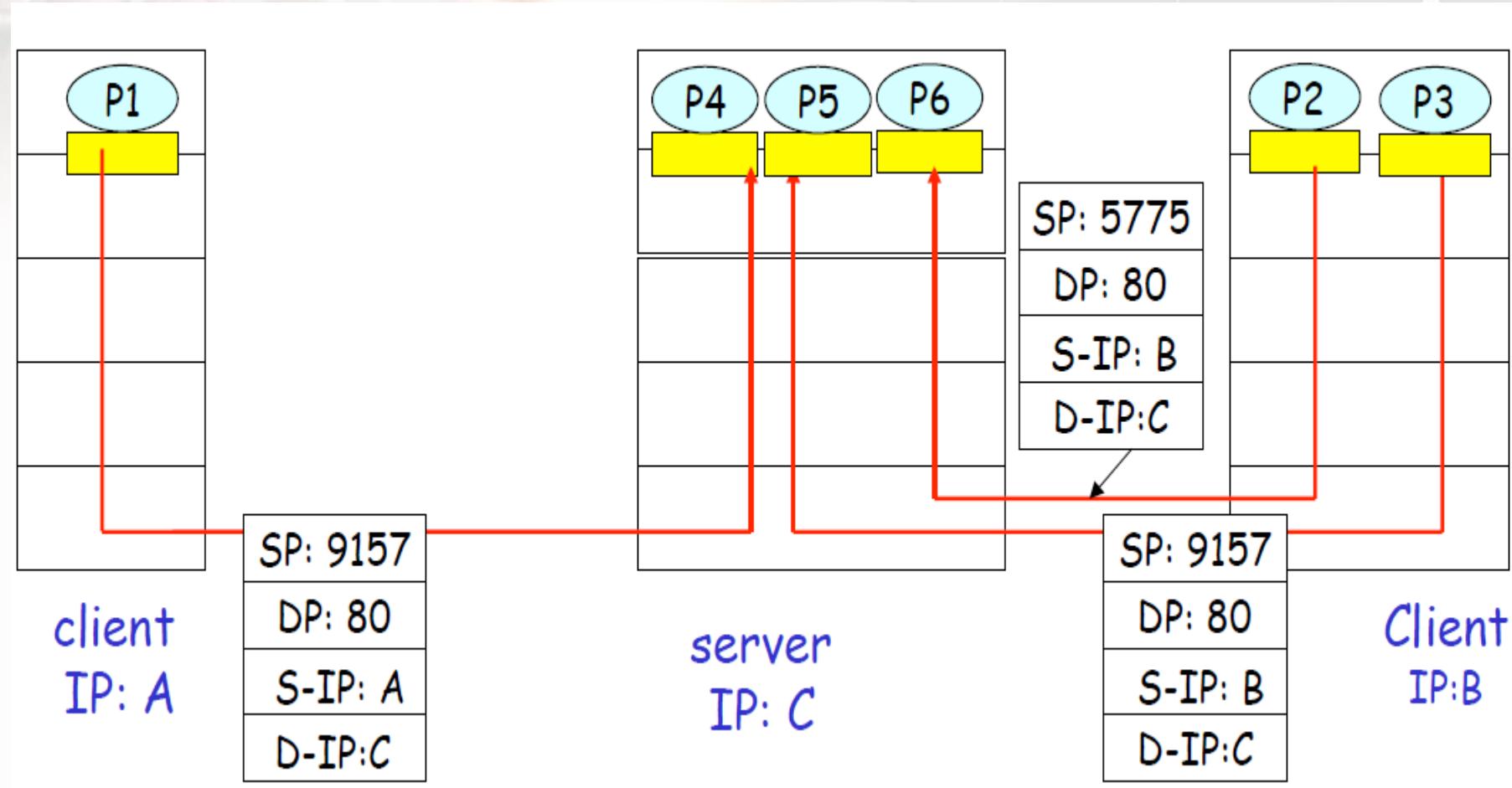
**UDP** transporta datos de manera no confiable entre hosts.

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

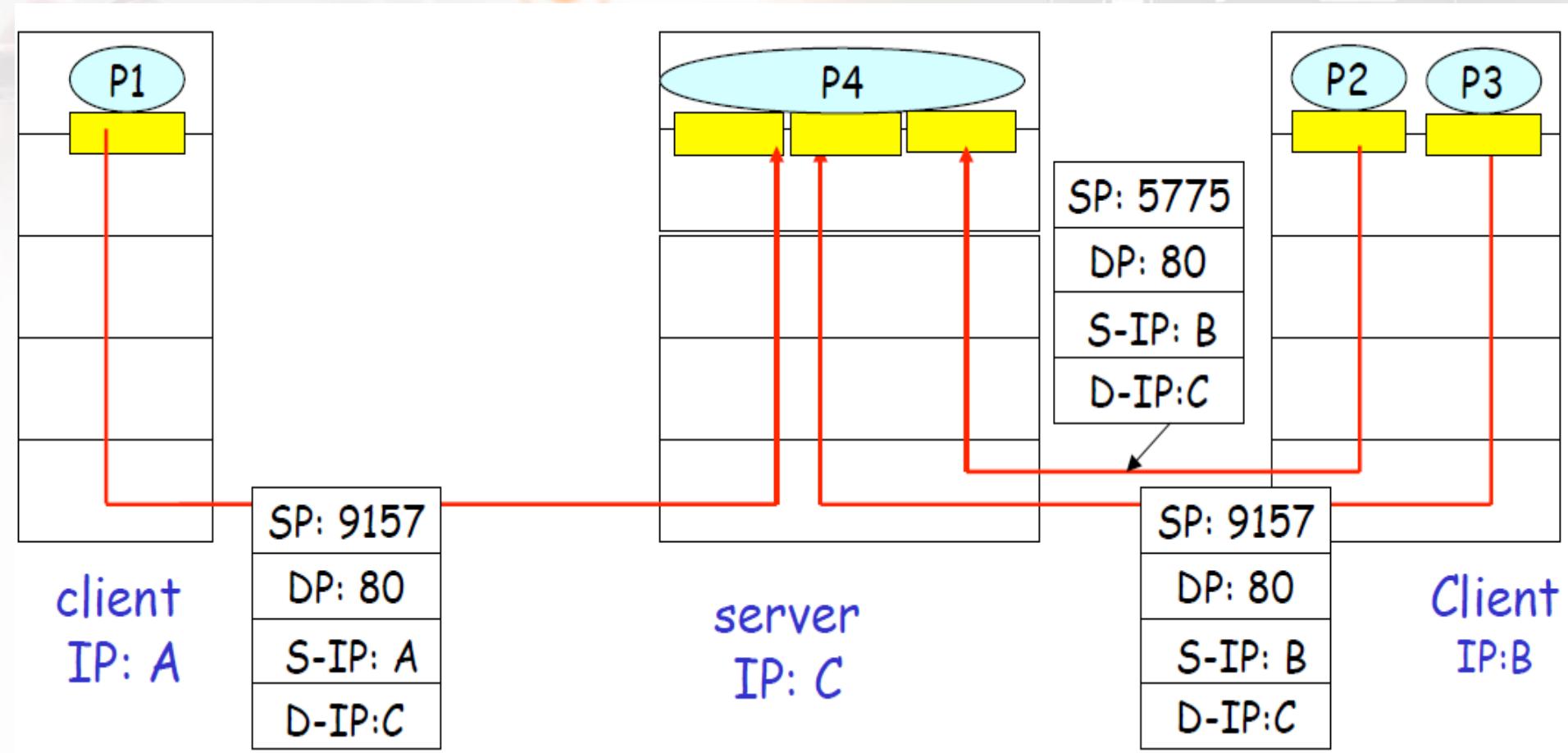
# Demultiplexión orientado a conexión

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

# Demultiplexión TCP



# Demultiplexión TCP: multithreading



# Transferencia de datos confiable

- ◆ Si el protocolo de red es confiable
    - ◆ No se alteran bits
    - ◆ No se pierden paquetes
    - ◆ Los paquetes llegan en orden
- ⇒ La lógica a implementar es muy simple

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

Sender

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

Receiver

# Canal con posibles errores

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

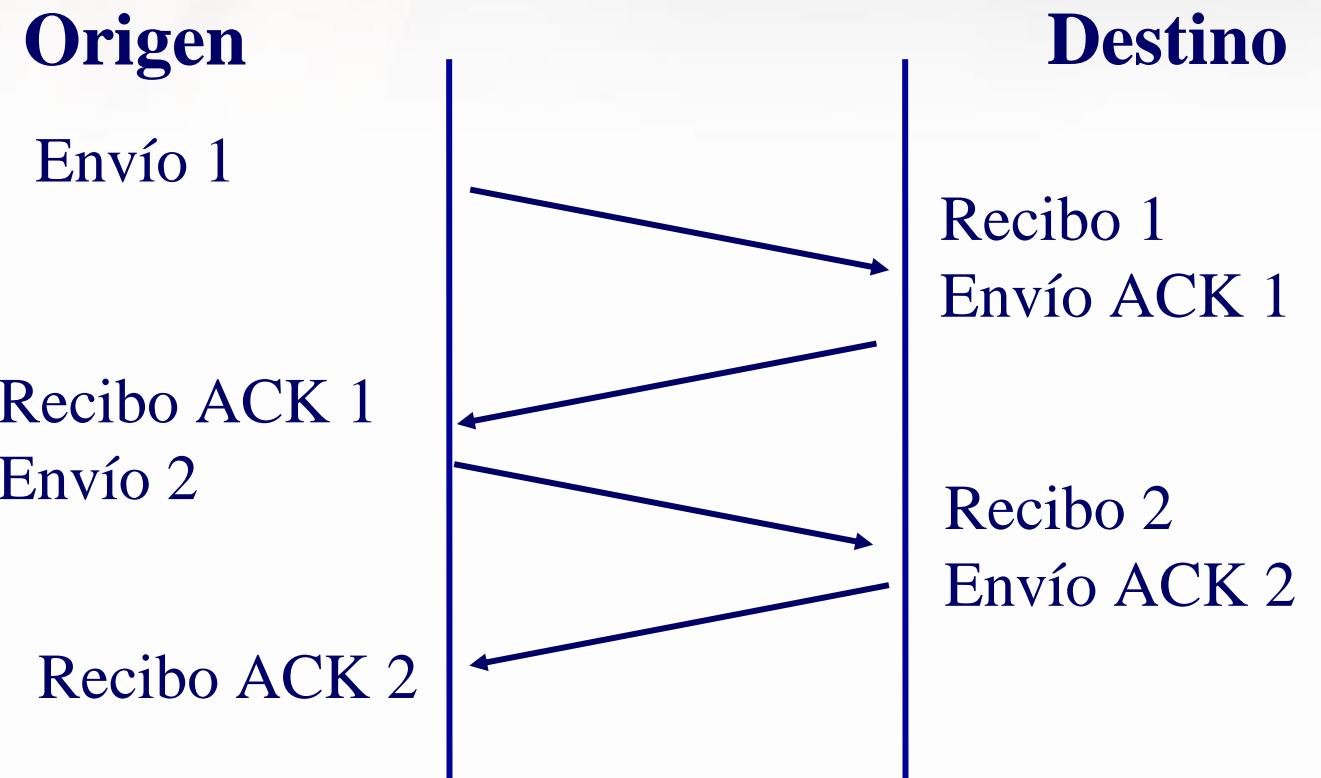
¿Qué sucede si se corrompe ACK/NAK?

¿Y si se envían paquetes duplicados?

Números de secuencia en cada paquete

# Control trivial: stop & wait

Acuse de recibo



# Control trivial: stop & wait

Acuse de recibo

**Origen**

Envío 1

Recibo NAK 1

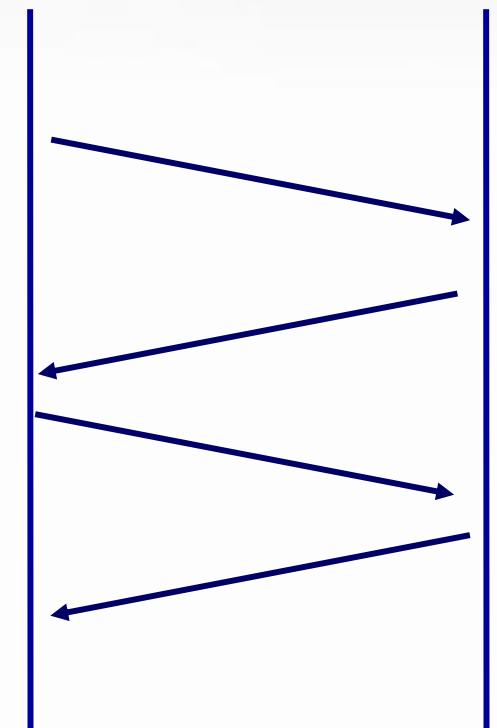
Envío 1

Recibo ACK 1

**Destino**

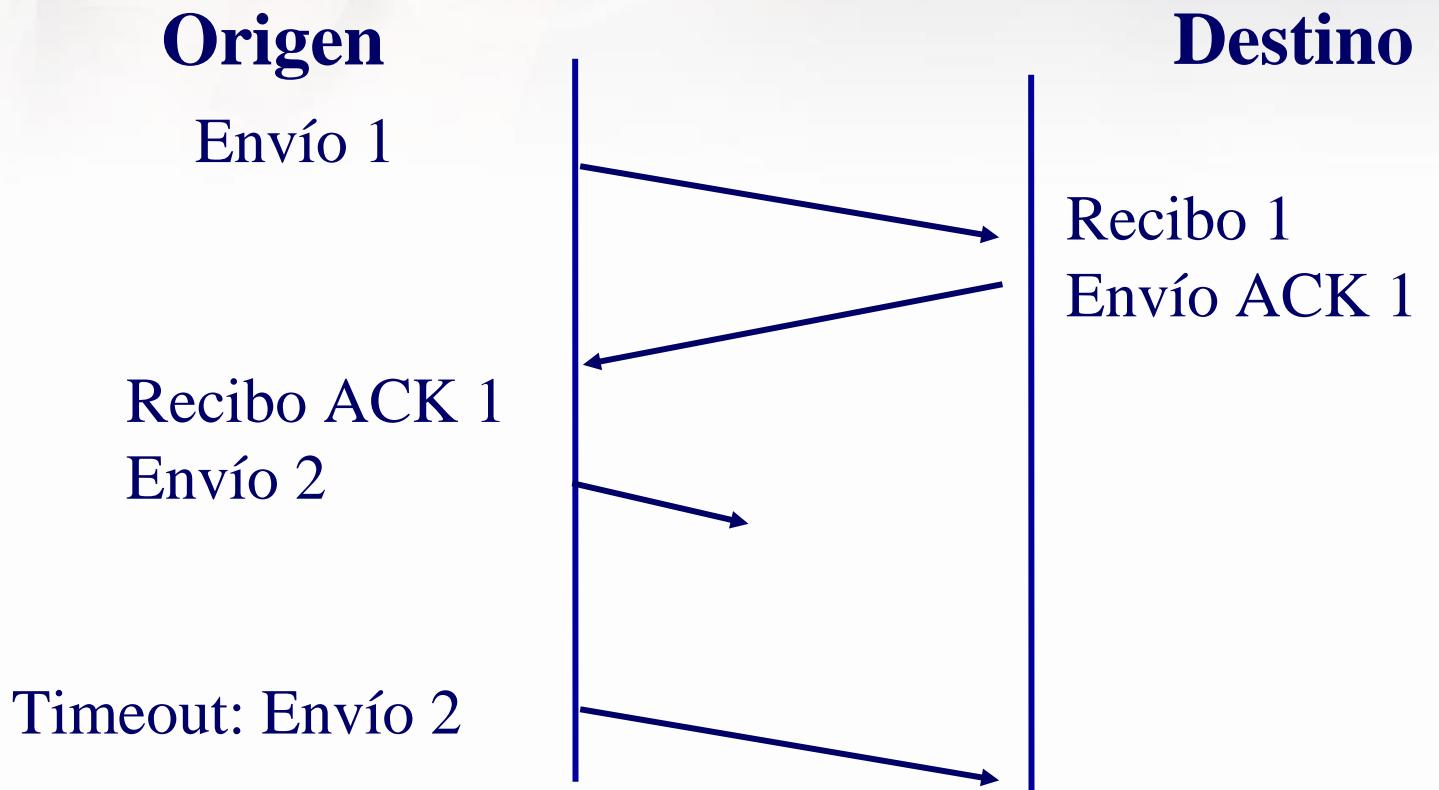
Recibo 1 con errores  
Envío NAK 1

Recibo 1  
Envío ACK 1



# Control trivial: stop & wait

## Acuse de recibo



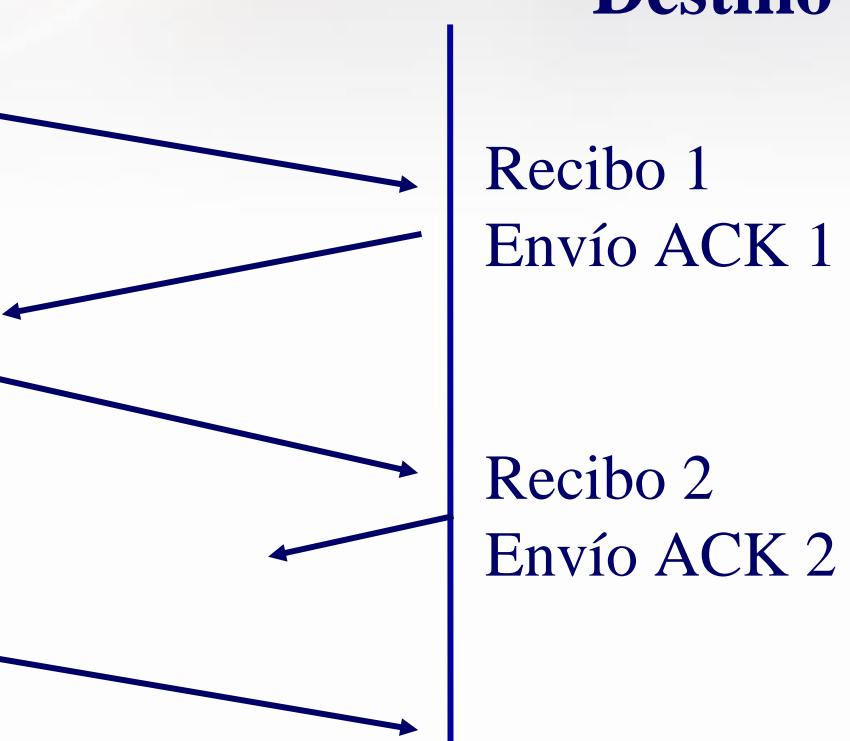
# Control trivial: stop & wait

Acuse de recibo  
**Origen**

Envío 1

Recibo ACK 1  
Envío 2

Timeout: Envío 2



¿Performance?

# Pipelining

Acuse de recibo

**Origen**

Envío 1

Envío 2

Envío 3

Recibo ACK 2

Envío 4

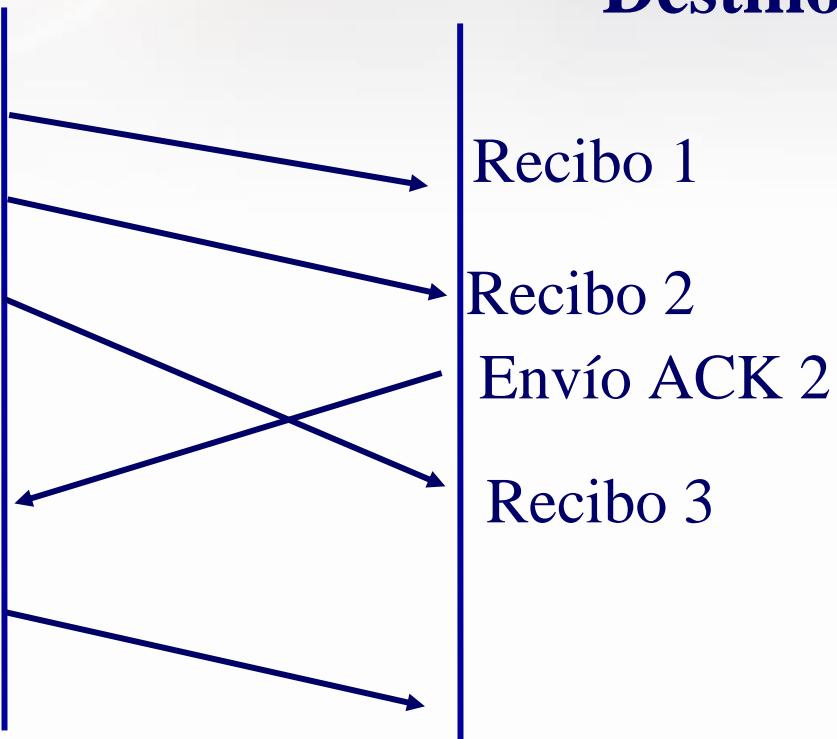
**Destino**

Recibo 1

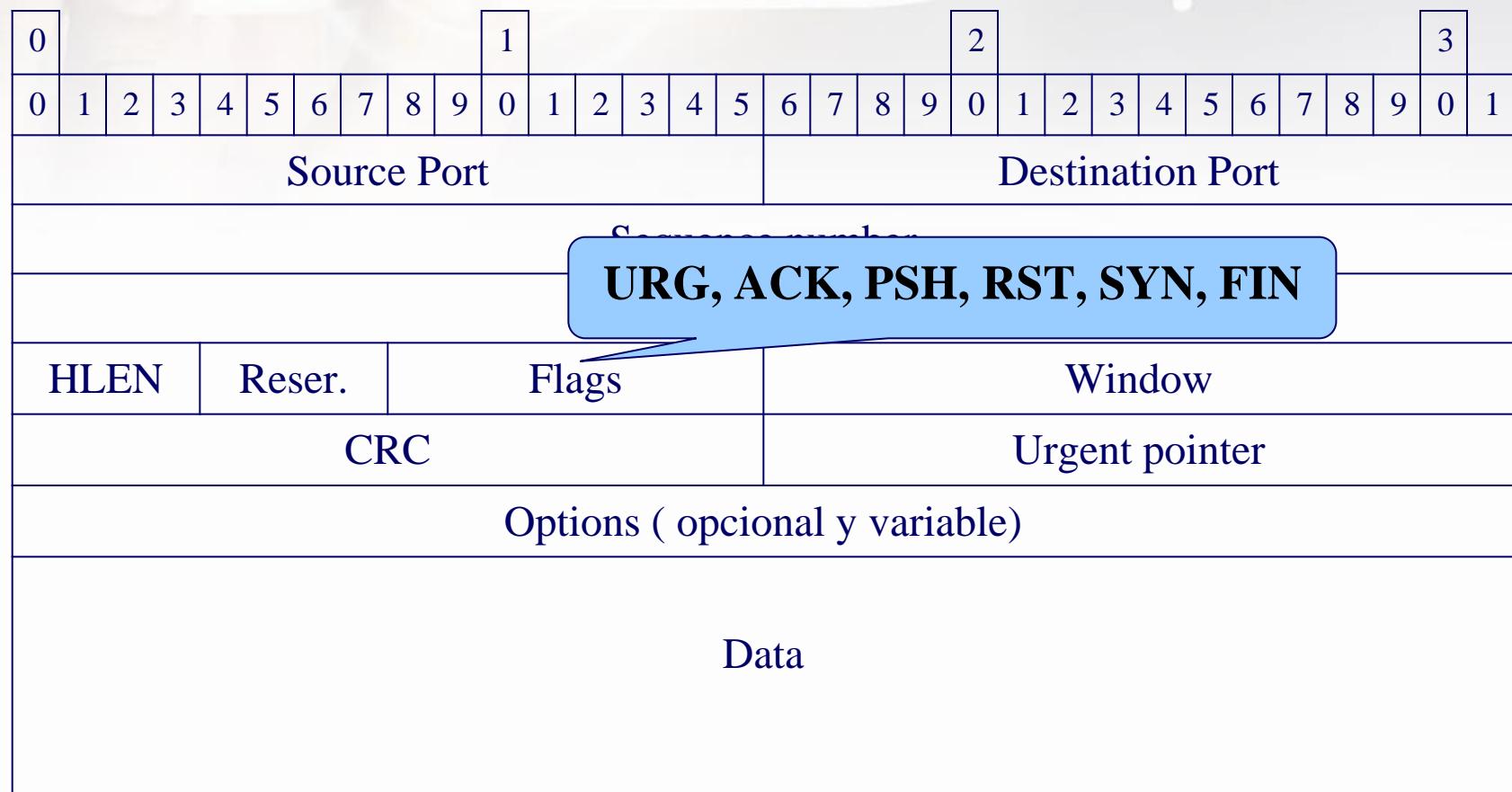
Recibo 2

Envío ACK 2

Recibo 3



## Formato de un mensaje TCP

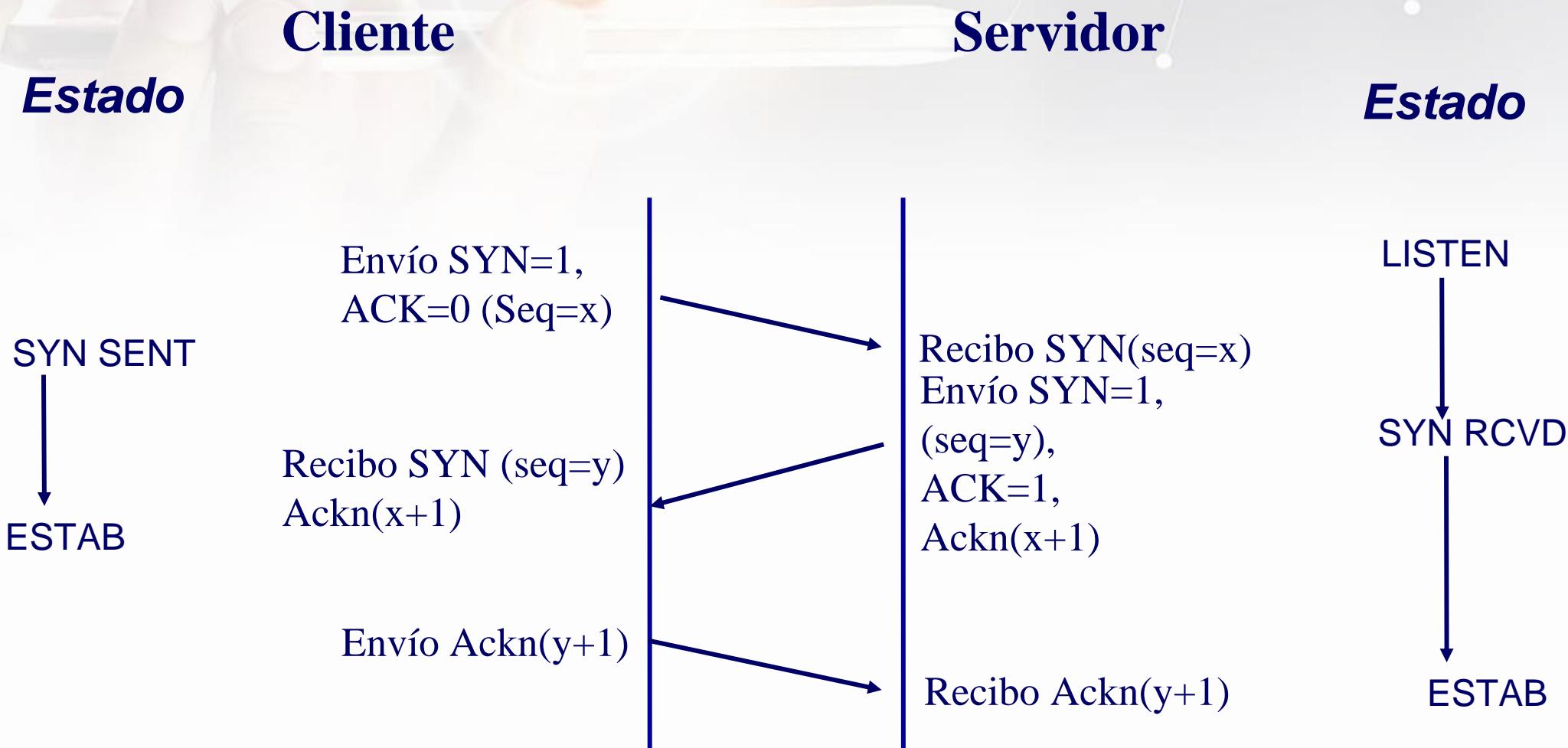




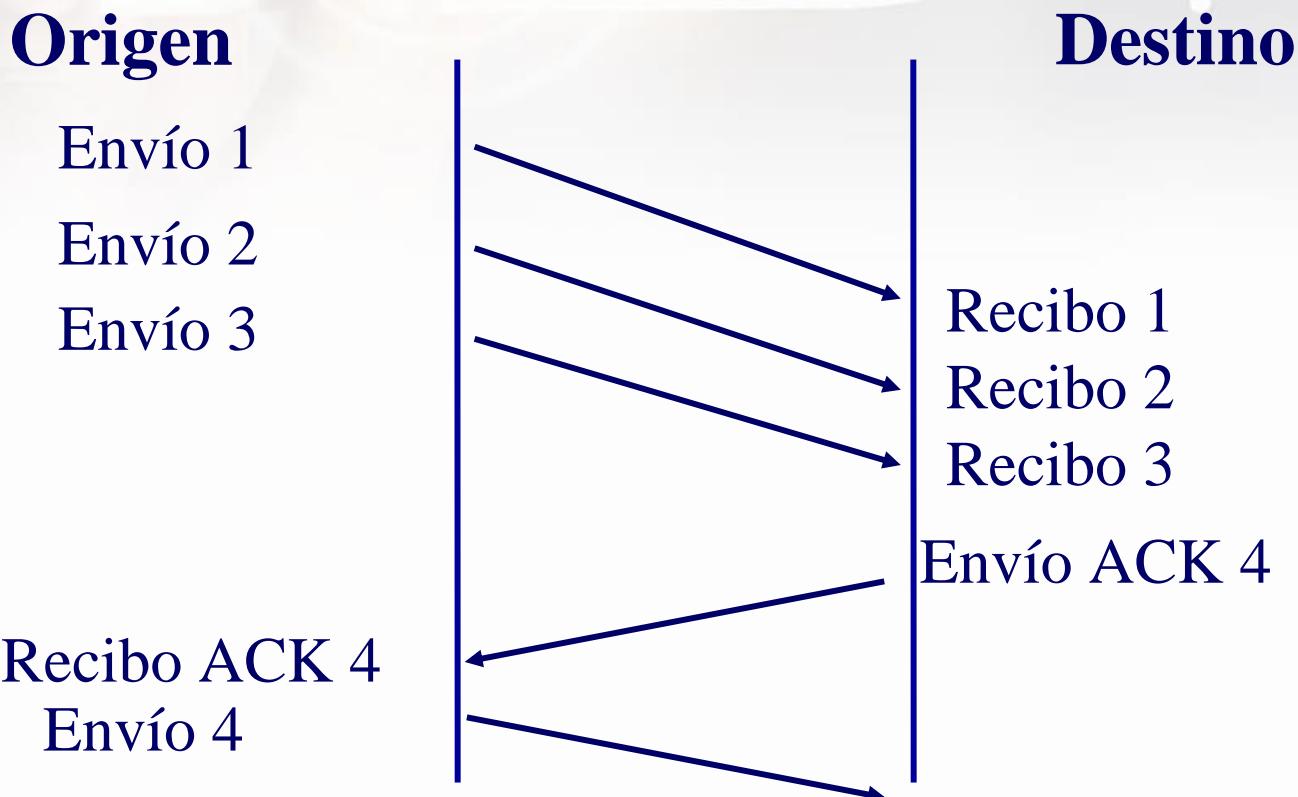
**TCP** ofrece un circuito virtual entre aplicaciones de usuario final.

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

# TCP: establecer conexión

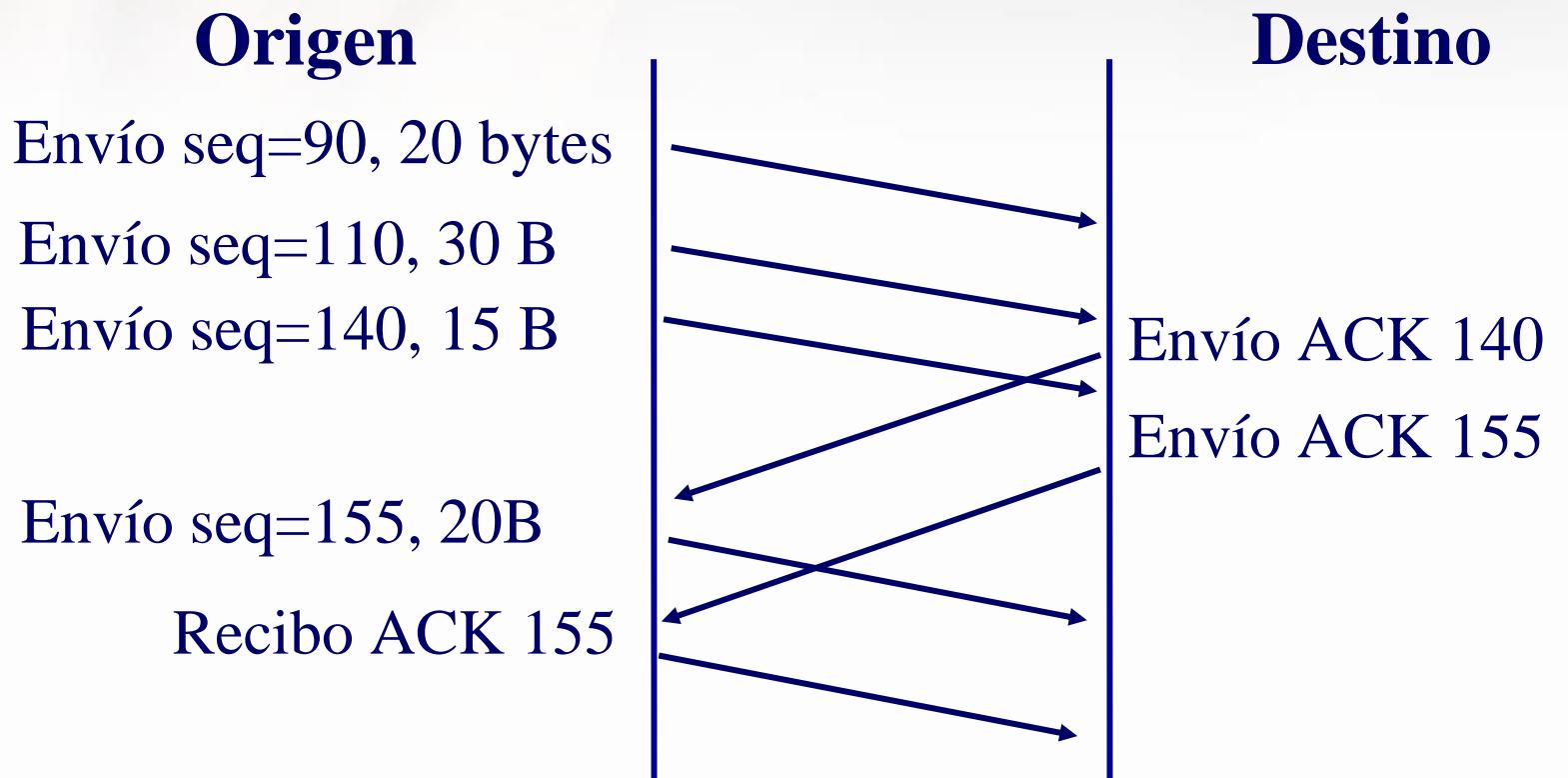


# TCP: acuse de recibo



# TCP: acuse de recibo

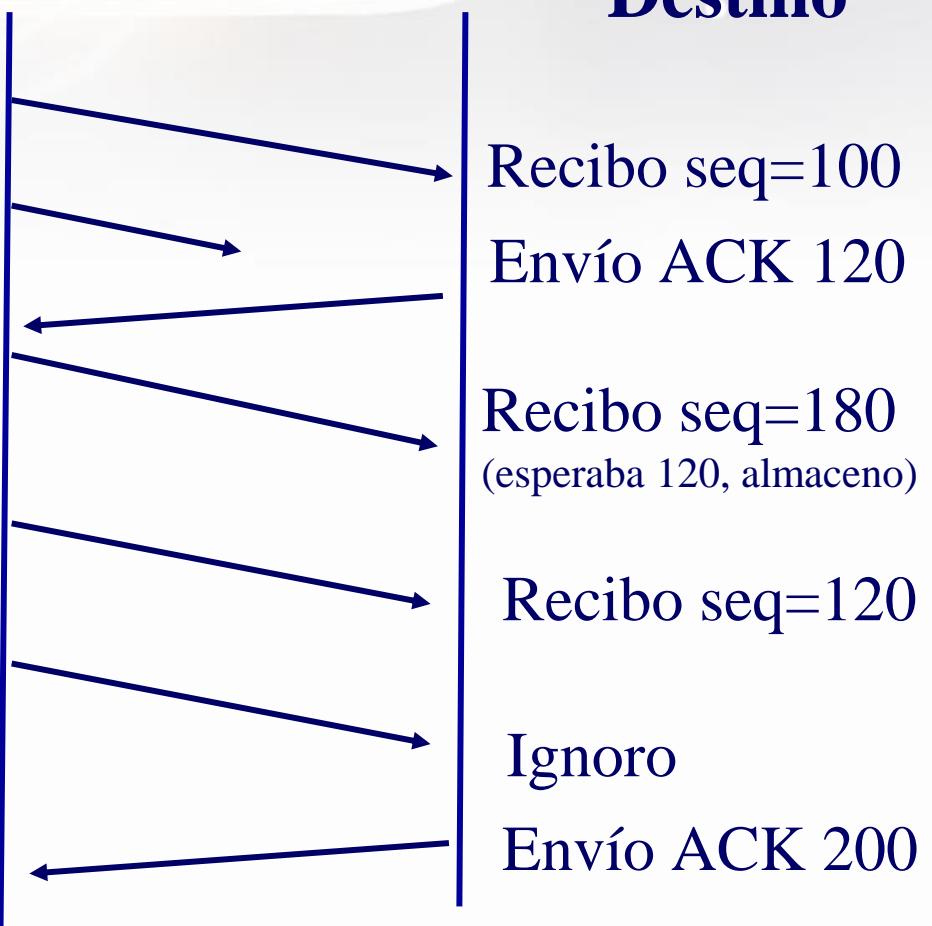
Como el canal es un «flujo de datos» no se envía ACK por segmento sino por el próximo byte a recibir



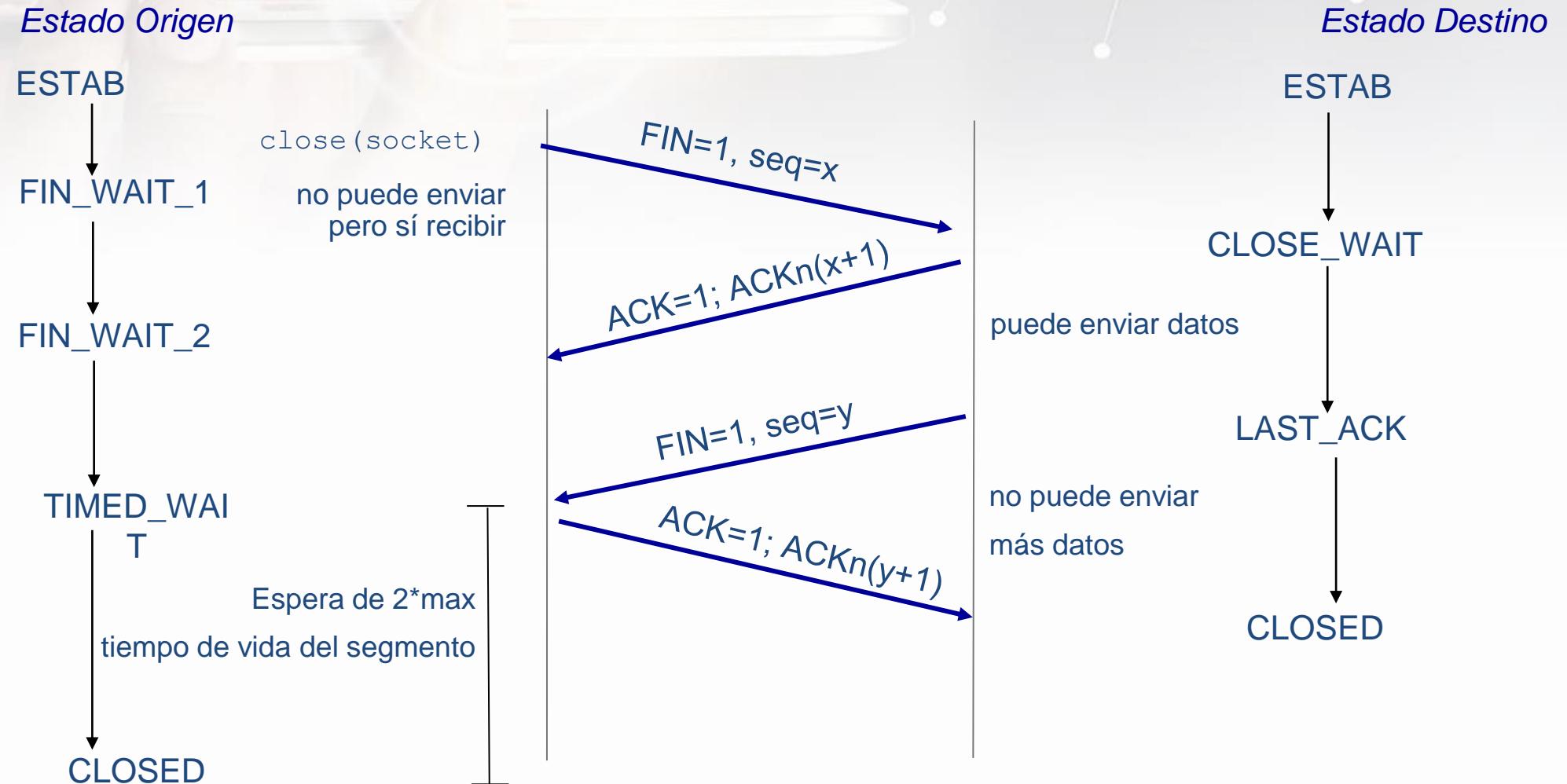
# TCP: pérdida de segmentos

**Origen**

Envío seq=100, 20B  
Envío seq=120, 60B  
  
Envío seq=180, 20B  
*timeout:*  
Envío seq=120, 60B  
  
Envío seq=180, 20B



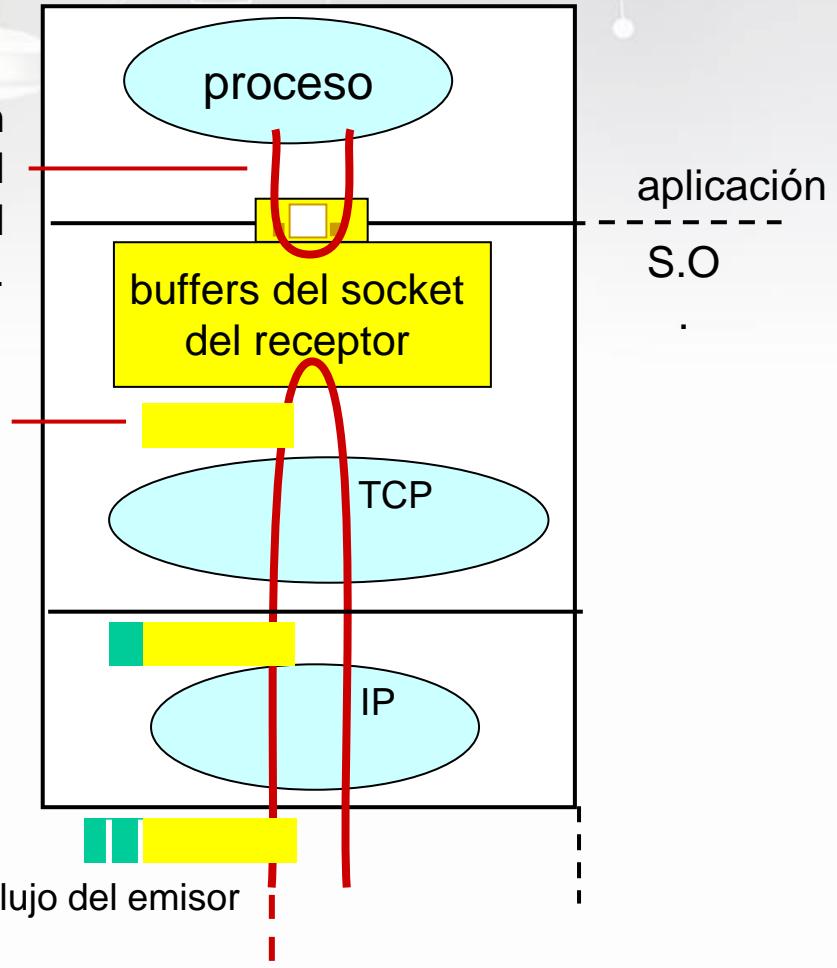
# TCP: cierre de conexión



# TCP: control de flujo

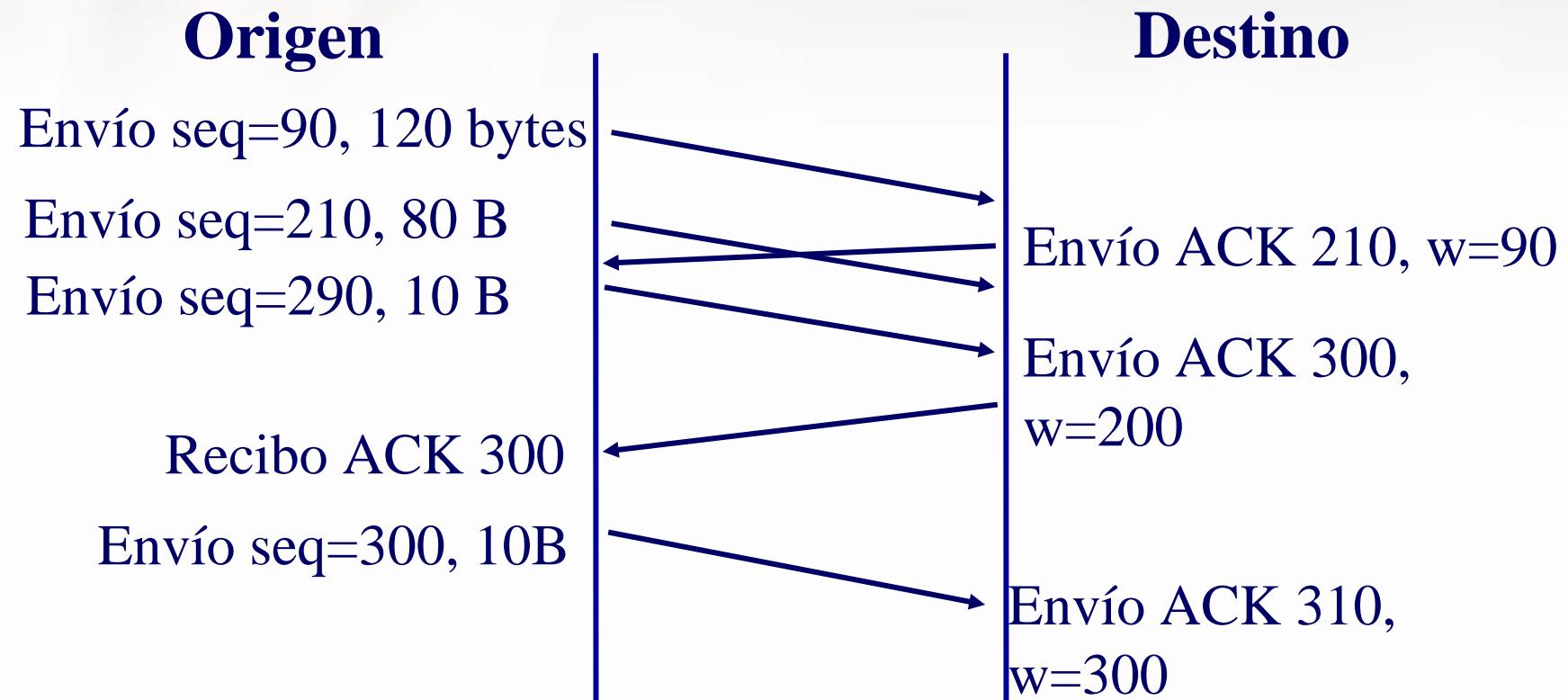
El receptor controla al emisor, de modo que el emisor no sobrecargue el buffer del receptor

la aplicación consume el buffer del receptor ....  
... más lento de lo que el emisor envía



# TCP: control de flujo

Junto con el ACK enviamos tamaño de “ventana”



# TCP: control de flujo

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

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

¿Es adecuado ese tamaño máximo hoy en día?

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

# Control de congestión

## Hace referencia al tráfico IP

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

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

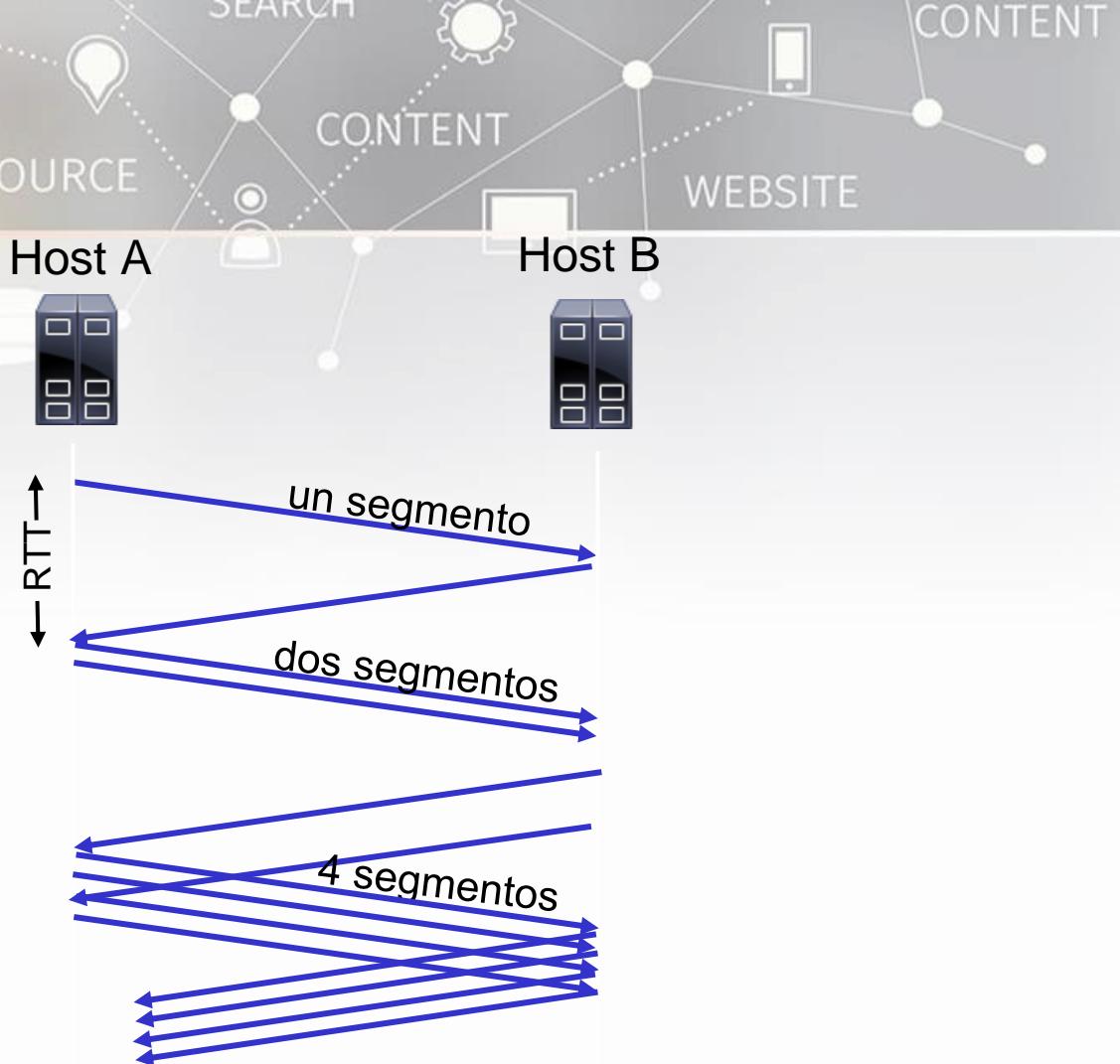
# Control de congestión en TCP

**AIMD:** *additive increase multiplicative decrease*

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

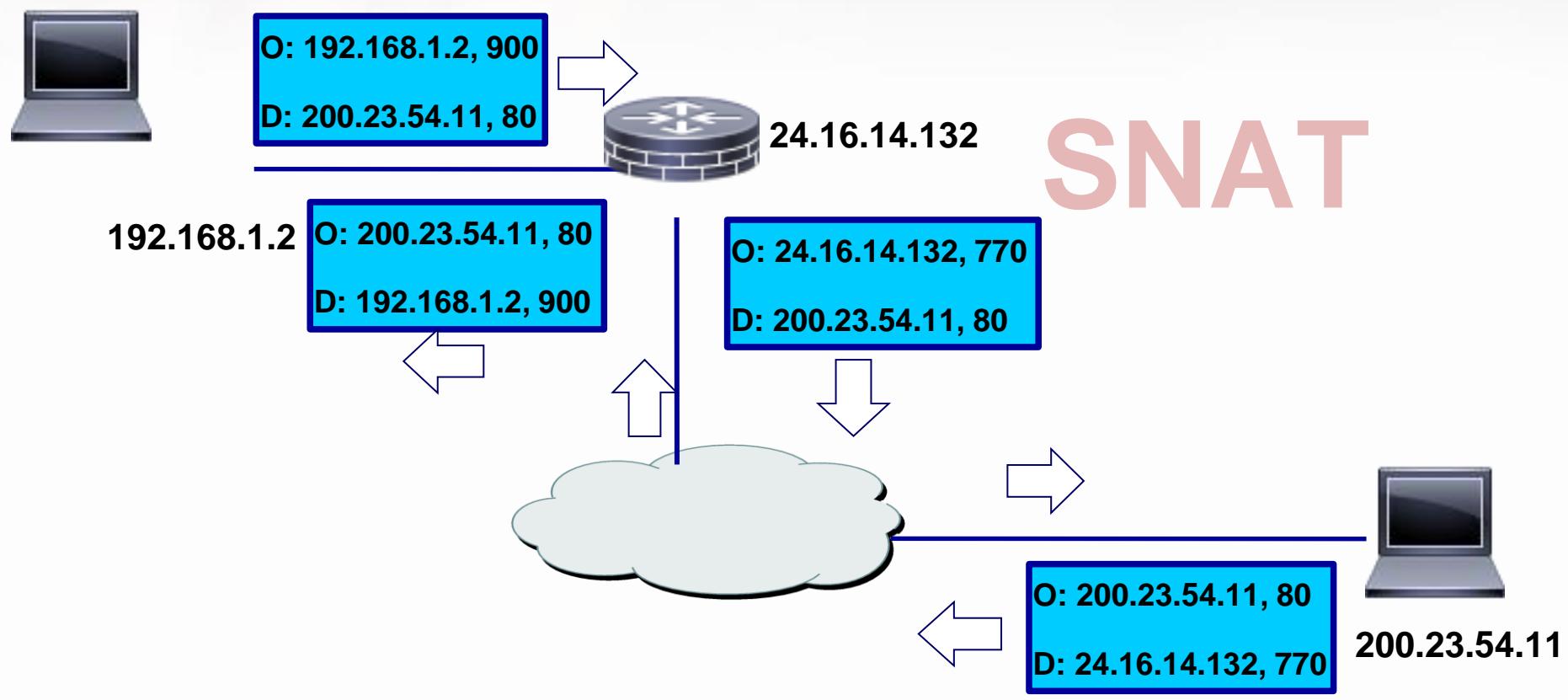
# TCP slow start

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



# NAPT (aka NAT)

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

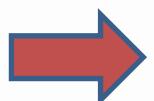


# NAT / NAPT



Tabla que mantiene el firewall para NAPT

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

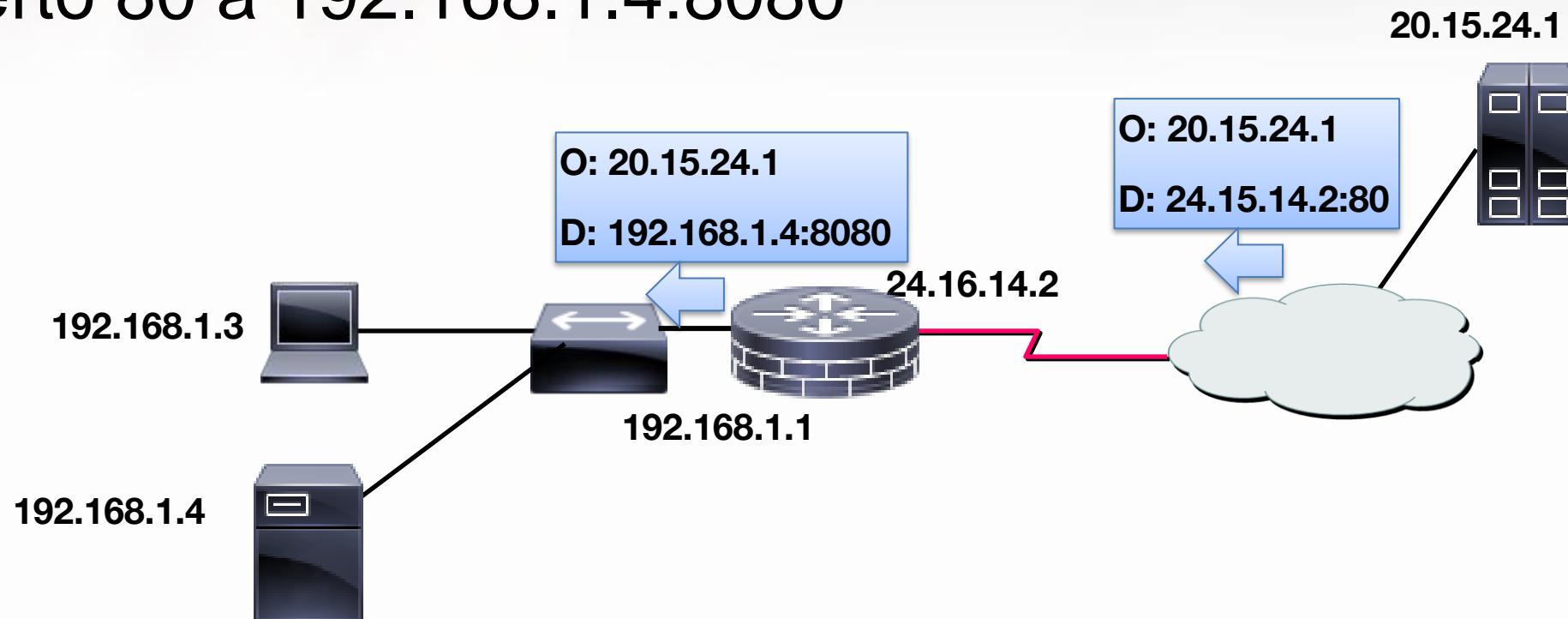


**DNS Caching-only Server**

**+ Firewall**

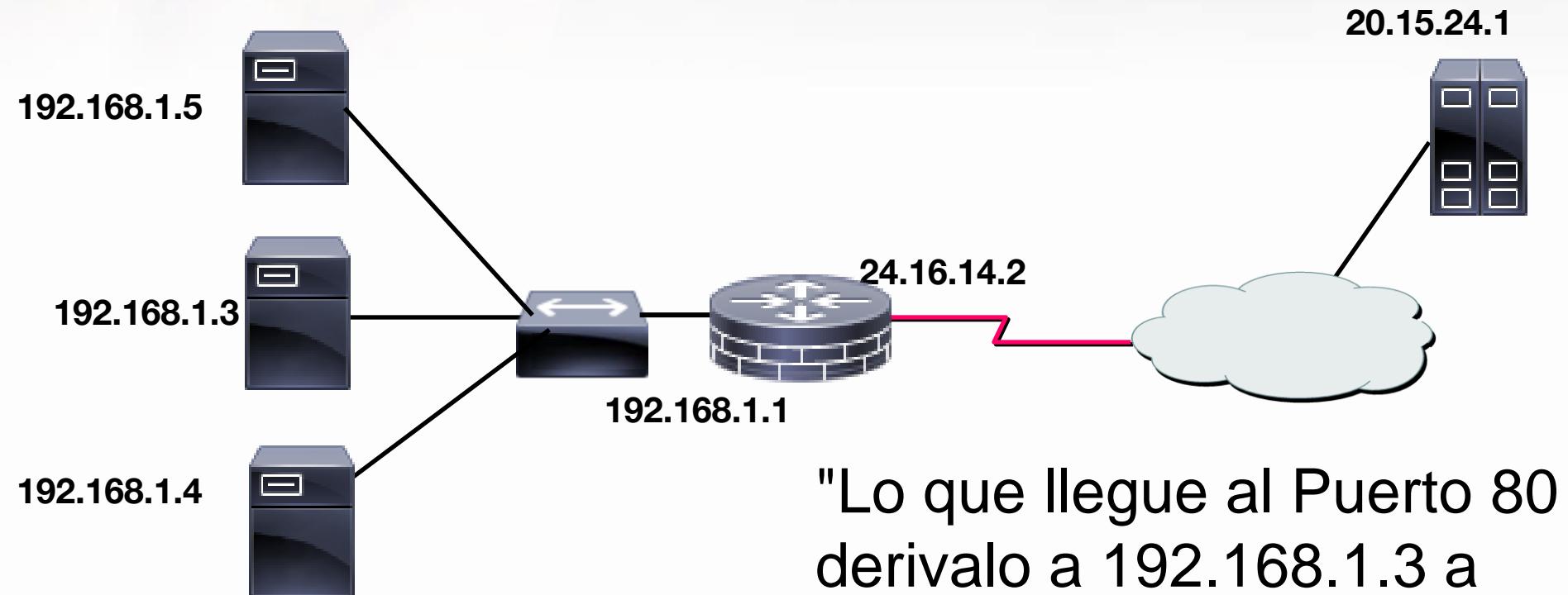
# DNAT (*port forwarding*)

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



# DNAT (*port forwarding*)

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

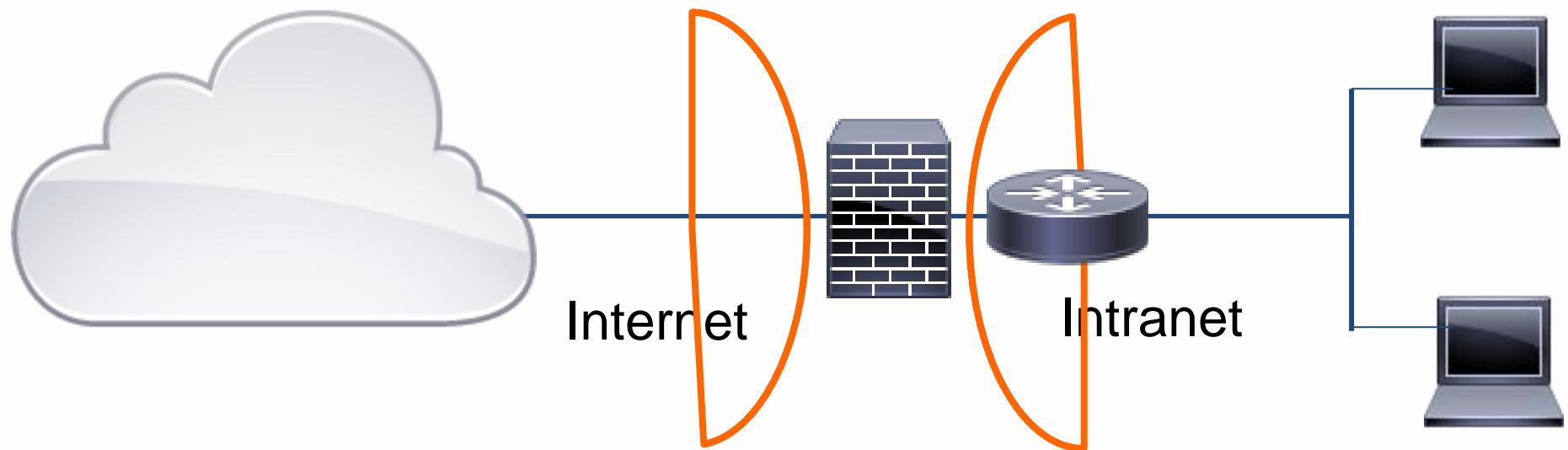


# Firewalls



Un firewall opera en el nivel más bajo de red:

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

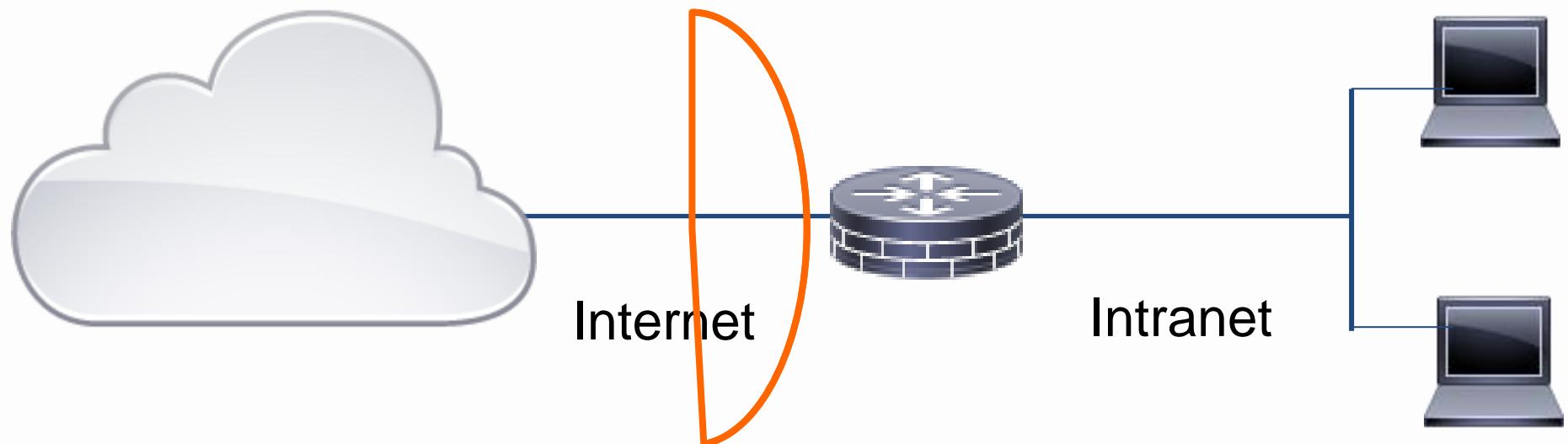


# Firewalls



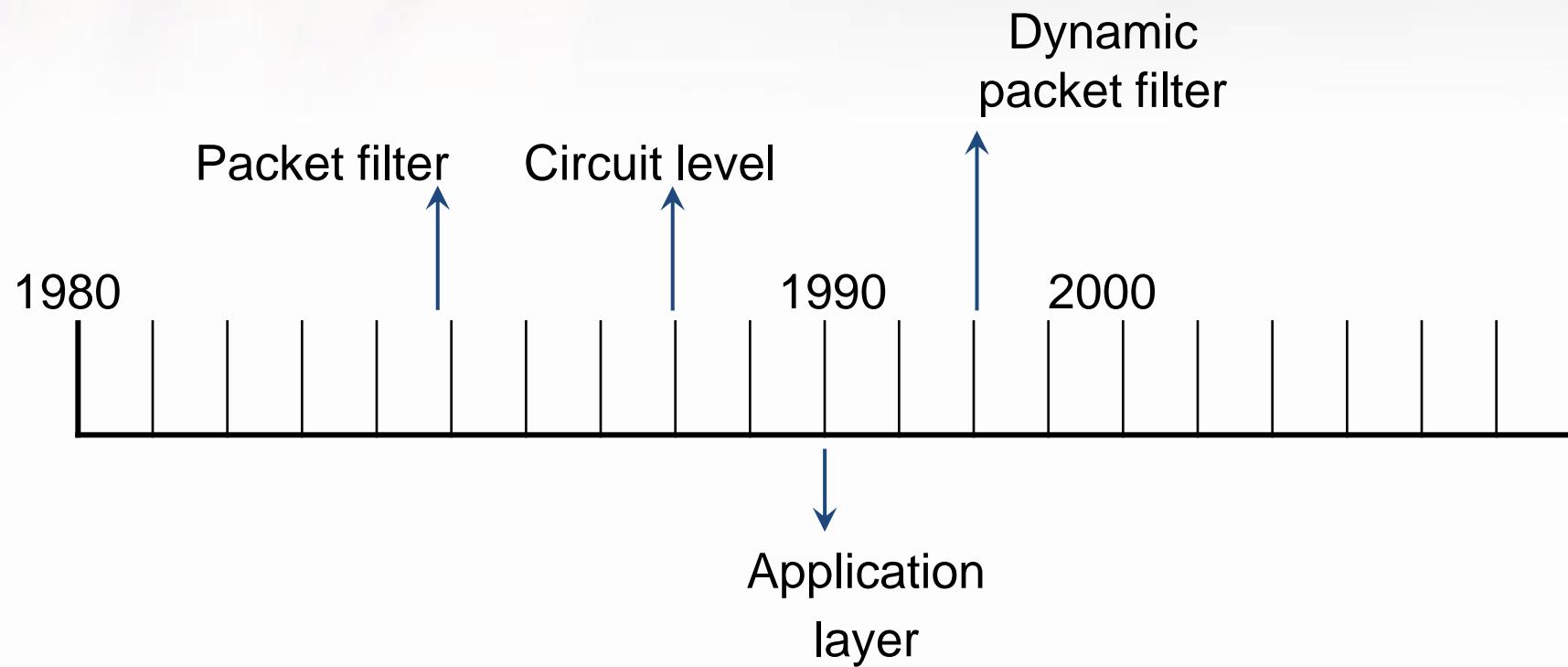
Un firewall opera en el nivel más bajo de red:

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



# Firewalls

## Evolución de la arquitectura de firewalls

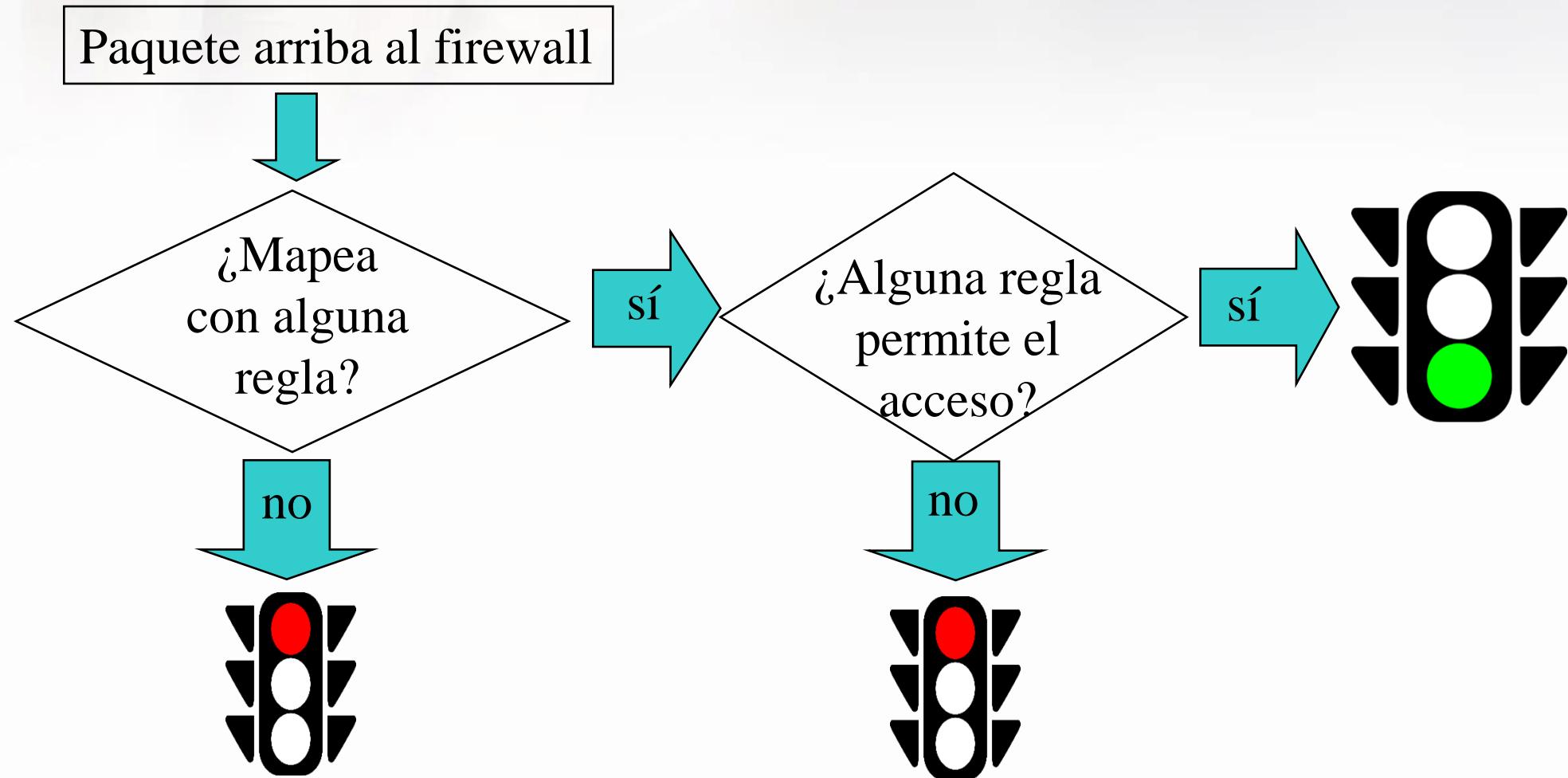


# Firewalls: *packet filter*

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

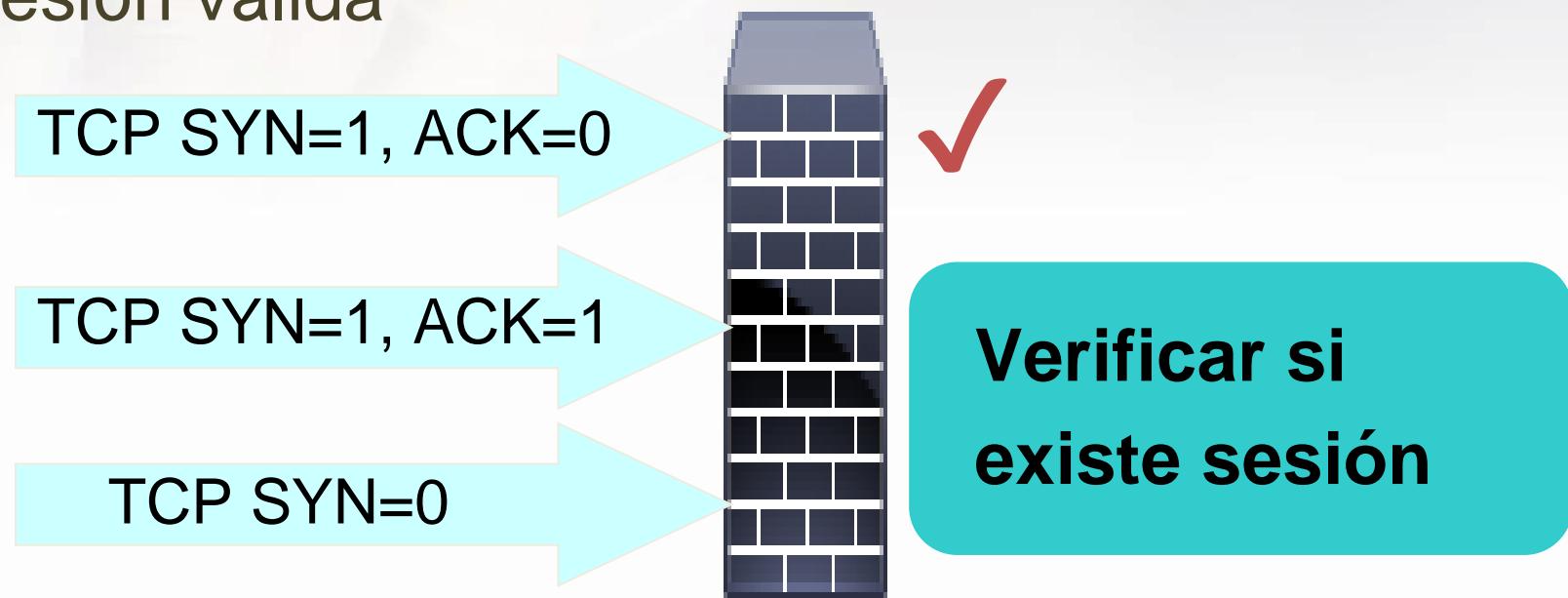
# Firewalls: *packet filter*

Algoritmo de inspección de paquetes



# Firewalls: *circuit level*

Pueden validar si un paquete pertenece a una sesión válida



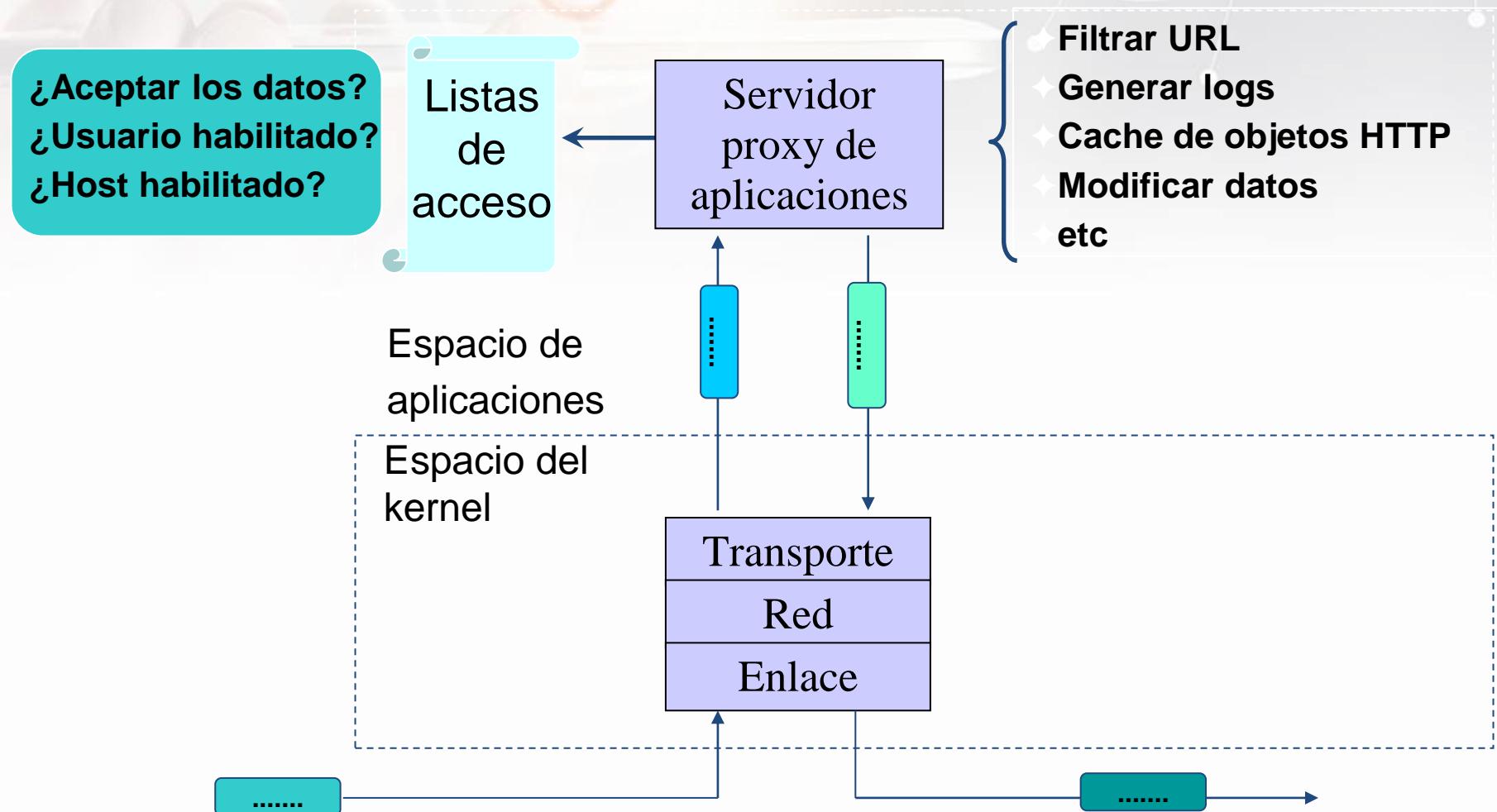
# Firewalls: *Circuit level*

Una vez establecida una sesión se suele almacenar:

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

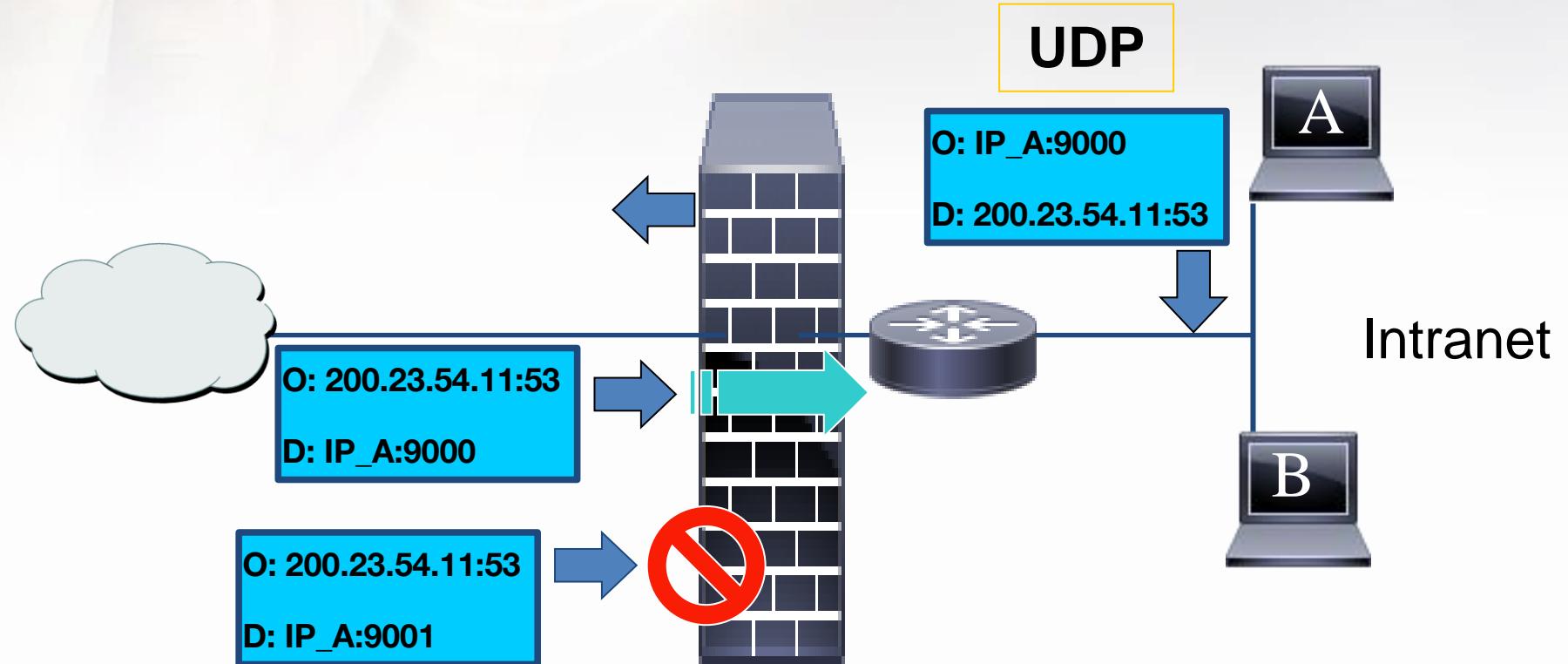
¿UDP?

# Firewalls: application layer



# Firewalls: *Dynamic packet filter*

Permiten la modificación de reglas “*on the fly*”



# Material de lectura

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



# Protocolos de transporte

- ◆ SCTP

# SCTP (rfc 2960, 4960)

Diseñado originalmente para transportar telefonía sobre IP

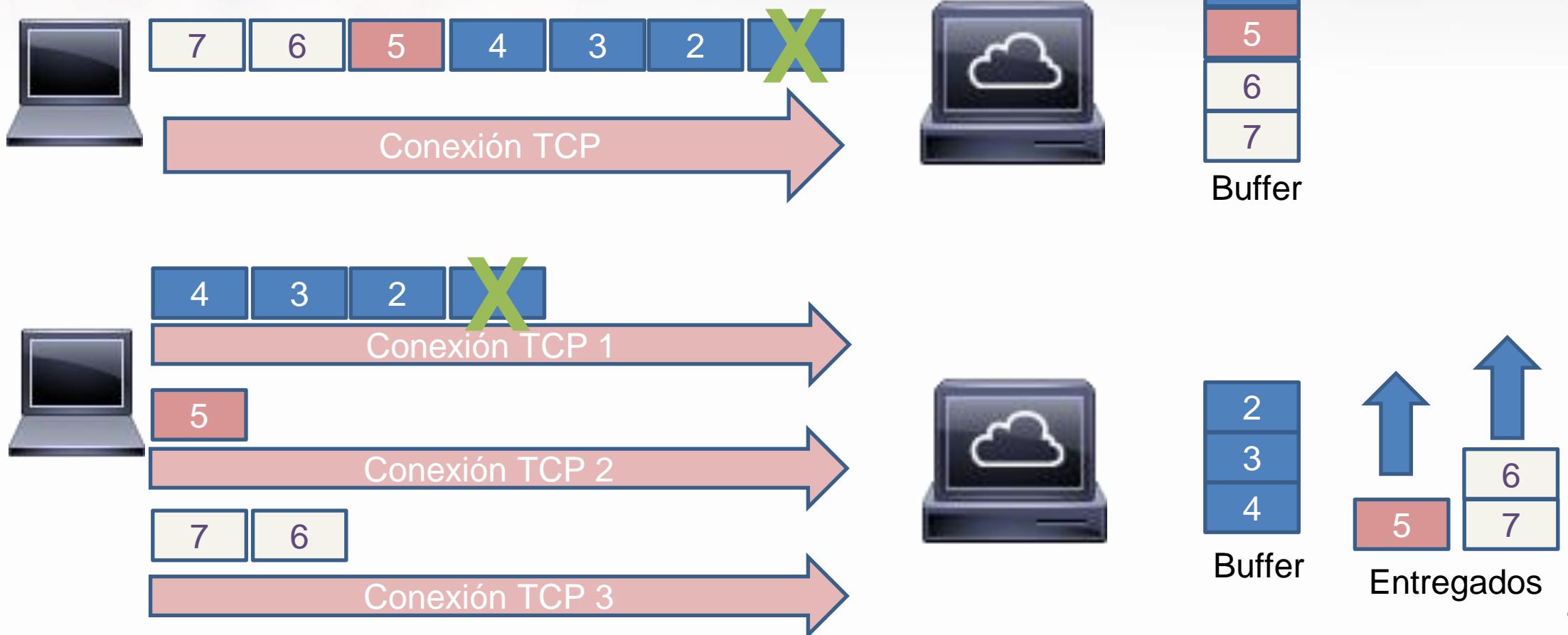
- ◆ Confiabilidad
- ◆ Control de flujo
- ◆ Secuenciación
- ◆ Orientado a mensaje
- ◆ Permite aceptar mensajes fuera de orden
- ◆ *Multihoming*: origen y/o destino con más de una IP
- ◆ Permite definir flujos paralelos: solucionar *Head Of the Line blocking*.

# SCTP: multihoming



# SCTP: multiStreaming

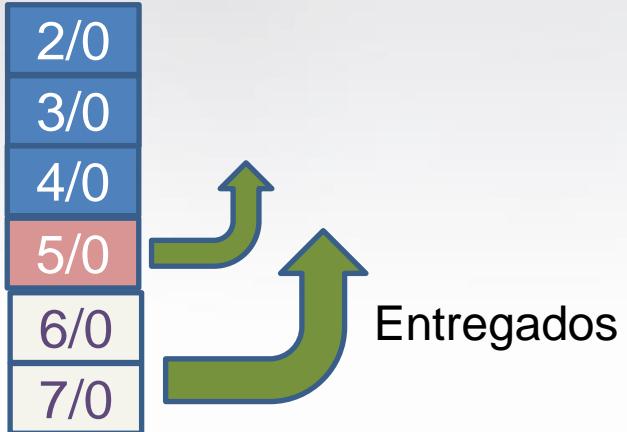
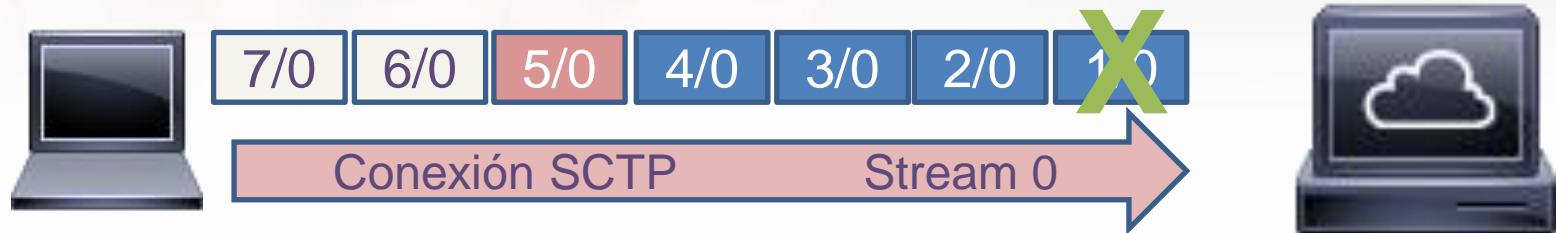
Resuelve el “*Head-of-line blocking*”



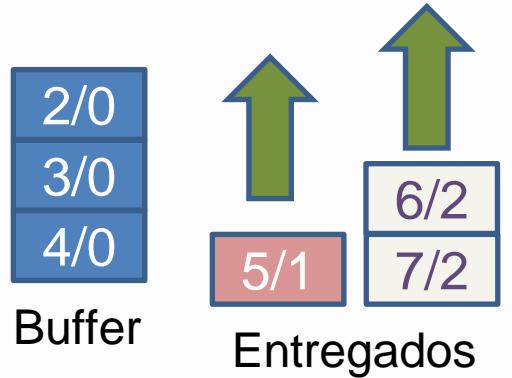
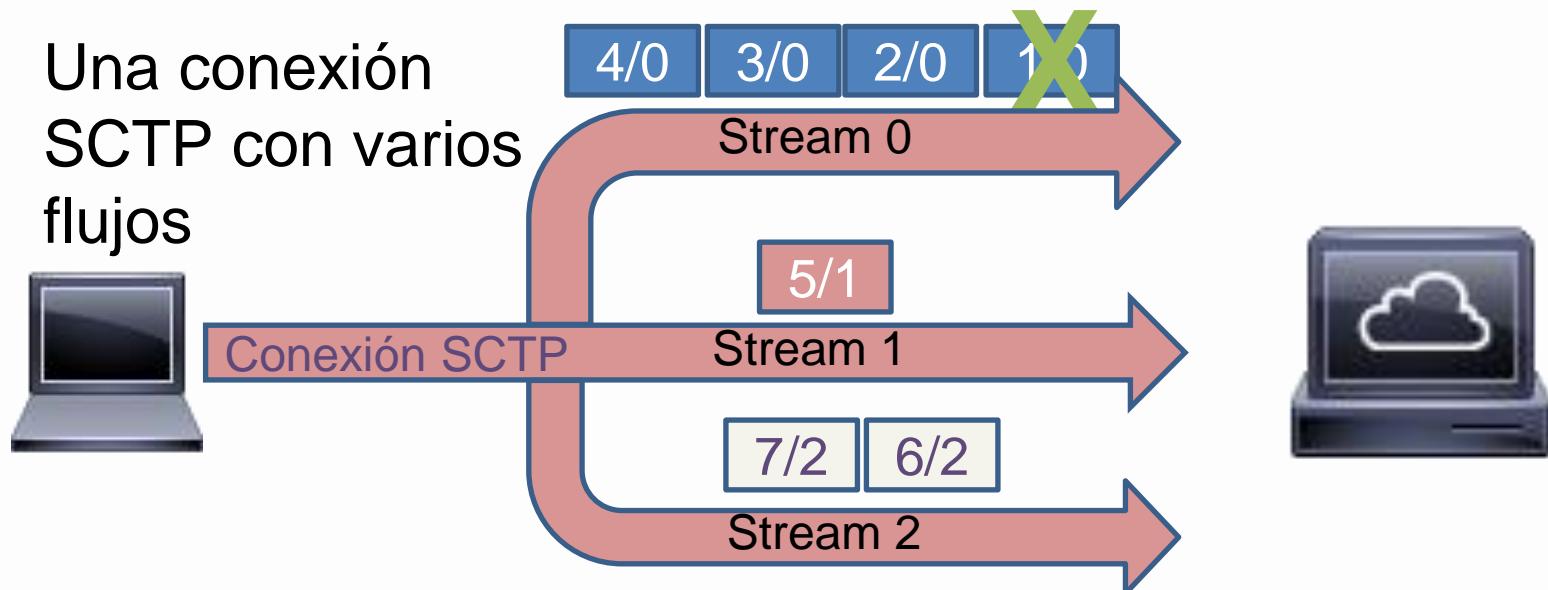
# SCTP: multiStreaming

Resuelve el “*Head-of-line blocking*”

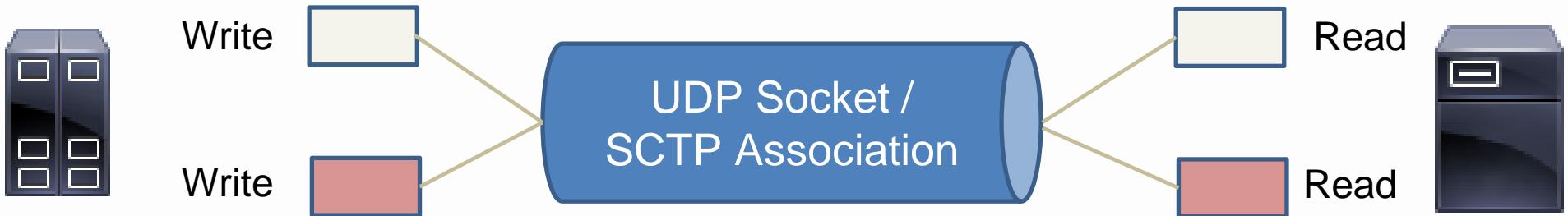
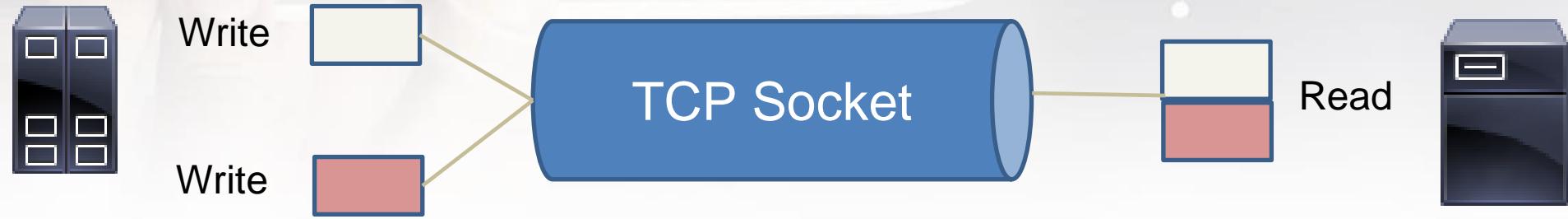
Una conexión SCTP enviando mensajes en desorden



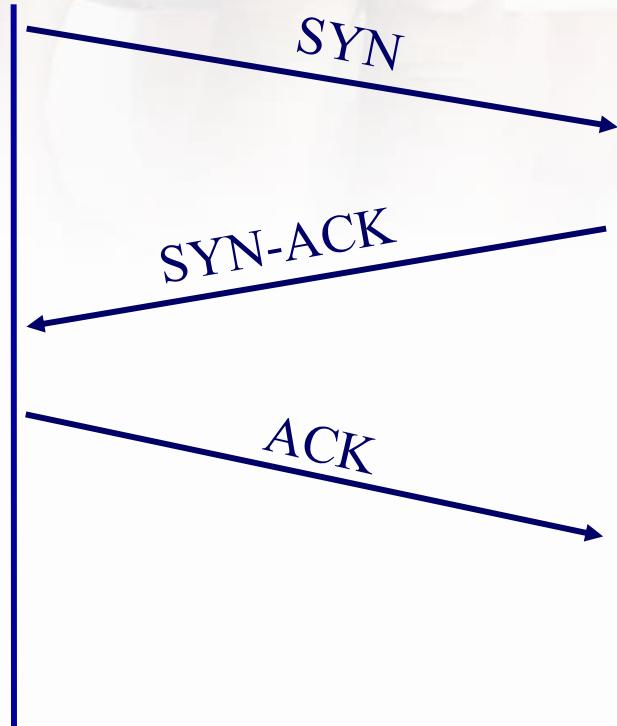
Una conexión  
SCTP con varios  
flujos



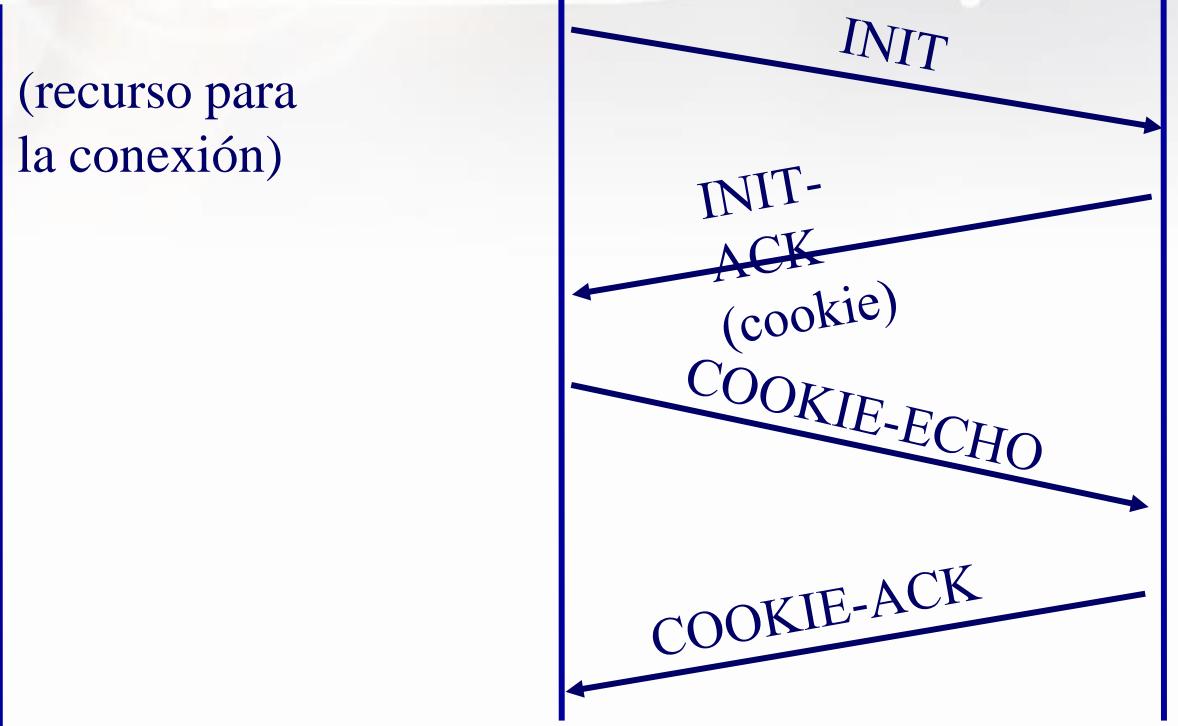
# SCTP: orientado a mensaje



# SCTP: inicio protegido



TCP 3-way handshake

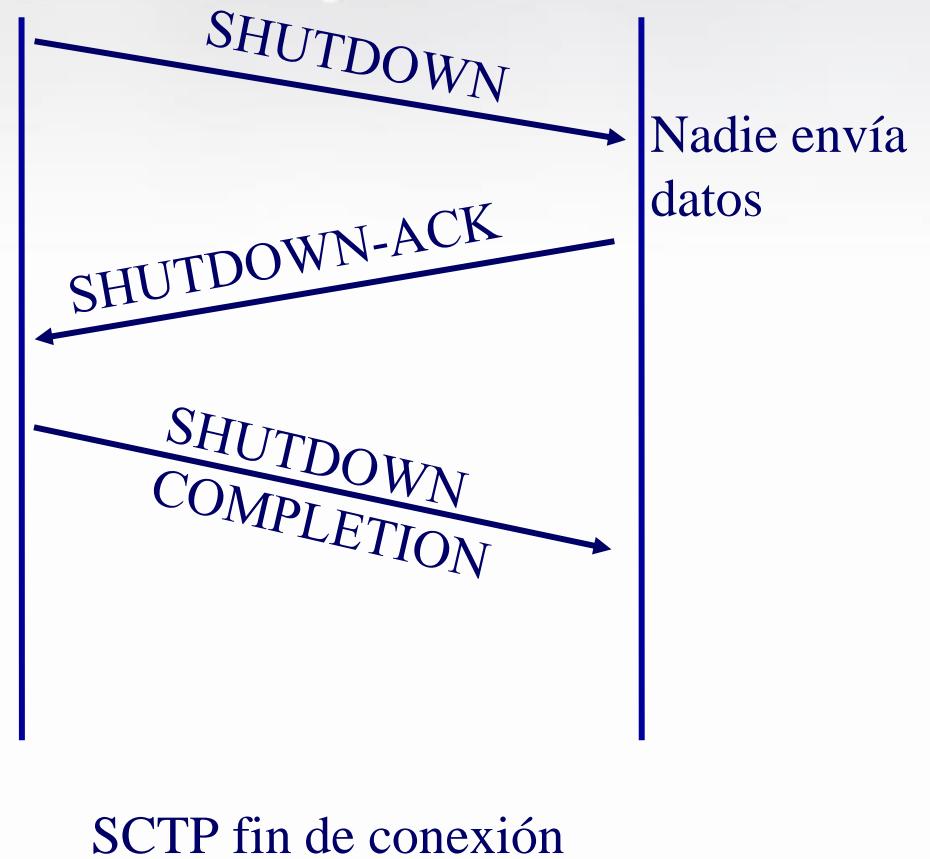
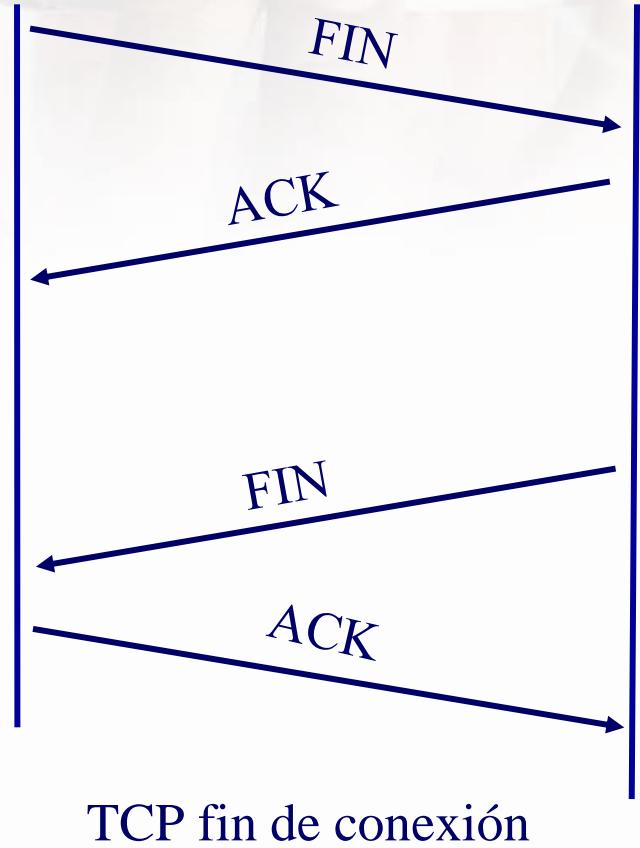


SCTP 4-way handshake

(envía cookie  
y no genera recursos  
para la conexión)

(recurso para  
la conexión)

# SCTP: cierre de conexión



# Comparación SCTP, TCP, UDP

Característica	SCTP	TCP	UDP
Estado almacenado en los hosts	SI	SI	NO
Transferencia de datos confiable	SI	SI	NO
Control de congestión	SI	SI	NO
Delimitación de límites de mensajes	SI	NO	SI
Fragmentación e integración de datos	SI	SI	NO
Multiplexación	SI	SI	NO
Soporte de multi-homing	SI	NO	NO
Soporte de multi-streaming	SI	NO	NO
Envío de datos desordenado	SI	NO	SI
Cookie de seguridad para evitar ataques de inundación de SYN	SI	NO	NO
Mensaje heartbeat	SI	NO	NO



## Capa de red

- ◆ Direcciónamiento
- ◆ Circuitos virtuales
- ◆ Comunicación de paquetes
- ◆ IPv4
- ◆ Subredes

# Capa de red



Tanto A como B deben poseer un identificador o dirección que permita enviar paquetes entre ellos.

Las direcciones de capa de red utilizan un esquema de direcccionamiento jerárquico que permite encontrar una ruta en forma eficiente.

# Capa de red

- ◆ Traslada segmentos de un host a otro
- ◆ En cada host se ejecutan los protocolos de nivel de red
- ◆ Cada host en el camino funciona como router: examina los encabezados IP y decide el camino a seguir

# Capa de red

- Posibles funciones en capa de red
  - Forwarding: mover paquetes de un “input” del router a un “output” del router
  - Routing: determinar la ruta que deben tomar los paquetes para llegar a destino
  - Establecer conexión (ATM, frame relay, X.25)
    - Antes de intercambiar datagramas, se establece un circuito virtual entre dos hosts

# Capa de red: servicios

- ¿Qué servicios adicionales puede ofrecer un protocolo de red?
  - Comutación de paquetes (datagramas)
    - Delivery garantizado
    - Delay mínimo garantizado
  - Flujo de datagramas (circuito virtual)
    - Entrega en orden
    - Bandwidth mínimo
  - Ambos
    - Integridad de datos, encriptación, ...

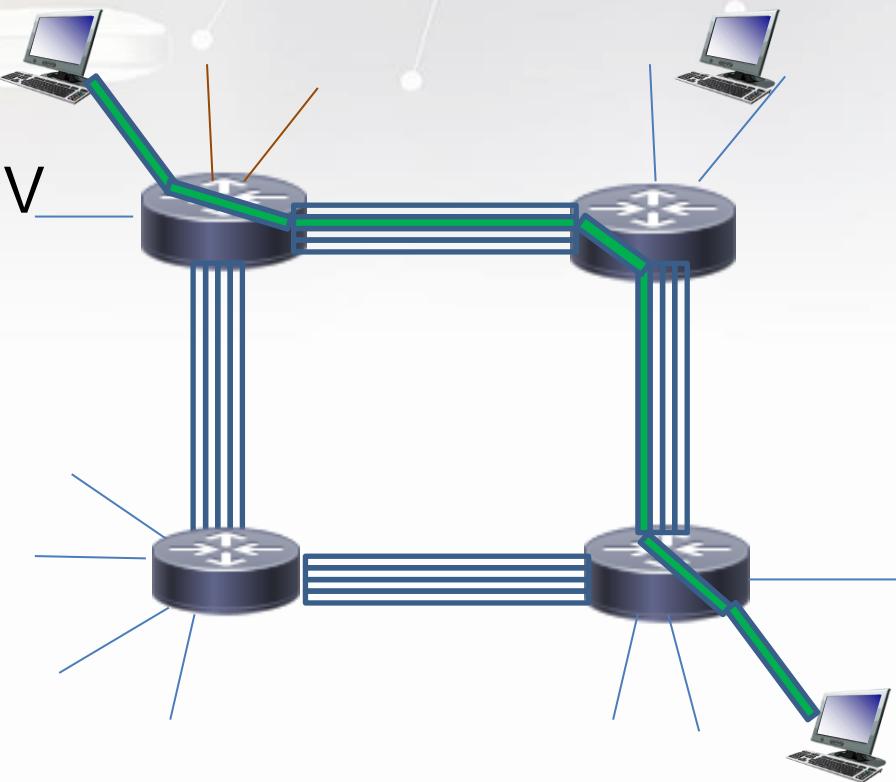
# Circuitos virtuales



- ◆ Establecer la conexión antes de enviar datos
- ◆ Cada paquete lleva un identificador de CV
- ◆ Cada router en el camino mantiene el estado de cada CV
- ◆ Opcional: reservar recursos para un CV
  - ◆ Bandwidth
  - ◆ Espacio en buffer

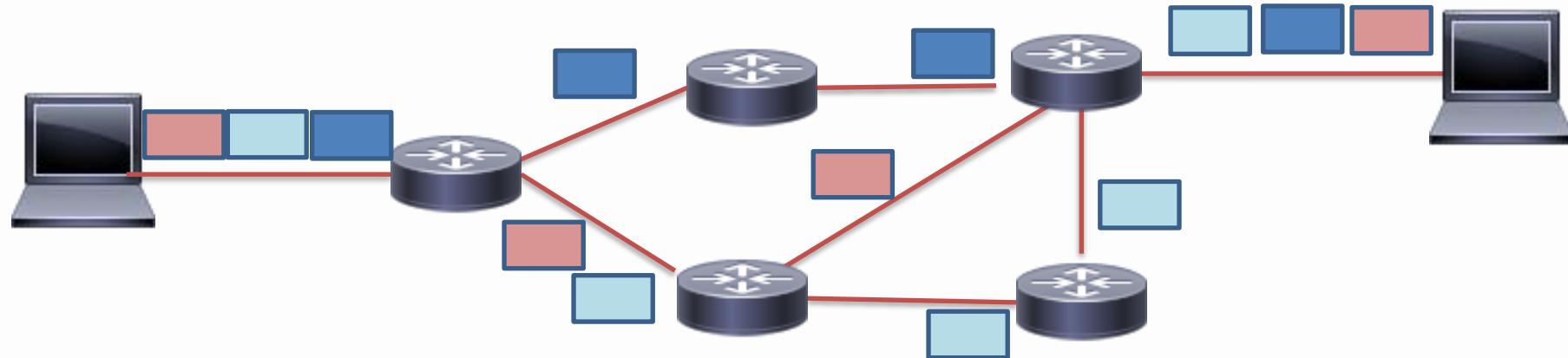
# Circuito virtual (Circuit switching)

- ◆ Establecer la conexión antes de enviar datos
- ◆ Cada paquete lleva un identificador de CV
- ◆ Cada router en el camino mantiene el estado de cada CV
- ◆ Opcional: reservar recursos para un CV
  - ◆ Bandwidth
  - ◆ Espacio en buffer



# Conmutación de paquetes

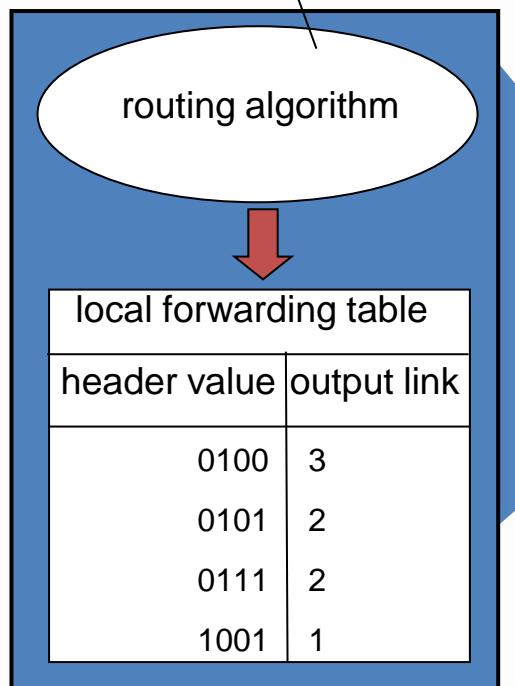
- “Datagram networks”
  - No hay llamada previa
  - Routers: no hay estado sobre las «puntas»
  - Los paquetes se «forwardean» en base a dirección de destino



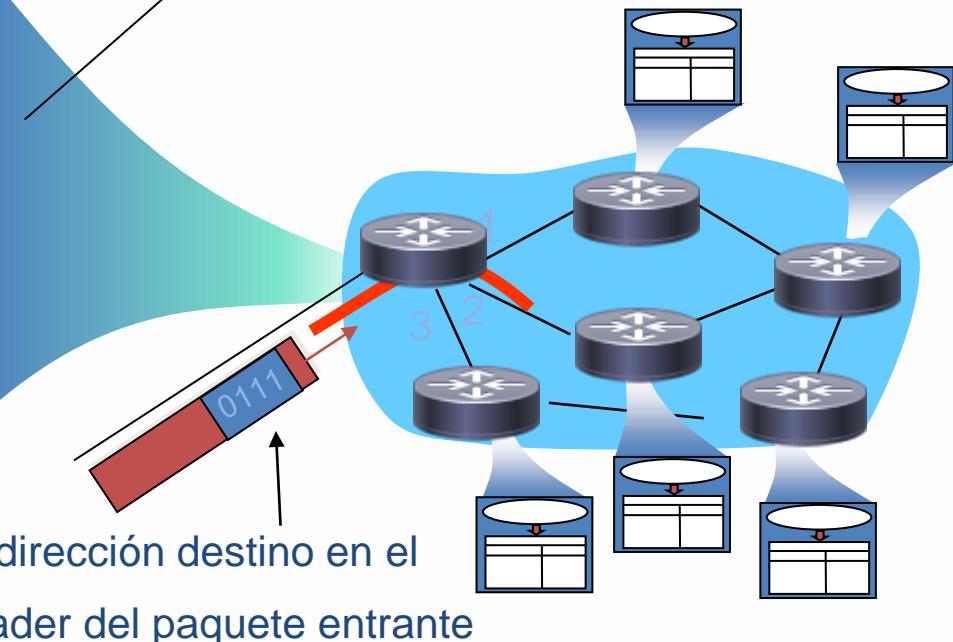
# Funciones de un router

*routing:* determinar la ruta que van a tomar los paquetes de acuerdo a su destino

- *algoritmos de routing*



*forwarding:* mover los paquetes del input a su correspondiente output



# C.V. vs Conmutación de paquetes

	Datagrama	Círculo virtual
Establecer conexión	---	Requerido
Direccionamiento	Id. De host origen y destino	Número de CV
Info de estado	No	Tabla de subred con números de CV en cada router
Enrutamiento	Cada paquete una ruta independiente	Todos los paquetes del CV una misma ruta
¿Si falla un nodo?	Se pierden algunos paquetes	Todos los CV que pasan por el nodo finalizan
Control de congestión	Complejo	Simple
Complejidad	En la capa de transporte	En la capa de red

# Capa de red en Internet

Capa de transporte: TCP, UDP, SCTP

**Protocolos de ruteo**  
(RIP, OSPF, BGP)  
Alimentan la tabla de  
ruteo

**Protocolo IP**

- Convención de direcciones
- Formato de paquetes
- Manejo de paquetes

**Protocolo ICMP**

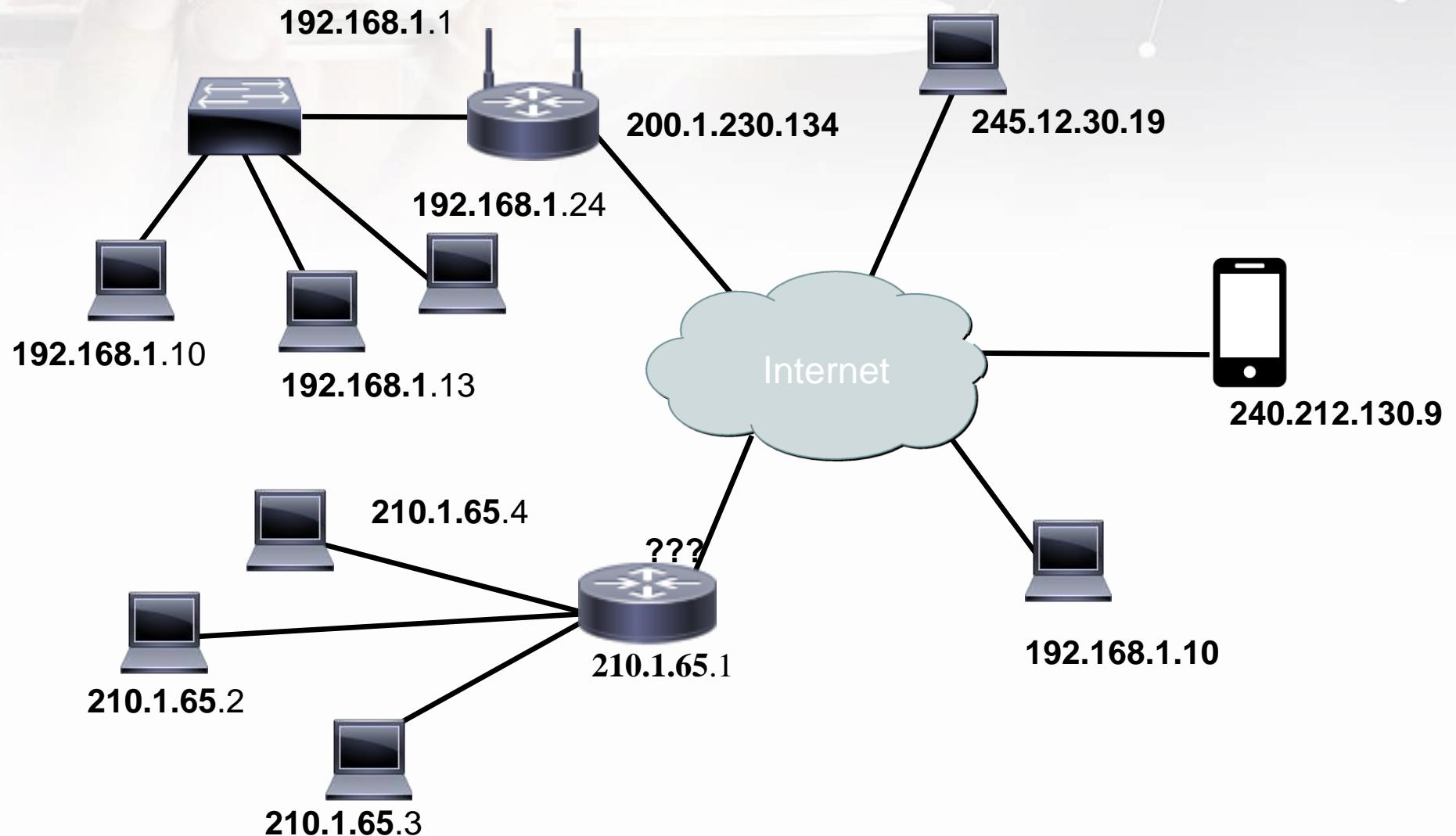
- Reporte de errores
- Avisos de enrutamiento

**Protocolo ARP**

Capa de enlace

Capa física

# Direccionamiento IPv4



# Direccionamiento IPv4

- ◆ Asigna una dirección única a cada host en Internet.
- ◆ Permite a cada host (computadora, impresora, router, etc.) en una red IP ser identificado únicamente.
- ◆ Cada dirección se forma con 32 bits (aprox. 3.700 millones de direcciones)
- ◆ Se escriben como 4 números separados por puntos (200.1.230.50, 192.168.2.1, etc.)
- ◆ Cada nro IP consta de 2 partes:
  - ◆ Network ID
  - ◆ Host ID
- ◆ Todos los host de una misma red comparten el Network ID

# Direccionamiento IPv4

## Clases de direcciones IP

A



B



C



D



E



# Direccionamiento IPv4

## Direcciones IP reservadas

- ◆ Loopback: las comenzadas con 127 (ej. 127.0.0.1)
- ◆ Identificar la red: *host id* con todos ceros
- ◆ Broadcast: *host id* con todos unos
- ◆ Redes privadas (RFC 1918):
  - ◆ 10.x.x.x
  - ◆ 172.16.0.0 a 172.31.255.255
  - ◆ 192.168.x.x
- ◆ Link local (RFC 6890)
  - ◆ 169.254.0.0/16

# Direccionamiento IPv4



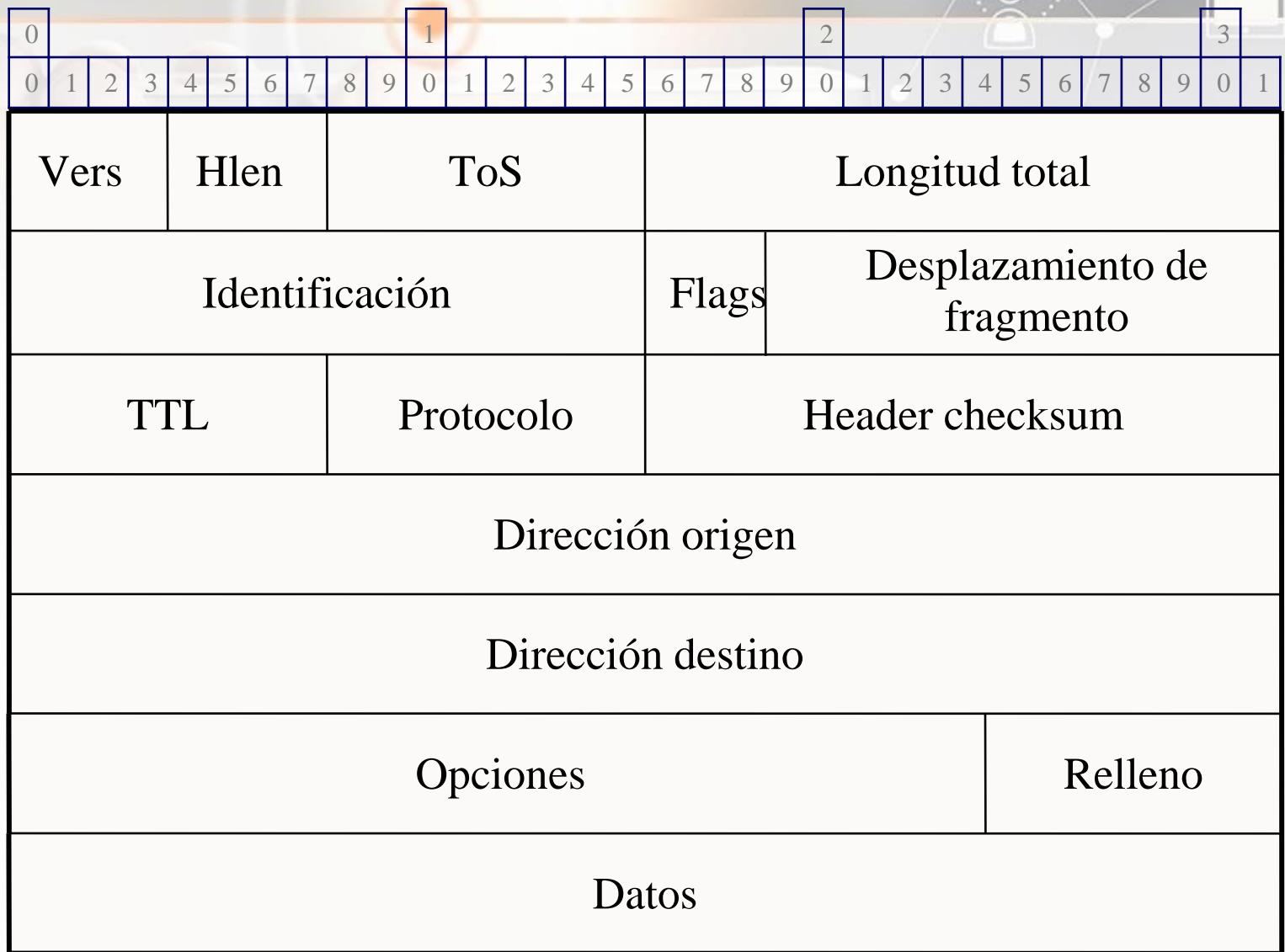
Clase IP	Rango	Máscara	Redes	Hosts en cada red
A	1.0.0.0 a 126.0.0.0	255.0.0.0	126	16,777,214
B	128.0.0.0 a 191.255.0.0	255.255.0.0	16,384	65,534
C	192.0.0.0 a 223.255.255.0	255.255.255.0	2,097,151	254

Dada una dirección IP se puede determinar la red a la cual pertenece y así “*rutear*” el paquete.

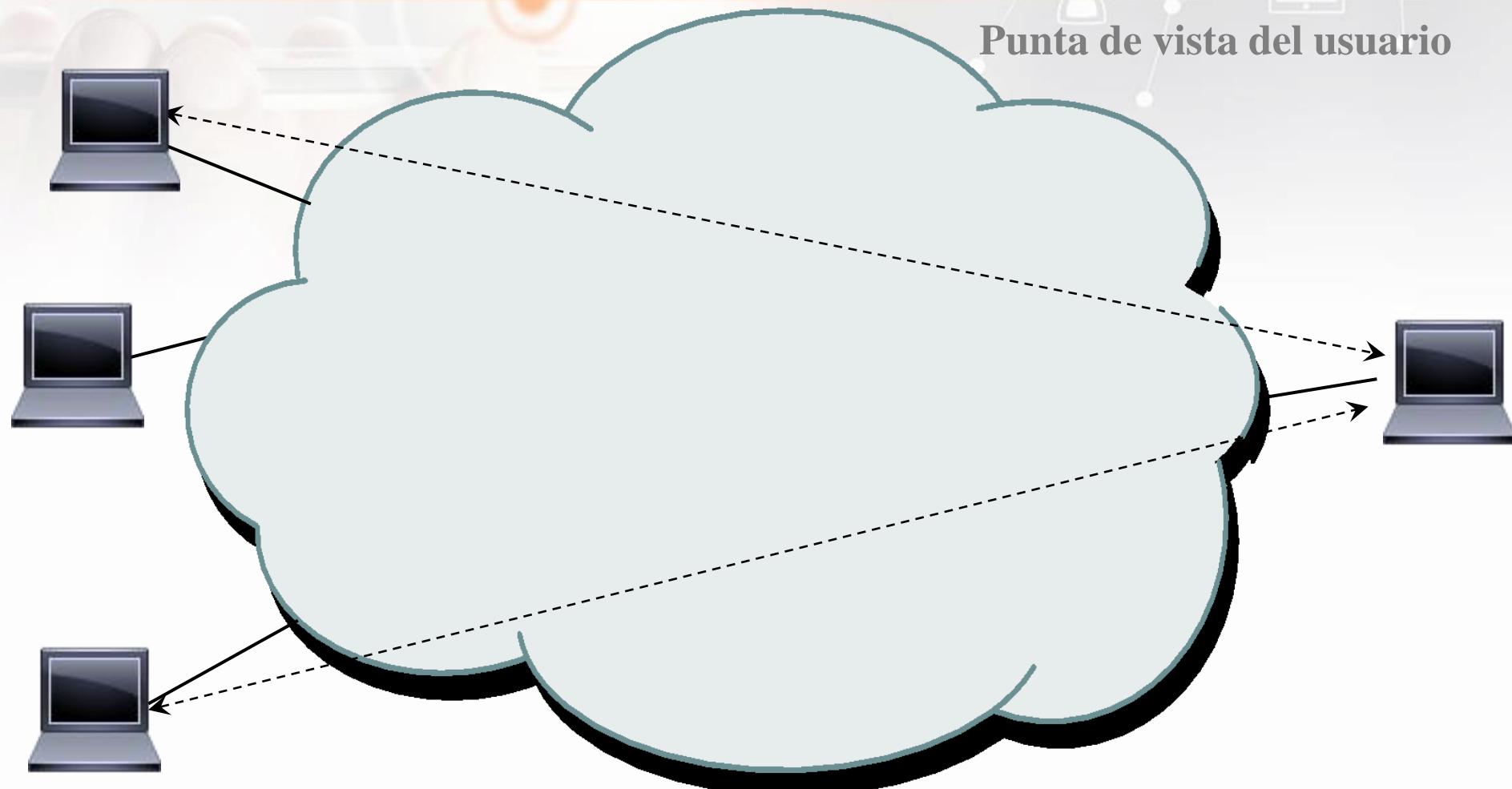
# Tabla de ruteo “classful”

Network	Gateway	Interface	Metric
11.0.0.0			
13.0.0.0			
129.10.0.0			
129.11.0.0			
129.12.0.0			
210.34.45.0			
210.45.34.0			
0.0.0.0 (*)			

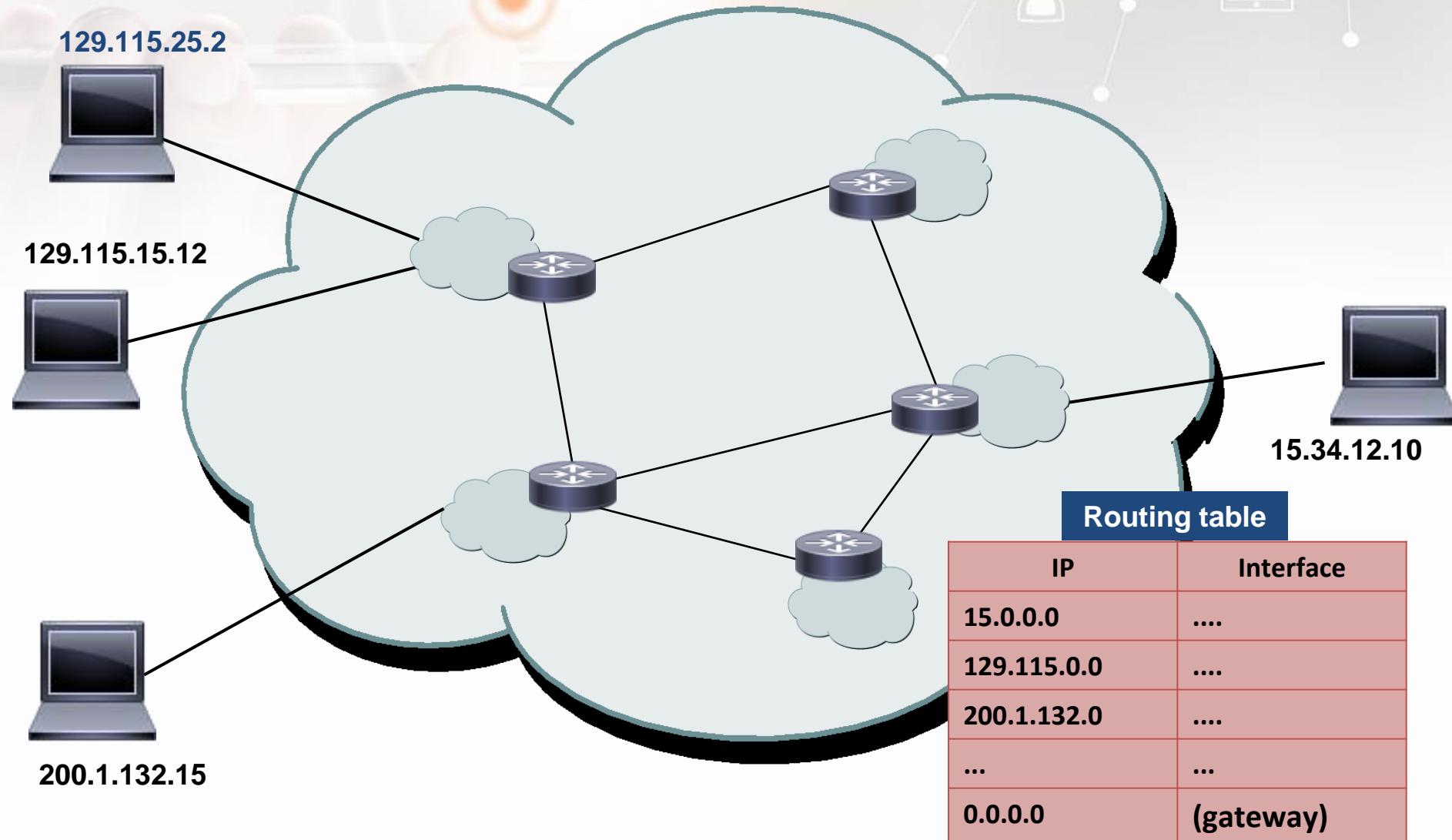
# Paquete IPv4 (datagrama)



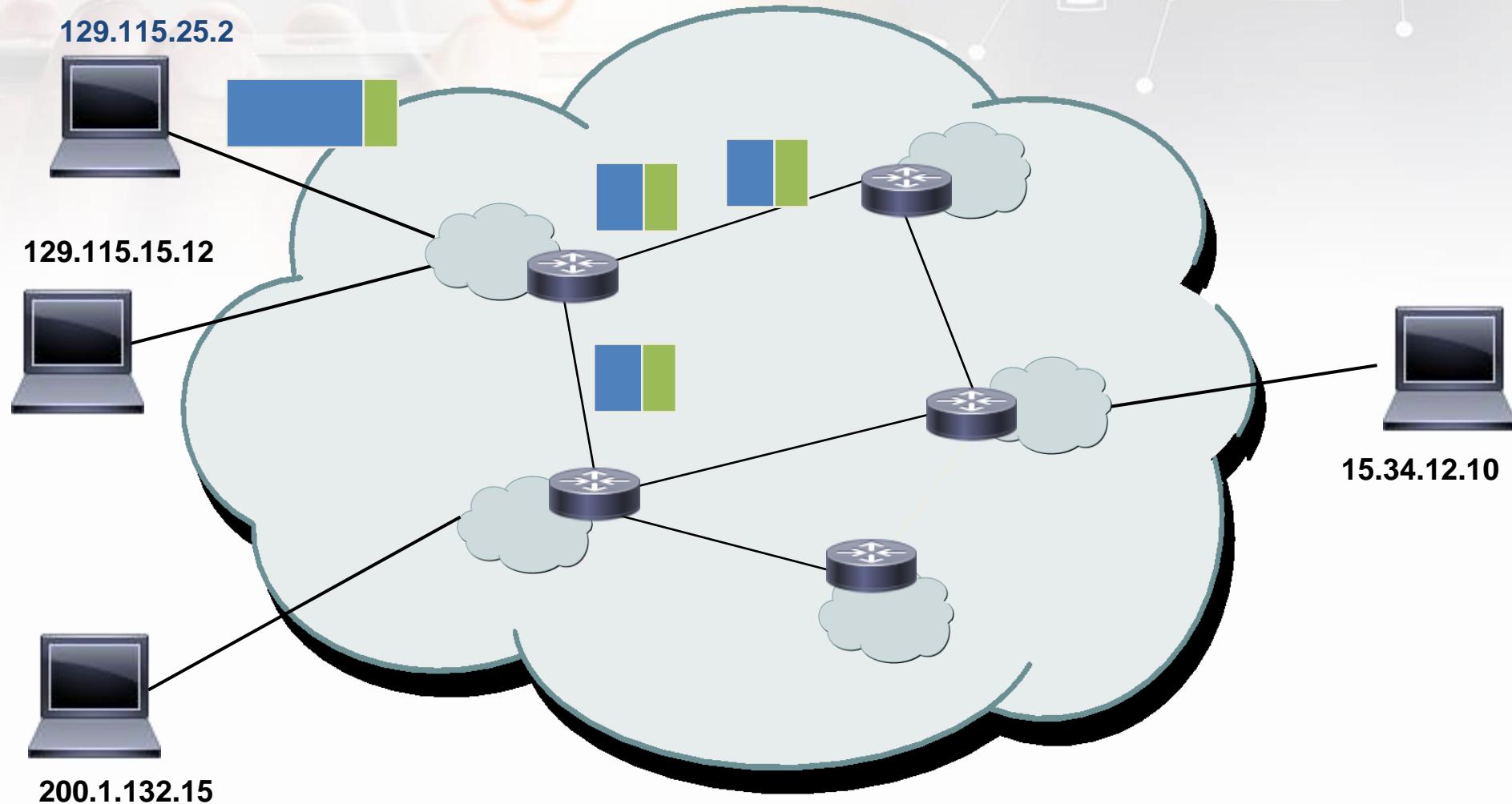
# Direccionamiento IP



# Direccionamiento IP



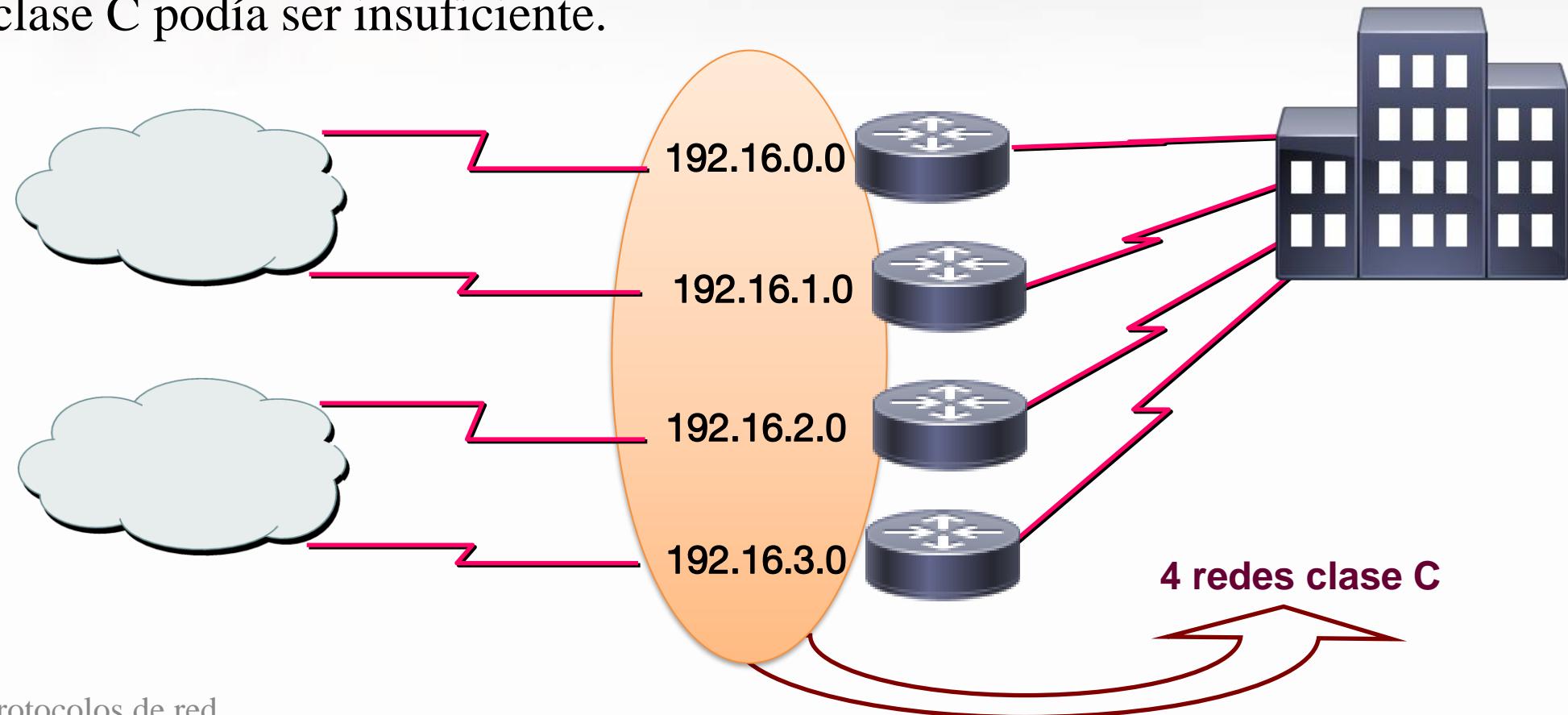
# Fragmentación y ensamblado



# Superredes

Cuando se diseñaron las clases de direcciones IP no se pensaba en un gran crecimiento en la cantidad de redes.

Las direcciones clase B se agotaban y para una organización una única dirección clase C podía ser insuficiente.



# Superredes

## Bloques CIDR (1993)

Uno de los problemas era mantener las tablas de los routers.

Se requería un nuevo esquema de asignación de direcciones, otorgando un rango de direcciones clase C pero identificada como una sola red (**superred**).

Se estableció el uso de una **máscara** en lugar de una clase para determinar la red de destino llamado CIDR (*Classless Inter-Domain Routing*).

# Bloques CIDR

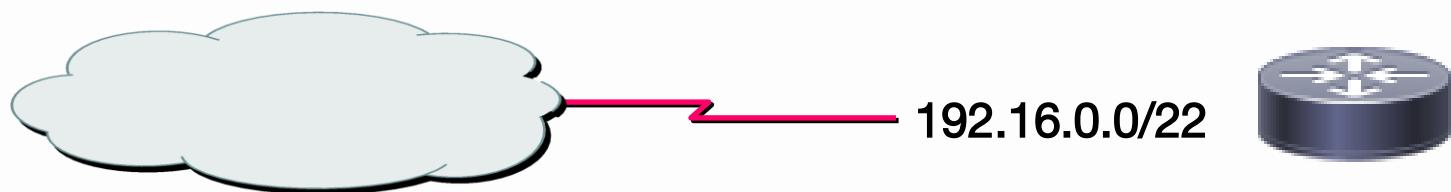


192.16.0.0 = 11000000 00010000 00000000 host

192.16.1.0 = 11000000 00010000 00000001 host

192.16.2.0 = 11000000 00010000 00000010 host

192.16.3.0 = 11000000 00010000 00000011 host



Dirección de red usando CIDR: **192.16.0.0/22**

Máscara de red:

**255.255.252.0**

# Bloques CIDR

En cada entrada de la tabla de ruteo se almacena la dirección de red junto con la cantidad de bits que forman la máscara.

Ejemplo: 200.1.230.0/24

*Ejemplo:* Un ISP obtiene el rango de direcciones  
192.100.0.0 a 192.100.7.255

Cant. de hosts: 2046 (*192.100.0.0 y 192.100.7.255 no se usan*)

Máscara: 255.255.248.0

Red: 192.100.0.0/21

# Ejemplo: tabla de ruteo pc "hogareña"

Destination	Gateway	Mask	Iface
0.0.0.0	192.168.1.1	0.0.0.0	eth0
192.168.1.0	*	255.255.255.0	eth0
192.168.1.15	*	255.255.255.255	lo
127.0.0.0	*	255.0.0.0	lo

# Subredes



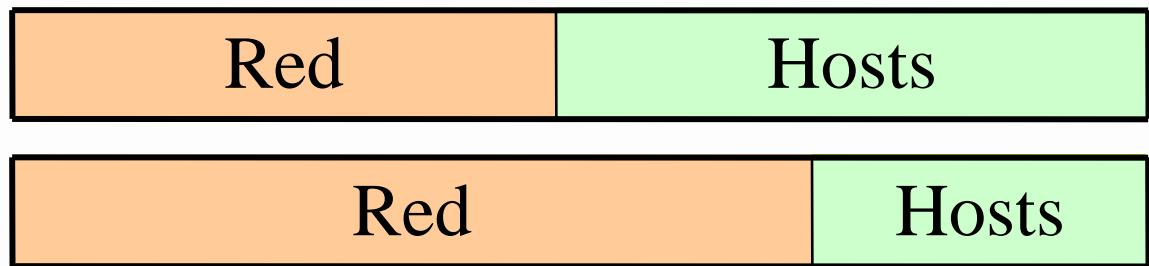
**Problema:** Una organización con dirección clase B. No se puede administrar como una única red de 65,534 hosts.

**Solución:** Dividir la red internamente en redes más pequeñas. Cada subred tendrá una máscara de acuerdo a la cantidad de hosts que necesite.

# Subredes

## Máscaras de subred

A los bits asignados a la red se le suman  $n$  bits asignados a los hosts y formar así  $2^n$  subredes.



Ver RFC 1812: *Requirements for IP Version 4 routers*

# Subredes



## Máscaras de subred (RFCs 1812 y 1878)

Ejercicio: Dada la red 192.100.32.0/24, se la quiere dividir en subredes de 12 hosts cada una.

Máscara: 255.255.255.240

Subred	Rango hosts	Broadcast
192.100.32.0	192.100.32.1 – 192.100.32.14	192.100.32.15
192.100.32.16	192.100.32.17 – 192.100.32.30	192.100.32.31
192.100.32.32	192.100.32.33 – 192.100.32.46	192.100.32.47
...		

Cantidad de subredes: 16

# Subredes

## Motivos para crear subredes

### Razones topológicas

- ◆ Hosts muy distantes
- ◆ Interconectar distintas capas físicas
- ◆ Filtrar tráfico entre redes

### Organización

- ◆ Simplificar la administración de la red
- ◆ Mapear la estructura de la organización
- ◆ Aislamiento del tráfico

# Direccionamiento IPv4

Clase IP	Rango	Máscara	Redes	Hosts en cada red
A	11.0.0.0 a 126.0.0.0	255.0.0.0	116	16,777,214
B	128.0.0.0 a 191.255.0.0	255.255.0.0	16,384	65,534
C	192.0.0.0 a 223.255.255.0	255.255.255.0	2,097,151	254

Cantidad de hosts: 3,552,542,234

¿ Será suficiente ?



## Capa de red

- ◆ Asignación de direcciones IP
- ◆ DNAT / SNAT
- ◆ ICMP
- ◆ IPv6

# Asignación de direcciones IP

Cada host en una red debe tener una dirección IP compatible con la red a la que pertenece, y a su vez dos hosts no pueden tener la misma dirección. Existen dos esquemas para asignación de direcciones

- ◆ **Direccionamiento estático:** se configura la interface con una dirección IP y máscara de red fija.
- ◆ **Direccionamiento dinámico:**
  - ◆ RARP ( RFC 903 )
  - ◆ BOOTP ( RFCs 951 y 1532 )
  - ◆ DHCP (RFCs 2131, 2132, upd: 3396, 3442, 3942)

# Asignación de direcciones IP

## BOOTP

Es usado por un dispositivo cuando éste se inicia para conocer cuál será su dirección IP. Envía un mensaje broadcast a la red con su número MAC preguntando por su número IP. Un servidor BOOTP le informa cuál es el número de IP y le puede informar también la máscara de red, gateway y otros datos.

BOOTP es reemplazado por DHCP.

# DHCP (*Dynamic Host Configuration Protocol*)

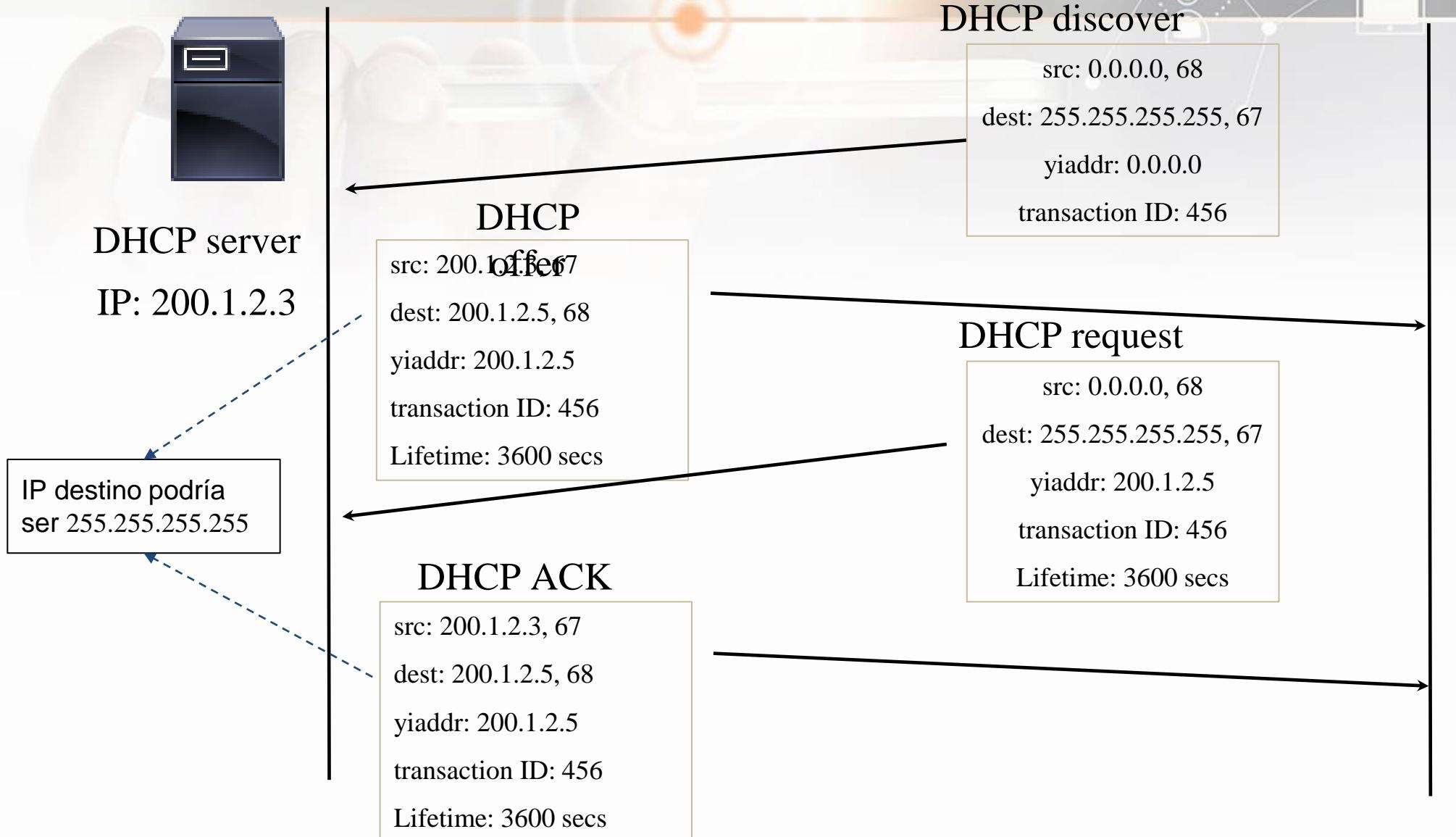
- ◆ Es un sucesor de BOOTP
- ◆ El cliente puede conocer o no la dirección IP del servidor DHCP
- ◆ La asignación de IP es dinámica, aunque permite reservar direcciones IP para direcciones MAC
- ◆ Permite obtener, además del número IP
  - ◆ Máscara de red
  - ◆ Gateways ( IP )
  - ◆ DNS servers (nombre e IP)
  - ◆ etc. ( ver RFC 2132 )

# DHCP

0	1	2	3
0 1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1
op (1)	htype (1)	hlen (1)	hops (1)
+-----+   xid (4)   +-----+			
secs (2)		flags (2)	
+-----+   ciaddr (4)   +-----+			
+-----+   yiaddr (4)   +-----+			
+-----+   siaddr (4)   +-----+			
+-----+   giaddr (4)   +-----+			
+-----+   chaddr (16)   +-----+			
+-----+   sname (64)   +-----+			
+-----+   file (128)   +-----+			
+-----+   options (variable)   +-----+			

Ver RFC 2131 item 4.4

# DHCP: solicitud



# DHCP: ejemplo

## Discover

```
▶ Internet Protocol Version 4, Src: 0.0.0.0, Dst: 255.255.255.255
◀ User Datagram Protocol, Src Port: 68, Dst Port: 67
  Source Port: 68
  Destination Port: 67
  Length: 308
  Checksum: 0x61a0 [unverified]
  [Checksum Status: Unverified]
  [Stream index: 0]
◀ Bootstrap Protocol (Discover)
  Message type: Boot Request (1)
  Hardware type: Ethernet (0x01)
  Hardware address length: 6
  Hops: 0
  Transaction ID: 0xfe089c15
  Seconds elapsed: 0
  ▶ Bootp flags: 0x0000 (Unicast)
    Client IP address: 0.0.0.0
    Your (client) IP address: 0.0.0.0
    Next server IP address: 0.0.0.0
    Relay agent IP address: 0.0.0.0
    Client MAC address: D-Link_12:47:cb (00:50:ba:12:47:cb)
    Client hardware address padding: 000000000000000000000000
    Server host name not given
    Boot file name not given
    Magic cookie: DHCP
  ▶ Option: (53) DHCP Message Type (Discover)
  ▶ Option: (116) DHCP Auto-Configuration
  ▶ Option: (61) Client identifier
  ▶ Option: (50) Requested IP Address
  ▶ Option: (12) Host Name
  ▶ Option: (60) Vendor class identifier
  ▶ Option: (55) Parameter Request List
  ▶ Option: (255) End
  Padding: 00000000
```

# DHCP: ejemplo

## Offer

# DHCP: ejemplo

## Request

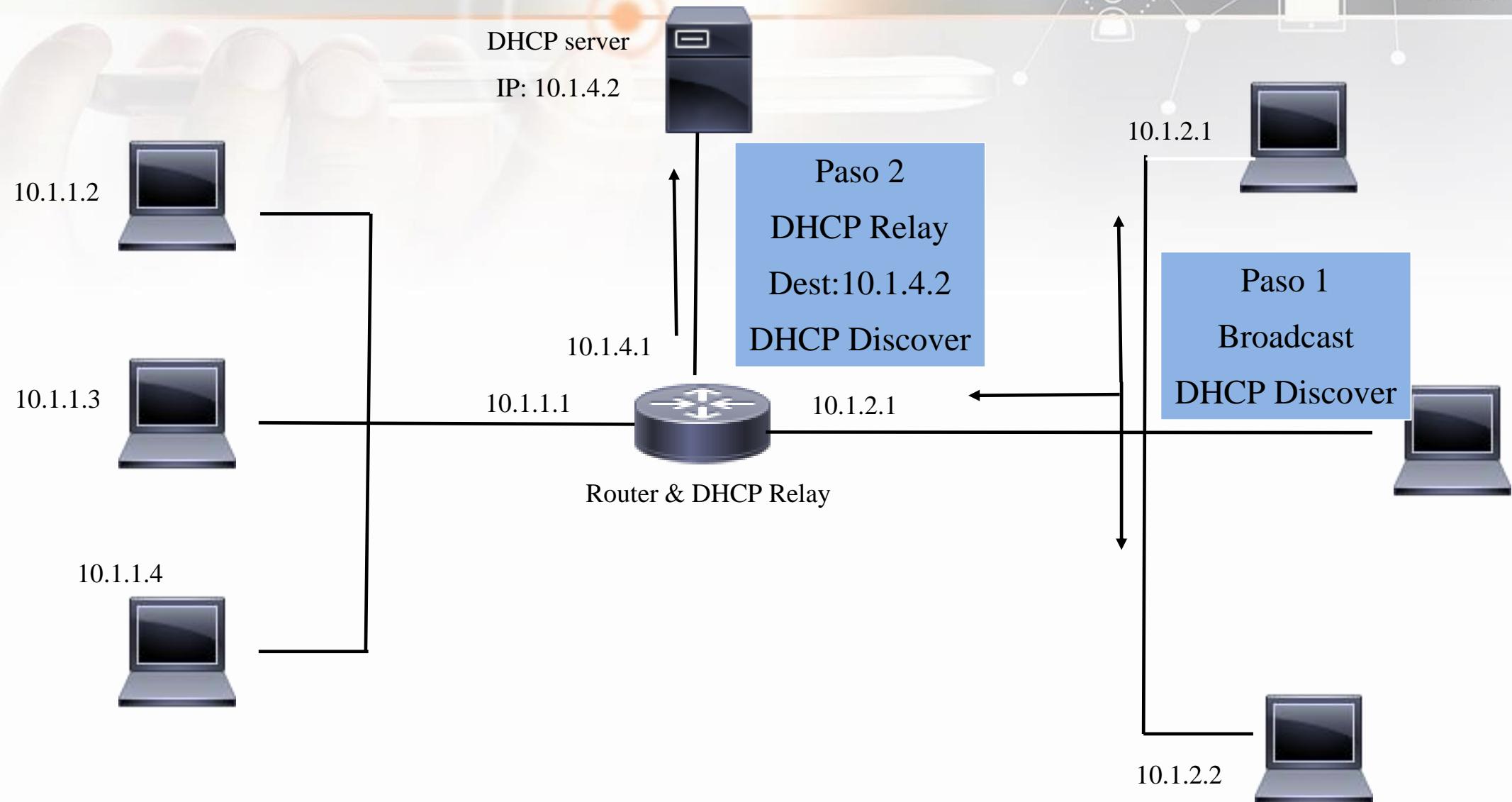
```
▶ Internet Protocol Version 4, Src: 0.0.0.0, Dst: 255.255.255.255
◀ User Datagram Protocol, Src Port: 68, Dst Port: 67
    Source Port: 68
    Destination Port: 67
    Length: 322
    Checksum: 0xf62b [unverified]
    [Checksum Status: Unverified]
    [Stream index: 0]
▶ Bootstrap Protocol (Request)
    Message type: Boot Request (1)
    Hardware type: Ethernet (0x01)
    Hardware address length: 6
    Hops: 0
    Transaction ID: 0xfe089c15
    Seconds elapsed: 0
    ▶ Bootp flags: 0x0000 (Unicast)
        Client IP address: 0.0.0.0
        Your (client) IP address: 0.0.0.0
        Next server IP address: 0.0.0.0
        Relay agent IP address: 0.0.0.0
        Client MAC address: D-Link_12:47:cb (00:50:ba:12:47:cb)
        Client hardware address padding: 000000000000000000000000
        Server host name not given
        Boot file name not given
        Magic cookie: DHCP
    ▶ Option: (53) DHCP Message Type (Request)
    ▶ Option: (61) Client identifier
    ▶ Option: (50) Requested IP Address
    ▶ Option: (54) DHCP Server Identifier
    ▶ Option: (12) Host Name
    ▶ Option: (81) Client Fully Qualified Domain Name
    ▶ Option: (60) Vendor class identifier
    ▶ Option: (55) Parameter Request List
    ▶ Option: (255) End
```

# DHCP: ejemplo

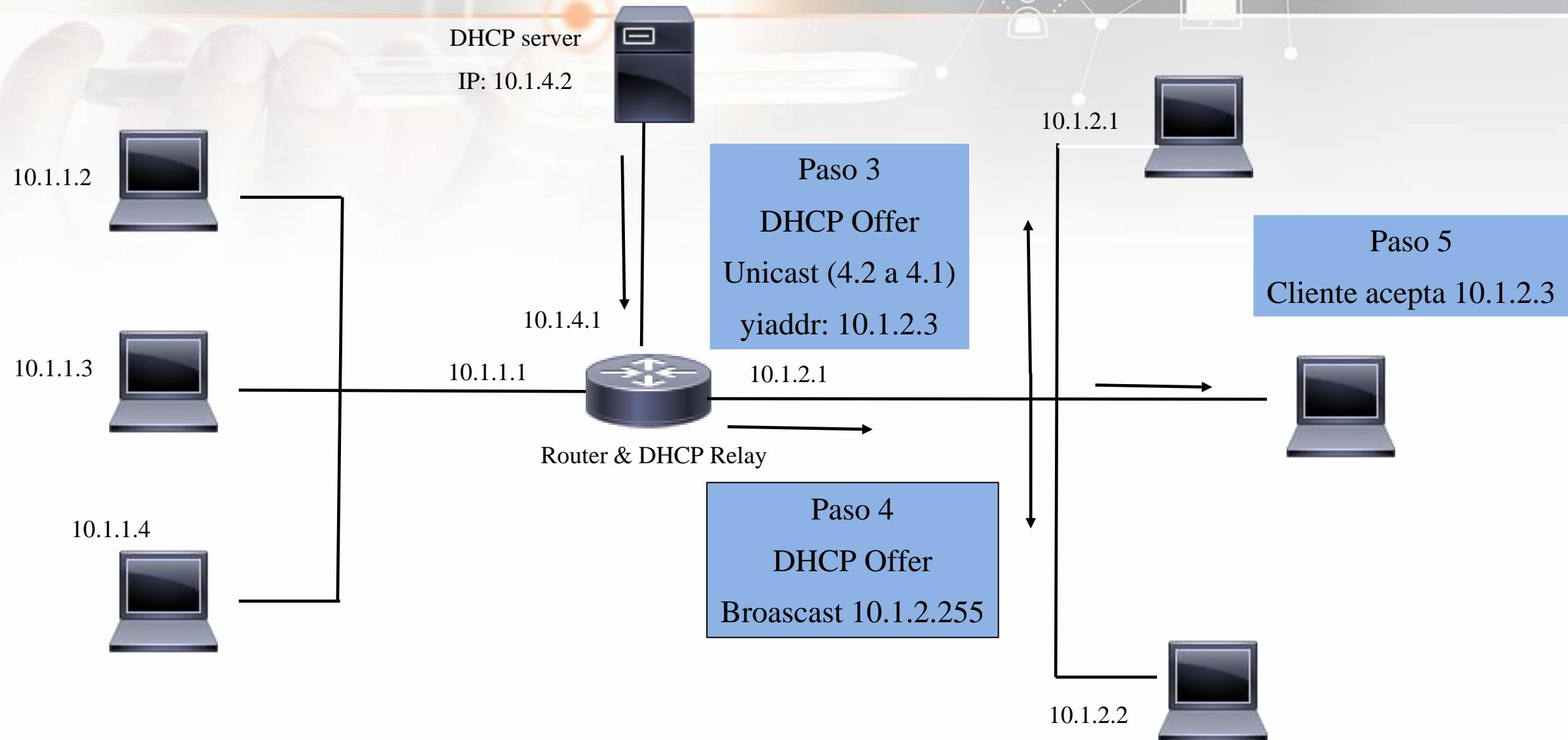
ACK

Internet Protocol Version 4, Src: 10.20.20.4, Dst: 255.255.255.255  
User Datagram Protocol, Src Port: 67, Dst Port: 68  
Source Port: 67  
Destination Port: 68  
Length: 317  
Checksum: 0xb7a8 [unverified]  
[Checksum Status: Unverified]  
[Stream index: 2]  
Bootstrap Protocol (ACK)  
Message type: Boot Reply (2)  
Hardware type: Ethernet (0x01)  
Hardware address length: 6  
Hops: 0  
Transaction ID: 0xfe089c15  
Seconds elapsed: 0  
Bootp flags: 0x0000 (Unicast)  
Client IP address: 0.0.0.0  
Your (client) IP address: 10.20.20.20  
Next server IP address: 10.20.20.4  
Relay agent IP address: 0.0.0.0  
Client MAC address: D-Link\_12:47:cb (00:50:ba:12:47:cb)  
Client hardware address padding: 000000000000000000000000  
Server host name not given  
Boot file name not given  
Magic cookie: DHCP  
Option: (53) DHCP Message Type (ACK)  
Option: (54) DHCP Server Identifier  
Option: (51) IP Address Lease Time  
Option: (81) Client Fully Qualified Domain Name  
Option: (1) Subnet Mask  
Option: (15) Domain Name  
Option: (6) Domain Name Server  
Option: (255) End

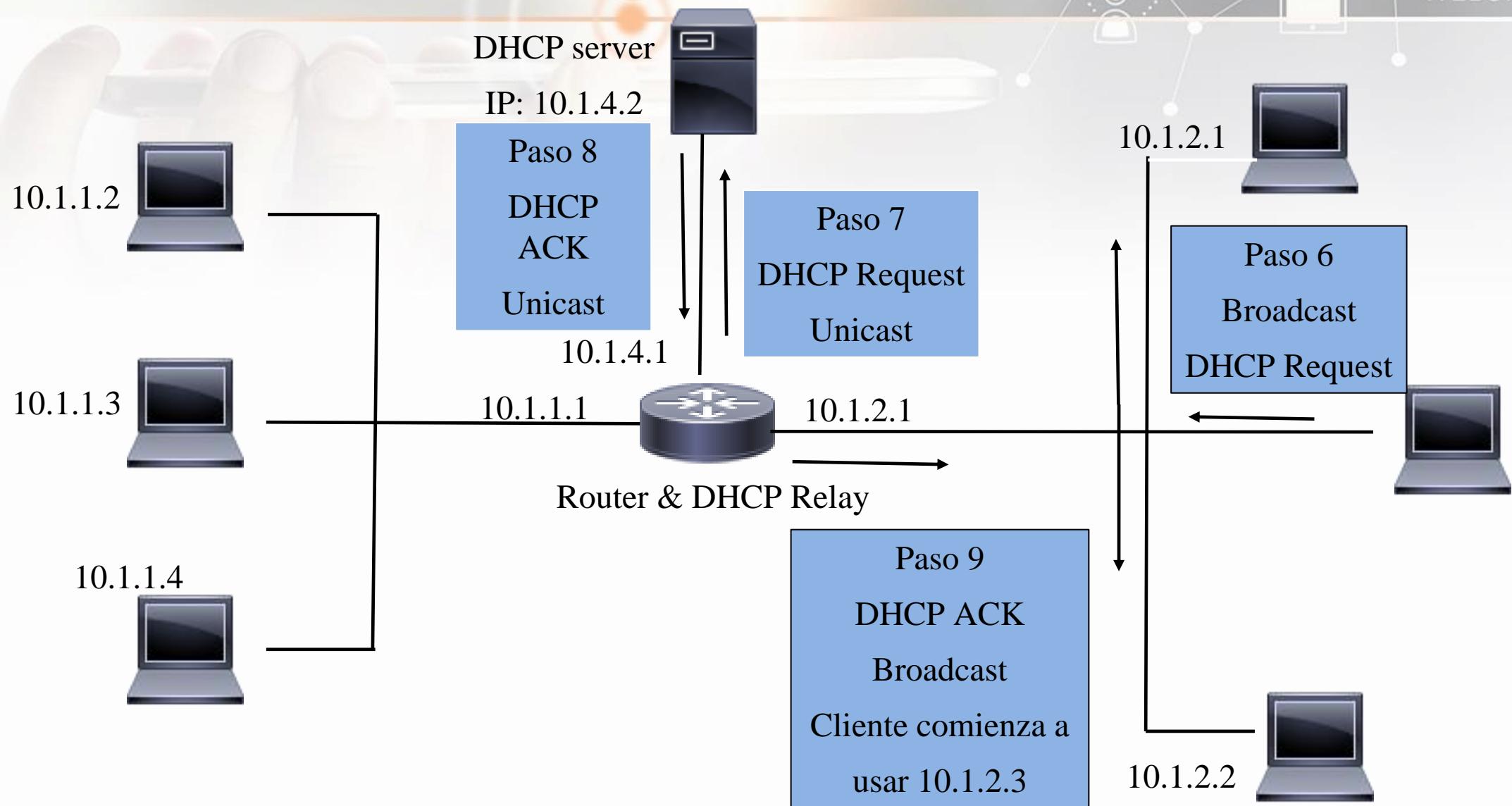
# DHCP: relay agent



# DHCP: relay agent



# DHCP: relay agent



# DHCP: renew request

```
► Internet Protocol Version 4, Src: 192.168.1.101, Dst: 192.168.1.1
► User Datagram Protocol, Src Port: 68, Dst Port: 67
◀ Bootstrap Protocol (Request)
  Message type: Boot Request (1)
  Hardware type: Ethernet (0x01)
  Hardware address length: 6
  Hops: 0
  Transaction ID: 0x0dd1b124
  Seconds elapsed: 0
  ▶ Bootp flags: 0x0000 (Unicast)
  Client IP address: 192.168.1.101
  Your (client) IP address: 0.0.0.0
  Next server IP address: 0.0.0.0
  Relay agent IP address: 0.0.0.0
  Client MAC address: AsustekC_22:77:a0 (40:16:7e:22:77:a0)
  Client hardware address padding: 00000000000000000000000000000000
  Server host name not given
  Boot file name not given
  Magic cookie: DHCP
  ▶ Option: (53) DHCP Message Type (Request)
  ▲ Option: (61) Client identifier
    Length: 7
    Hardware type: Ethernet (0x01)
    Client MAC address: AsustekC_22:77:a0 (40:16:7e:22:77:a0)

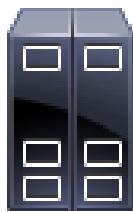
  ▶ Option: (12) Host Name
  ▶ Option: (81) Client Fully Qualified Domain Name
  ▶ Option: (60) Vendor class identifier
  ▲ Option: (55) Parameter Request List
    Length: 12
    Parameter Request List Item: (1) Subnet Mask
    Parameter Request List Item: (15) Domain Name
    Parameter Request List Item: (3) Router
    Parameter Request List Item: (6) Domain Name Server
    Parameter Request List Item: (44) NetBIOS over TCP/IP Name Server
    Parameter Request List Item: (46) NetBIOS over TCP/IP Node Type
    Parameter Request List Item: (47) NetBIOS over TCP/IP Scope
    Parameter Request List Item: (31) Perform Router Discover
    Parameter Request List Item: (33) Static Route
    Parameter Request List Item: (121) Classless Static Route
    Parameter Request List Item: (249) Private/Classless Static Route (Microsoft)
    Parameter Request List Item: (43) Vendor-Specific Information
  ▶ Option: (255) End
```

# DHCP: renew ACK

# DNS dinámico

## Integración con DHCP

Cliente  
DHCP



DHCP discover

DHCP offer

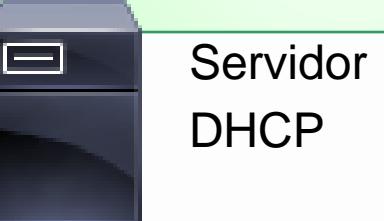
DHCP request

DHCP ACK

DHCP code 81 +  
FQDN cliente DHCP

dhcp.conf

```
ddns-updates on;  
ddns-update-style interim;  
ddns-domainname "midominio.com";  
ddns-rev-domainname "in-addr.arpa";  
option domain-name "midominio.com";
```



Servidor  
DHCP

Registro de  
actualización

Servidor  
DNS



# DDNS: ejemplo



## Security

View and change router settings

Firewall

DMZ

Apps and Gaming

[DDNS](#) | [Single Port Forwarding](#) | [Port Range Forwarding](#) | [Port Range Triggering](#)

Select a provider: [NO-IP.com](#) ▾

User Name:

[mgarbe@itba.edu.ar](#)

Internet IP address: [REDACTED]

No-IP Password: [REDACTED]

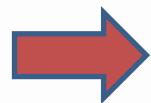
Status: Success

Host name: [REDACTED]

[Update](#)

# DHCP

DHCP es atendido por un servidor DHCP, no por un router



+



+

**DNS Caching-only Server**

+

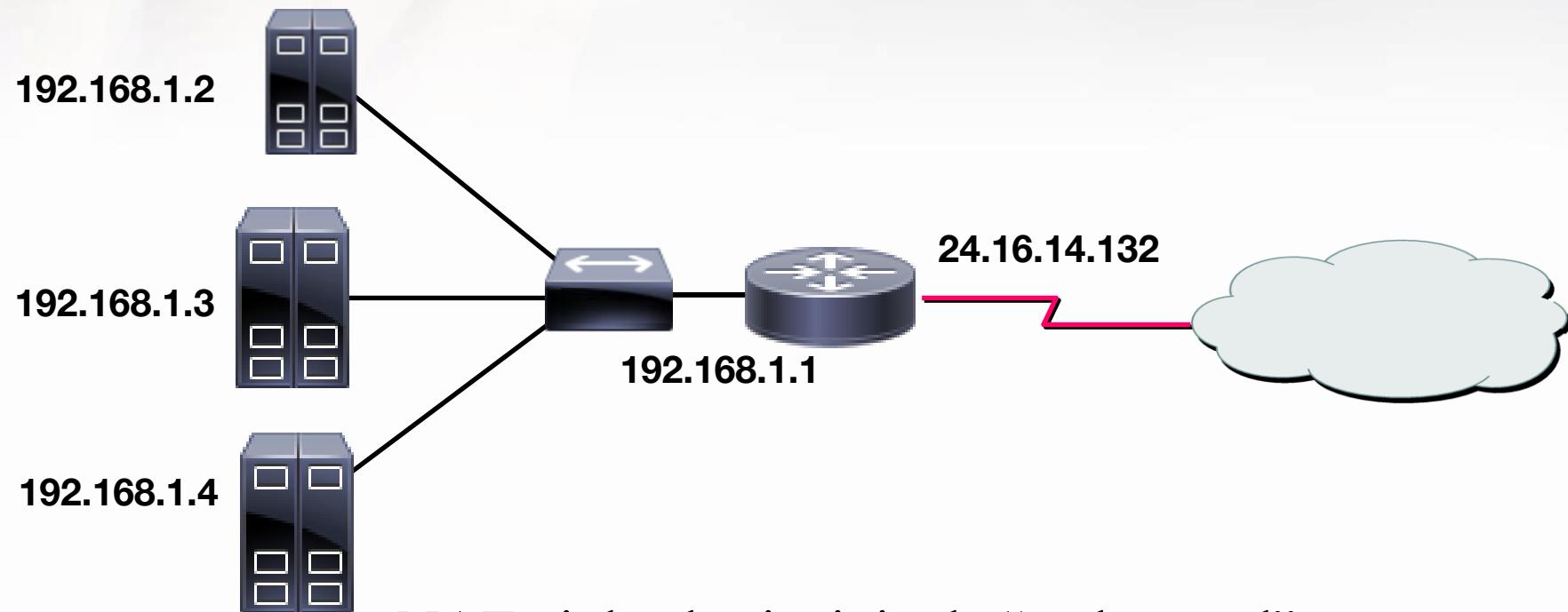
**DHCP Server**

+

**Firewall**

# NAT (SNAT)

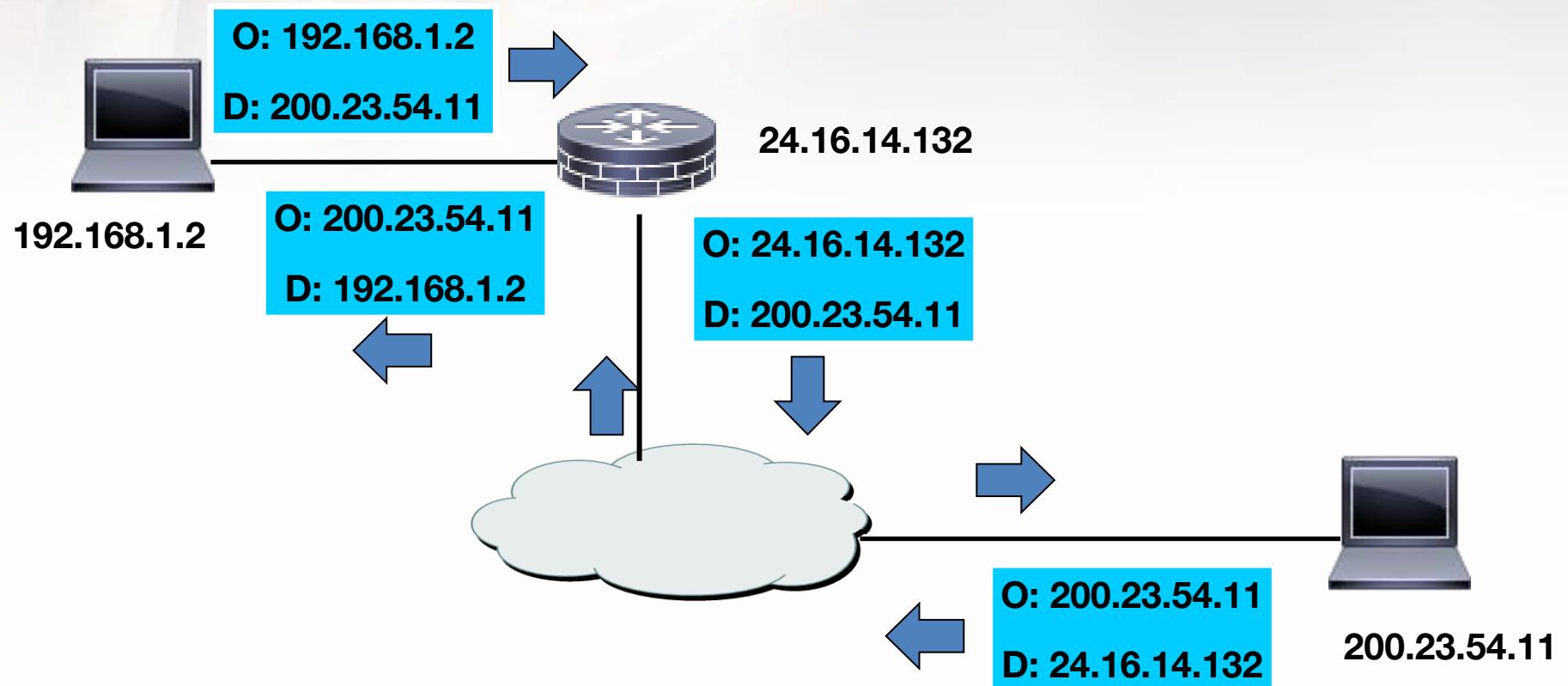
Una medida para aliviar la necesidad de IPs públicas fue **NAT**  
(1994, RFCs 1631 y 3022 )



NAT viola el principio de “end-to-end”.  
Debe ser tenido en cuenta por aplicaciones P2P

# SNAT

## Ejemplo



# SNAT

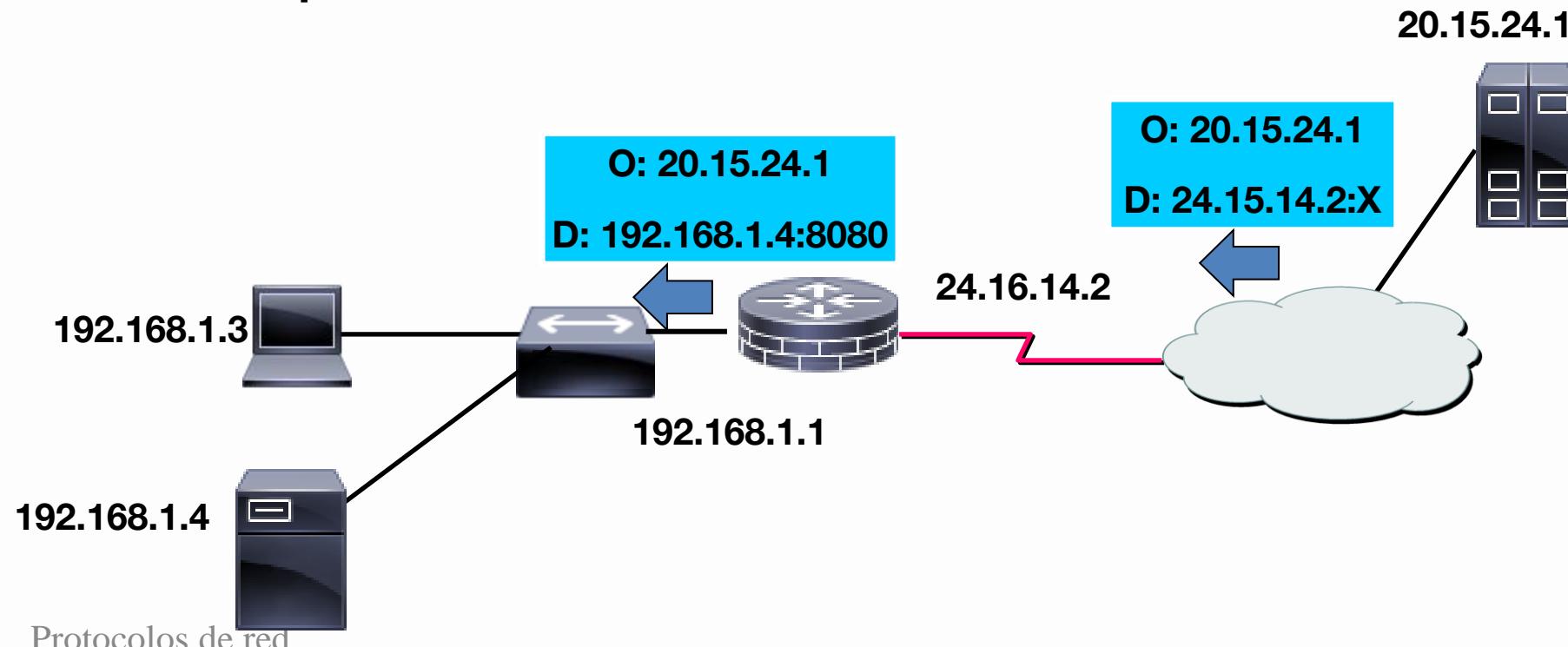


¿Qué campos se modifican al aplicar NAT a un paquete IP?

Vers	Hlen	ToS	Longitud total	
Identificación		Flags	Desplazam. de fragmento	
TTL	Protocolo	Header checksum		
Dirección origen				
Dirección destino				
Opciones			Relleno	
Datos				

# DNAT: port forwarding

- ◆ Quiero conectarme a un servidor en 192.168.1.4:8080
- ◆ Solución 1: configurar NAT para reenviar conexiones entrantes de puerto X a 192.168.1.4:8080



# DNAT: Ejemplo



## Security

View and change router settings

Firewall   DMZ   Apps and Gaming

DDNS | Single Port Forwarding | Port Range Forwarding | Port Range Triggering

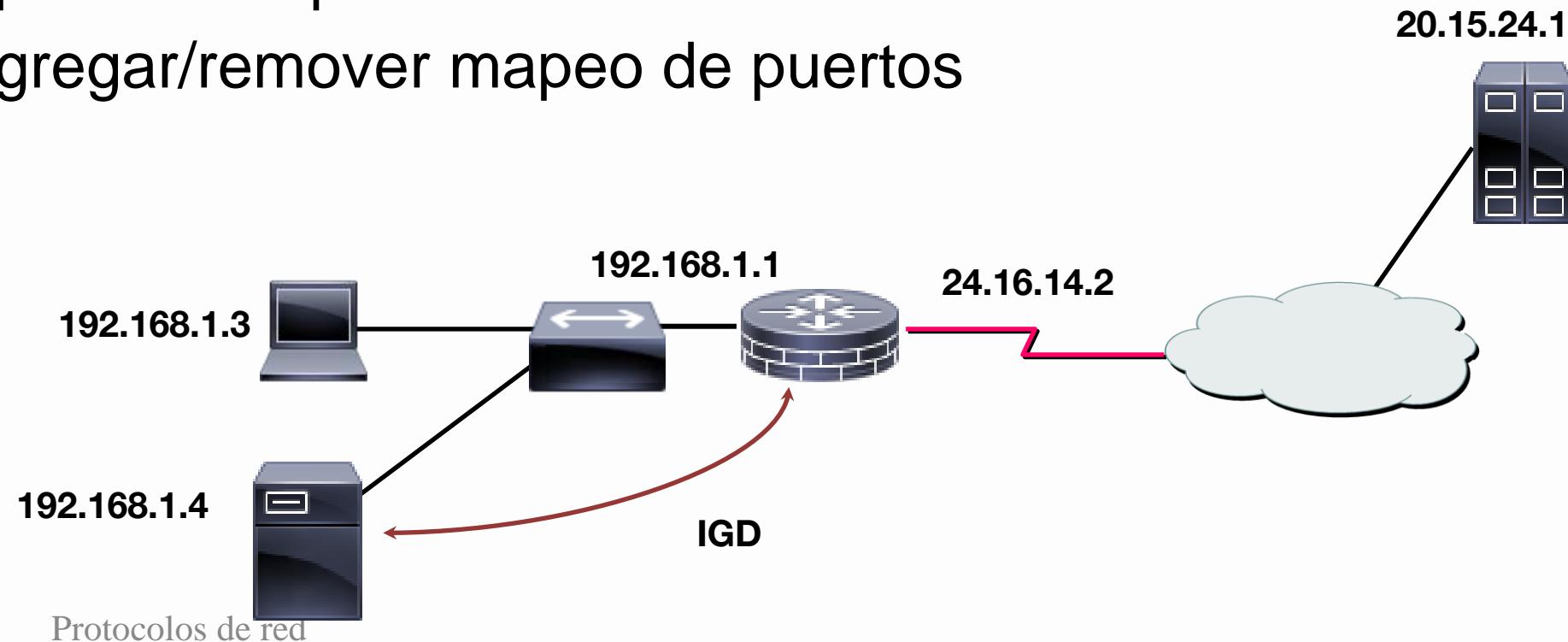
Application name	External Port	Internal Port	Protocol	Device IP#	Enabled	
Apache	9090	8080	TCP	192.168.1.142	True	<a href="#">Edit/Delete</a>
Remoto Marce	4489	3389	Both	192.168.1.127	True	<a href="#">Edit/Delete</a>
Apache Crea	9093	8080	Both	192.168.1.149	True	<a href="#">Edit/Delete</a>
Front	9100	8100	Both	192.168.1.149	True	<a href="#">Edit/Delete</a>
Back	9105	8105	Both	192.168.1.149	True	<a href="#">Edit/Delete</a>

Add a new Single Port Forwarding

# DNAT: port forwarding

- ◆ Solución 2: Universal Plug and Play (UPnP) Internet Gateway Device (IGD) Protocol. Permite al host “nateado”

- ◆ Aprender IP pública
- ◆ Agregar/remover mapeo de puertos



# UPnP



## UPnP

Note: Make sure the nat is **enable** if you want the UPnP configuration take effect

Current UPnP Status:

Enabled

[Disable](#)

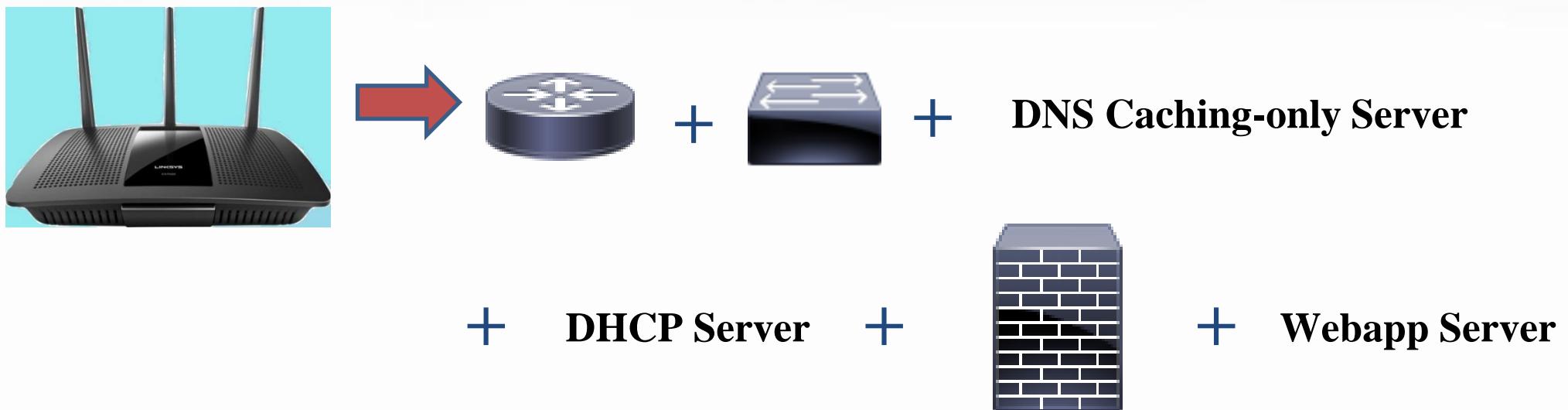
### Current UPnP Settings List

ID	App Description	External Port	Protocol	Internal Port	IP Address	Status
1	uTorrent (TCP)	21103	TCP	21103	192.168.1.106	Enabled
2	uTorrent (UDP)	21103	UDP	21103	192.168.1.106	Enabled

[Refresh](#)

# NAT

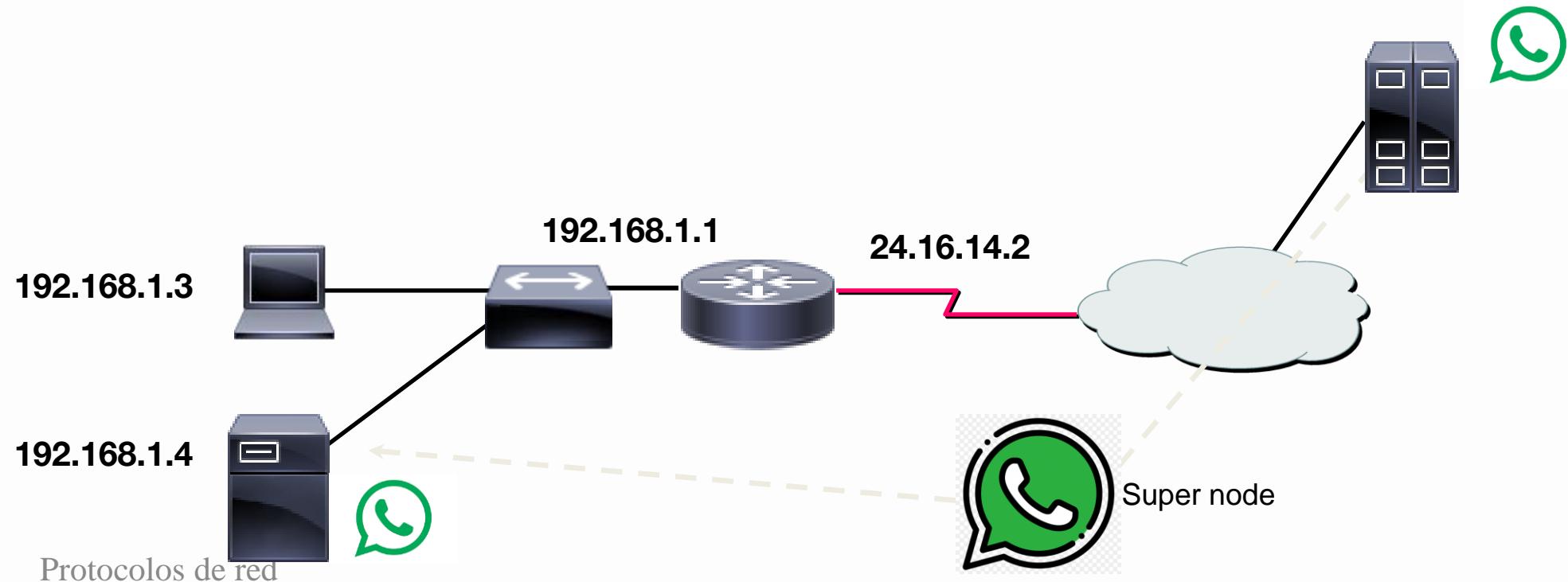
Tanto DNAT como SNAT son funciones de un firewall, no de un router



# NAT

## ♦ Solución 3: relaying (p.e. Skype, Whatsapp)

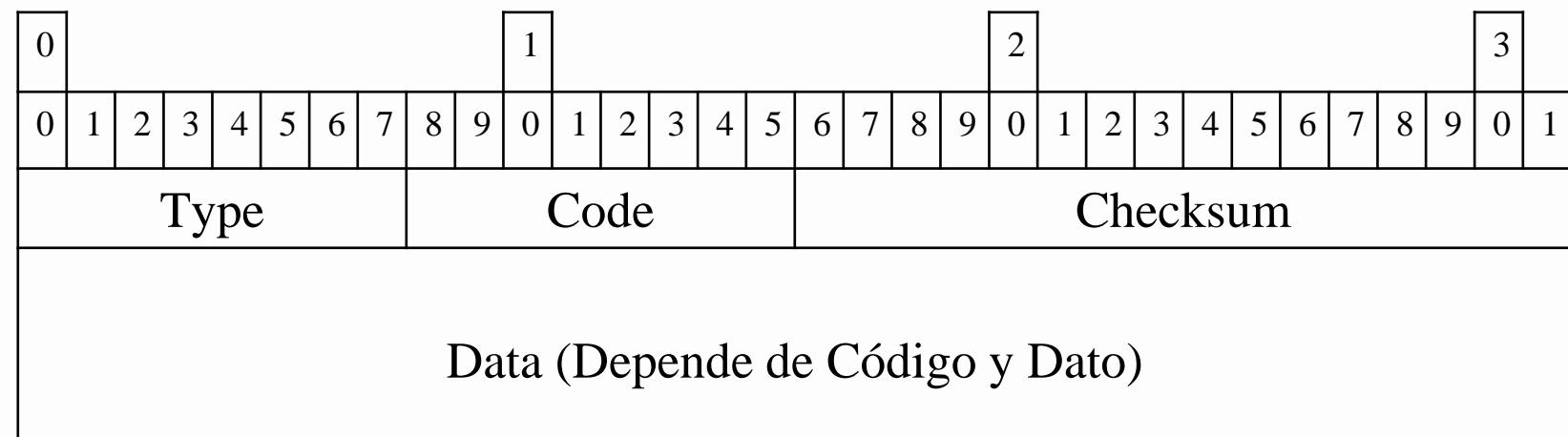
- ♦ Host “nateado” conecta con “relay”
- ♦ Host externo conecta con “relay”
- ♦ El “relay” funciona como puente o proxy



# ICMP



**IP no maneja errores ni retransmisiones.  
En caso de querer avisar de un error se debe usar  
mensajes ICMP**



Los valores para los campos Tipo y Código se definen en RFC1700

# ICMP: Tipo y Código más comunes

Tipo	Código	Descripción
3	0	Net unreachable
3	1	Host unreachable
3	2	Protocol unreachable
0	0	Echo reply
8	0	Echo request
5	0	Redirect datagrams for the Network
5	1	Redirect datagrams for the Host
11	0	TTL exceeded
13/14	0	Timestamp request/reply

# ICMP: ejemplos

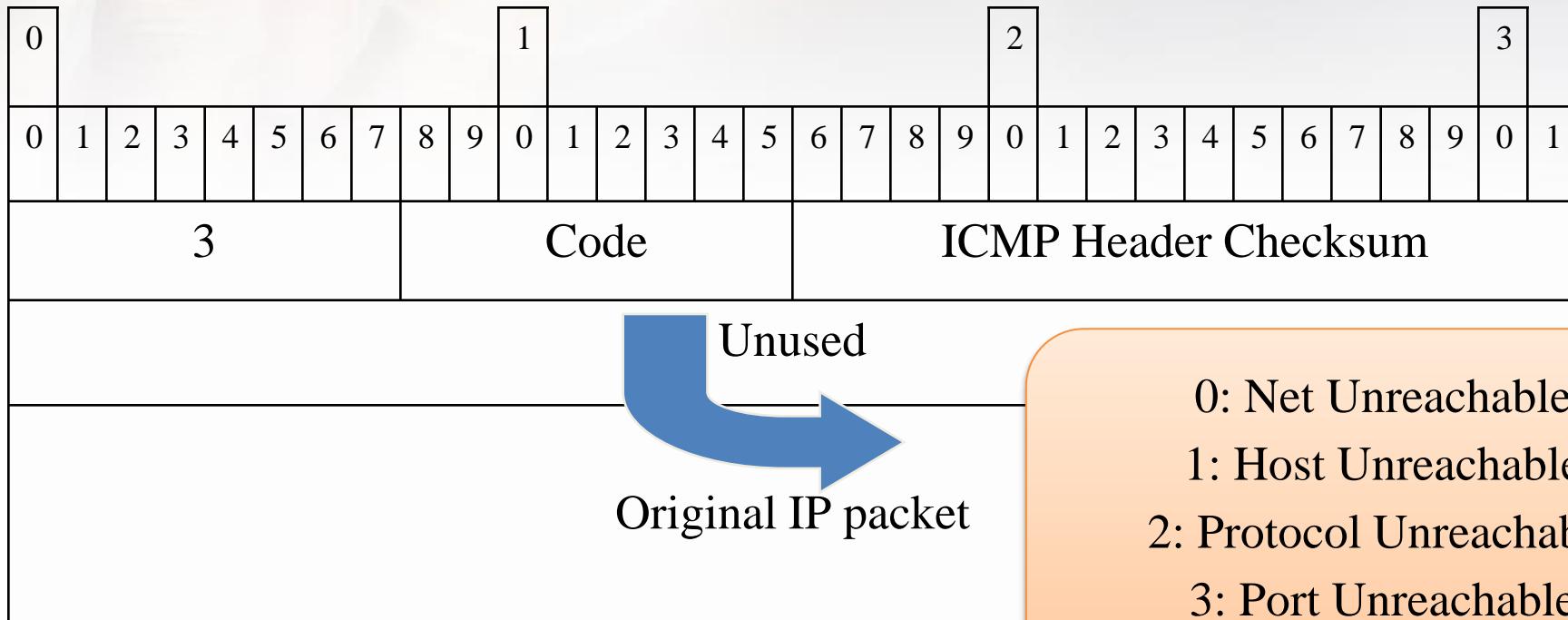
Timestamp request / reply

0									1										2									3																																	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1																														
13/14										0								ICMP Header Checksum																																											
Identifier																Sequence number																																													
Originate timestamp																																																													
Receive timestamp																																																													
Transit timestamp																																																													

# ICMP: ejemplos



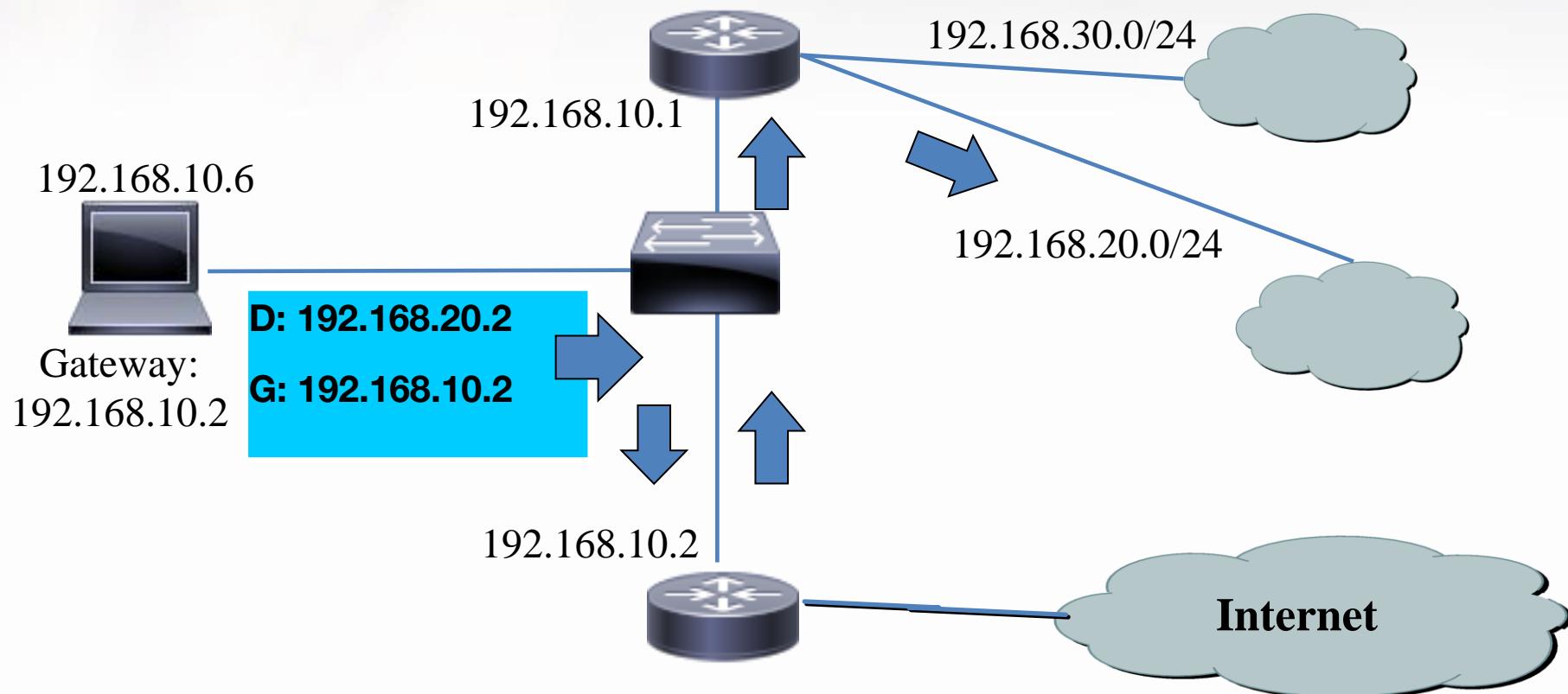
Destino inalcanzable



- 0: Net Unreachable
- 1: Host Unreachable
- 2: Protocol Unreachable
- 3: Port Unreachable
- 4: Fragmentation needed
- ...

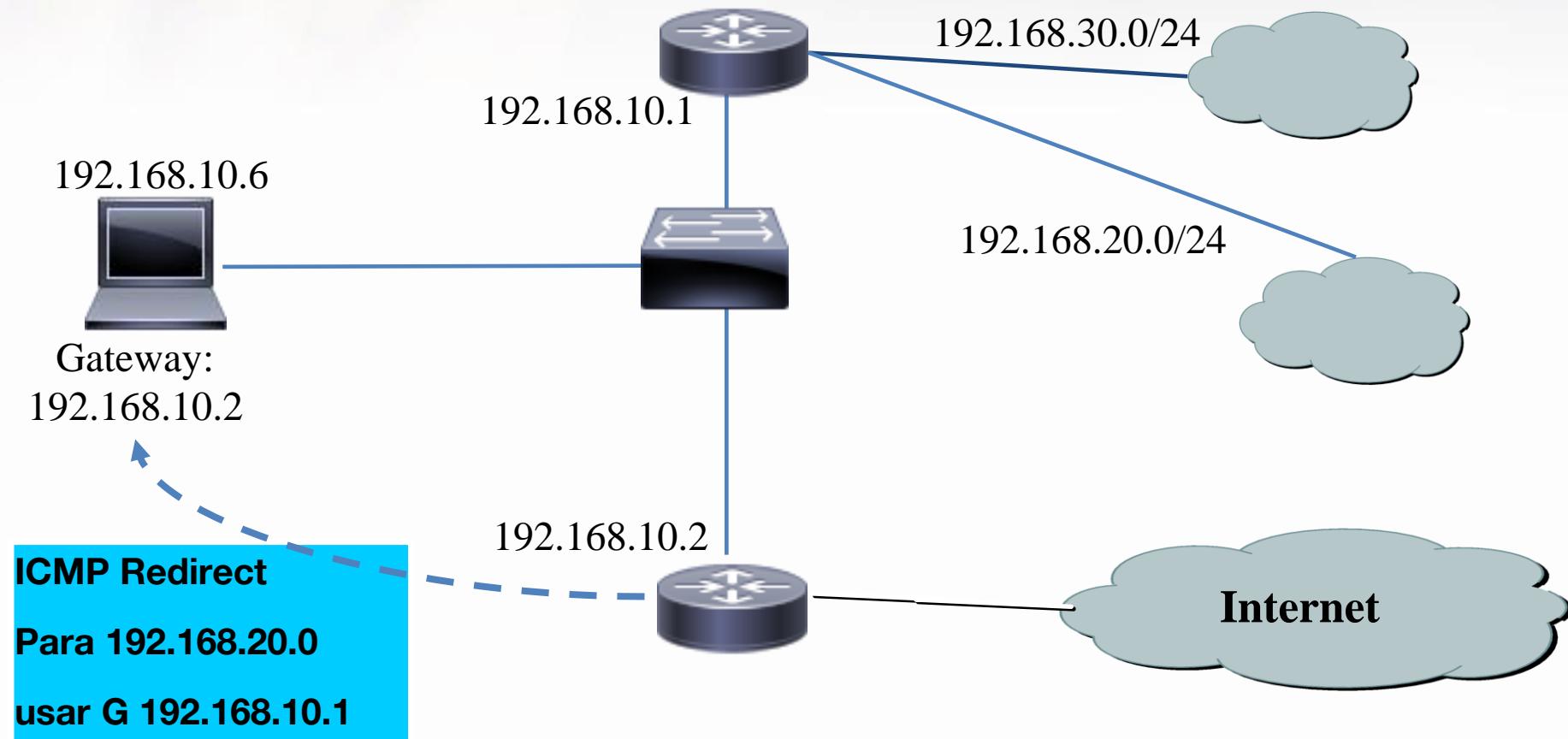
# ICMP: redirect

Las tablas de ruteo también se pueden actualizar en base a paquetes ICMP Redirect.



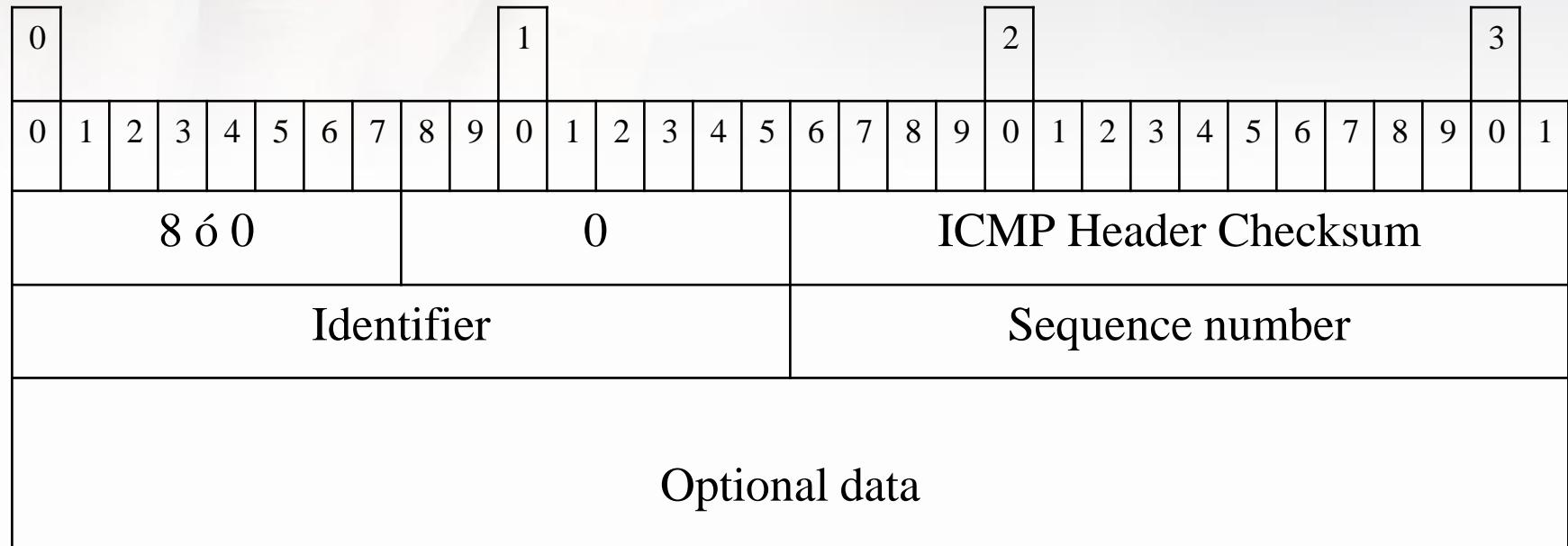
# ICMP: redirect

Las tablas de ruteo también se pueden actualizar en base a paquetes ICMP Redirect.



# Utilidades básicas de networking

**ping *hostDestino*:** Utiliza mensajes ICMP tipo 8 y 0



Si *hostDestino* es alcanzable, mostrará un resumen de paquetes enviados, recibidos, perdidos y tiempo medio de transferencia.

# Utilidades básicas de networking

```
user@server:~$ ping 200.49.213.50
```

```
PING 200.49.213.50 (200.49.213.50) 56(84) bytes of data.  
64 bytes from 200.49.213.50: icmp_seq=1 ttl=55 time=21.4 ms  
64 bytes from 200.49.213.50: icmp_seq=2 ttl=55 time=37.2 ms  
64 bytes from 200.49.213.50: icmp_seq=3 ttl=55 time=35.7 ms  
64 bytes from 200.49.213.50: icmp_seq=4 ttl=55 time=25.2 ms  
64 bytes from 200.49.213.50: icmp_seq=5 ttl=55 time=23.2 ms  
  
--- 200.49.213.50 ping statistics ---  
5 packets transmitted, 5 received, 0% packet loss, time 4036ms  
rtt min/avg/max/mdev = 21.424/28.601/37.273/6.594 ms
```

# Utilidades básicas de networking

```
user@server:~$ ping -c1 200.49.213.50 -r
PING 200.49.213.50 (200.49.213.50) 56(124) bytes of data.
64 bytes from 200.49.213.50: icmp_seq=1 ttl=56 time=2286 ms
RR:      192.168.2.10          209.13.133.250
          200.26.75.69          209.13.133.57
          200.51.241.170         200.51.240.82
          200.51.241.161         200.51.241.57
          172.30.248.46

--- 200.49.213.50 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 2286.637/2286.637/2286.637/0.000 ms
```

# Utilidades básicas de networking

En caso de tener problemas de conectividad, el comando **ping** nos puede ayudar a localizar dónde se encuentra el problema.

1. ping 127.0.0.1
2. ping *direccion\_IP\_local*
3. ping *host\_en\_mi\_segmento*
4. ping *host\_en\_otro\_segmento*
5. ping *host\_fuera\_de\_la\_red*

Tener en cuenta que muchos routers o hosts no responden solicitudes de eco.

# Utilidades básicas de networking

## traceroute

Al igual que **ping**, trata de alcanzar un host destino, pero informa cada router por el que va pasando hasta llegar a destino.

Lo que hace en realidad es enviar mensajes **ICMP echo request** (igual que ping) pero primero con un tiempo de vida de 1 salto, luego de 2 saltos, 3 saltos, etc, con un máximo por defecto de 30 saltos.

Especialmente útil en dos casos:

- ② Detectar dónde “muere” un paquete
- ② Detectar dónde un paquete se “empantana”

# Traceroute: demoras reales

```
user@server:~$ traceroute campus.itba.edu.ar
```

```
Tracing route to campus.itba.edu.ar [192.230.225.58]
```

```
 1 <1 ms <1 ms <1 ms 192.168.1.1
 2 * * * Request timed out.
...
 6 15 ms 15 ms 15 ms 129-161-89-200.fibertel.com.ar [200.89.161.129]
 7 13 ms 15 ms 16 ms 130-165-89-200.fibertel.com.ar [200.89.165.130]
 8 13 ms 14 ms 11 ms 222-165-89-200.fibertel.com.ar [200.89.165.222]
 9 * * * Request timed out.
10 139 ms 140 ms 138 ms ae1-300G.ar5.MIA1.gblx.net [67.17.94.249]
11 * * * Request timed out.
12 * * * Request timed out.
13 160 ms 163 ms 159 ms BLACKBOARD.ear2.Washington1.Level3.net
[4.59.144.218]
14 165 ms 159 ms 164 ms eth4-1-bb1-dc2.blackboard.com [69.196.255.133]
15 159 ms 169 ms 160 ms eth4-3-bb1-va2.mhint [69.196.255.145]
16 166 ms 174 ms 175 ms eth3-3-gw1-va2.mhint [69.196.255.158]
17 158 ms 159 ms 159 ms itba.blackboard.com [192.230.225.58]
```

# IPv6



- ◆ Direcciones de 128 bits
- ◆ Estructura de direcciones jerárquica
- ◆ Número de MAC como parte de la dirección IP
- ◆ Incorpora seguridad y multicasting
- ◆ Encabezado de tamaño fijo, pensado para hacer más eficiente su procesamiento por los routers
- ◆ Facilita QoS
- ◆ No permite fragmentación
  
- ◆ El protocolo IPv6 se define en RFC 2460 ( Dic. 1998 )

# Estadísticas de uso

- <https://www.internetsociety.org/deploy360/ipv6/statistics/>
  - [IPv6 – Google](#)
  - [IPv6 Measurement Maps](#)

# Prefijos IPv6



TIPO DE DIRECCIÓN	BITS DE PREFIJO	PREFIJO
Link-Local	1111 1110 10	<b>FE80::/10</b>
Unspecified	0000 ... 0 (128 bits)	<b>::/128</b>
Loopback	0000 ... 01 (128 bits)	<b>::1/128</b>
Multicast	1111 1111	<b>FF00::/8</b>
IPv4-Mapped	000 ... 011111111111 (96 bits)	<b>::FFFF/96</b>
ULA	1111 110	<b>FC00::/7</b>
Global Unicast	001	<b>2000::/3</b>
Anycast		

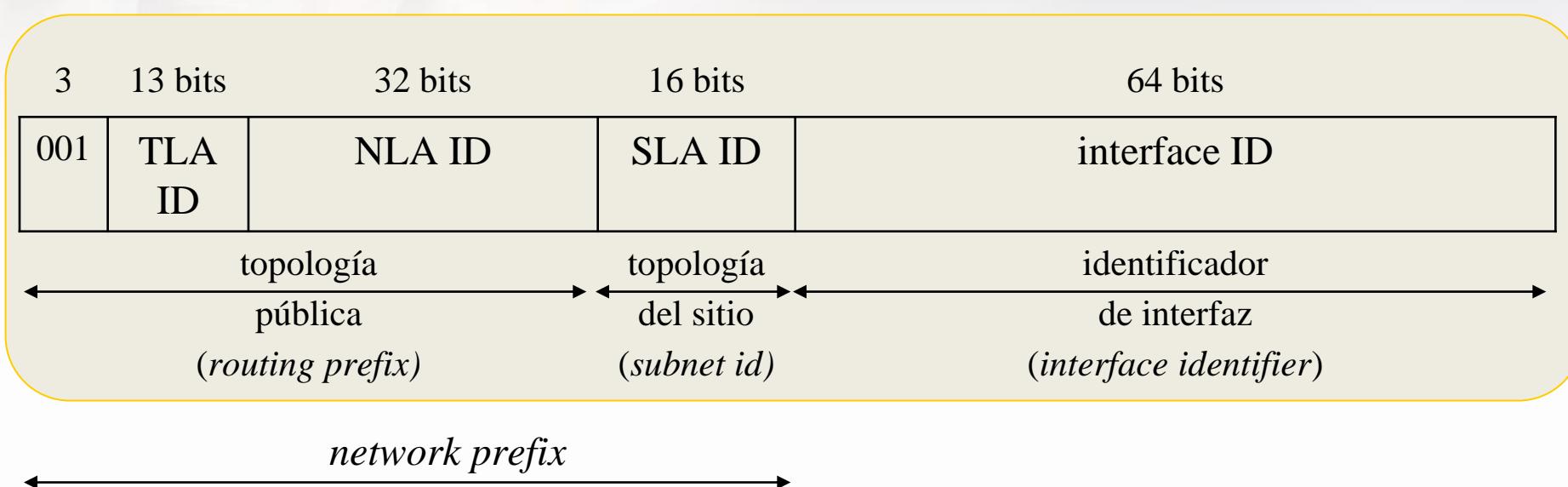
# IPv6



- ◆ IPv4-compatible:
  - ◆ 0:0:0:0:FFFF:192.168.30.1
  - ◆ = ::FFFF:192.168.30.1
  - ◆ = ::FFFF:C0A8:1E01
- ◆ Como URL, se la encierra entre corchetes
  - ◆ [http://\[2001:1:4F3A::206:AE14\]:8080/index.html](http://[2001:1:4F3A::206:AE14]:8080/index.html)
- ◆ No existe IP broadcast

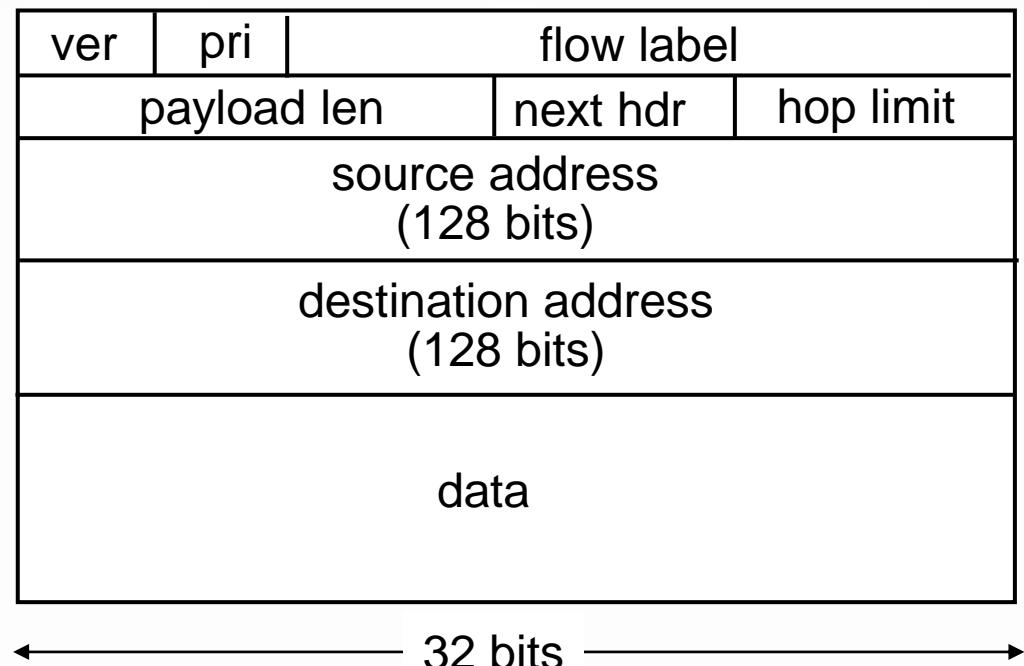


## Direcciones Global Unicast (RFC 3587)



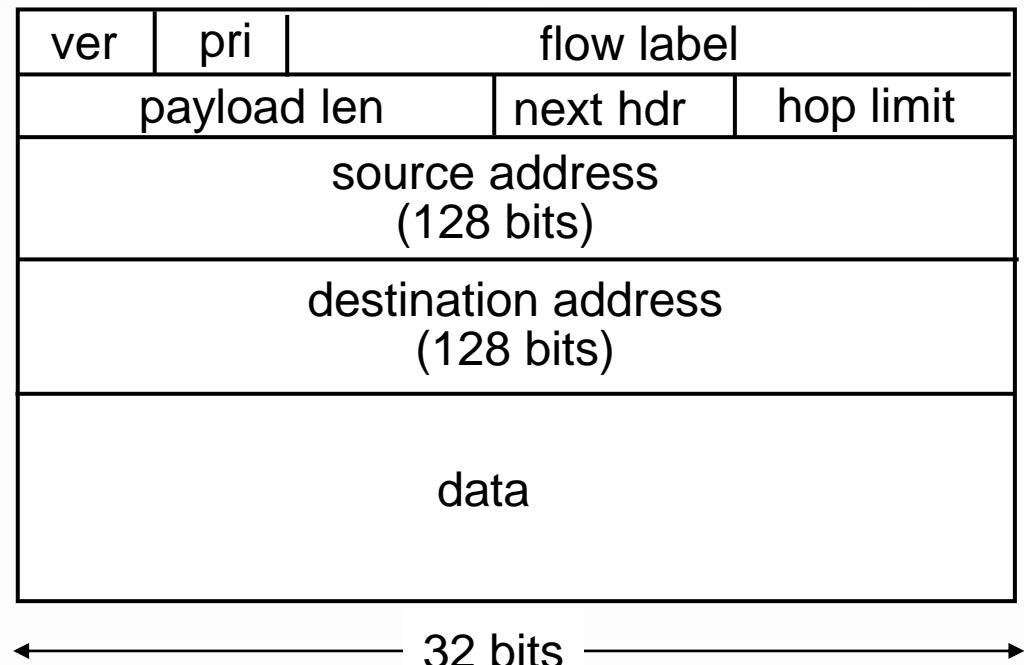
# IPv6: formato

- ◆ Priority: prioridad del paquete dentro del flujo
- ◆ Flow label: identifica paquetes dentro de un mismo “flujo”
- ◆ Checksum: no está más

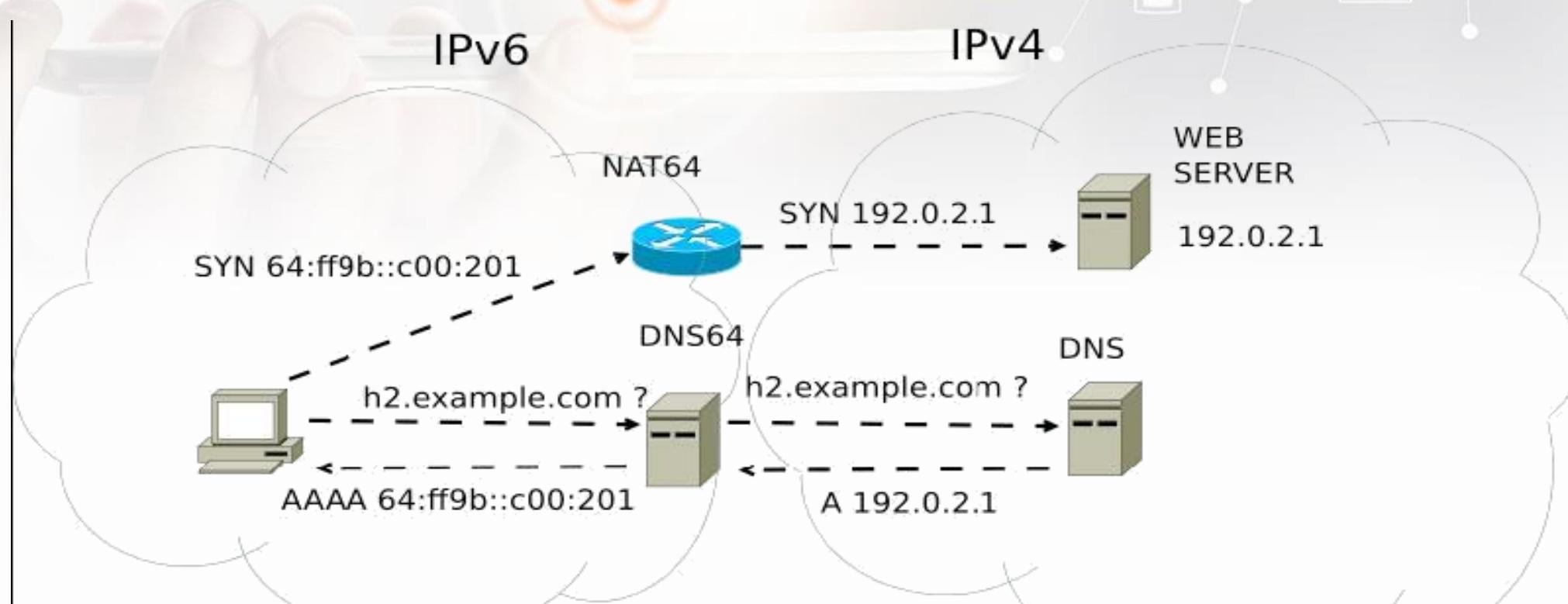


# IPv6: formato

- ◆ Payload len: longitud en octetos a continuación del header (hasta 64KB)
- ◆ Next header: similar a tipo de protocolo en IPv4
- ◆ Hop limit: similar a TTL en IPv4



# NAT64: conectando IPv6 con IPv4

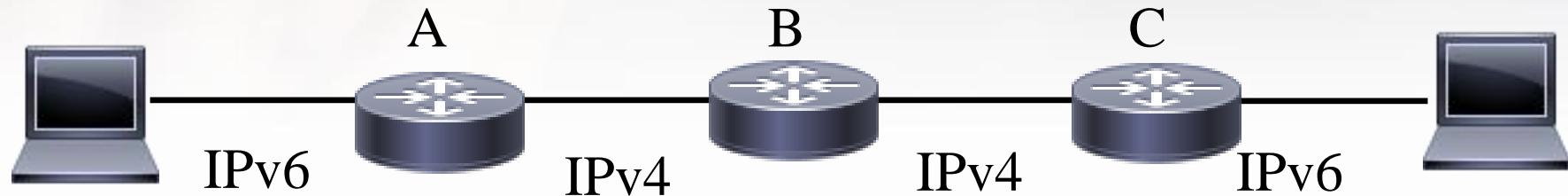


**NAT64** se encarga de la traducción de las direcciones entre hosts que sólo tienen conectividad IPv6 con hosts que sólo tienen IPv4. Tras la resolución del nombre por el DNS64, se pide la dirección web a través de NAT64.

# IPv6



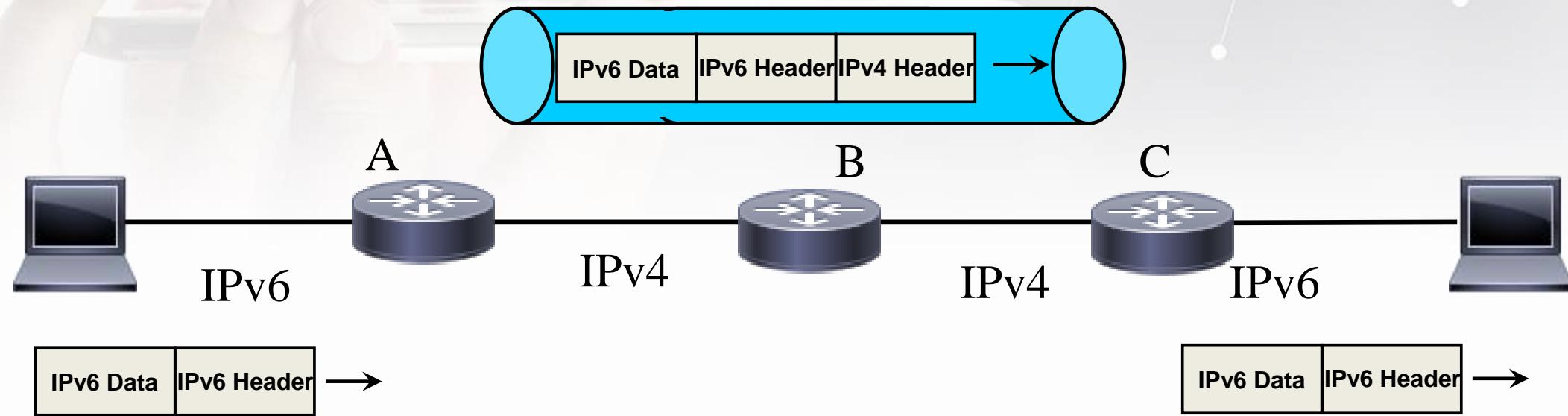
¿Qué sucede si una red intermedia sólo entiende IPv4?



El Router A debe "encapsular" el paquete Ipv6 dentro de un paquete IPv4

*Tunneling:* consiste en encapsular datos de un protocolo en los datos de otro protocolo de igual o mayor nivel.

# IPv6 over IPv4



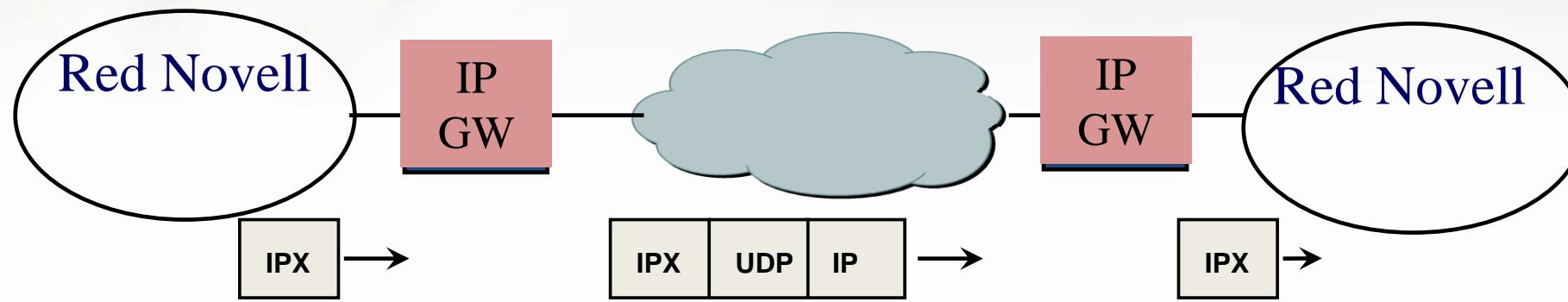
# Túneles

Supongamos que existen dos LAN Novell distantes que queremos conectar a través de Internet.



# Túneles

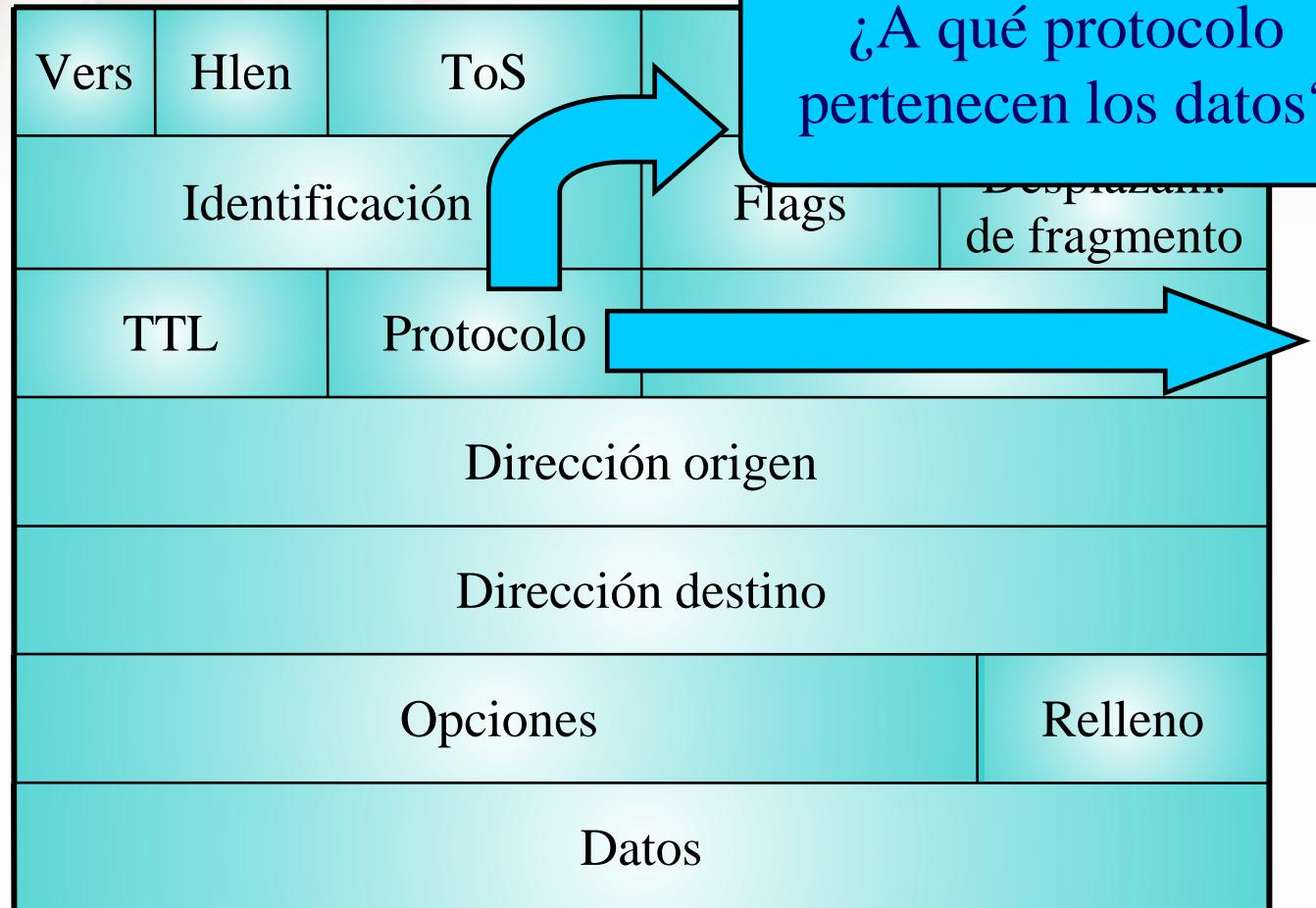
Solución: Encapsular los paquetes IPX en paquetes UDP



# Túneles IP-en-IP

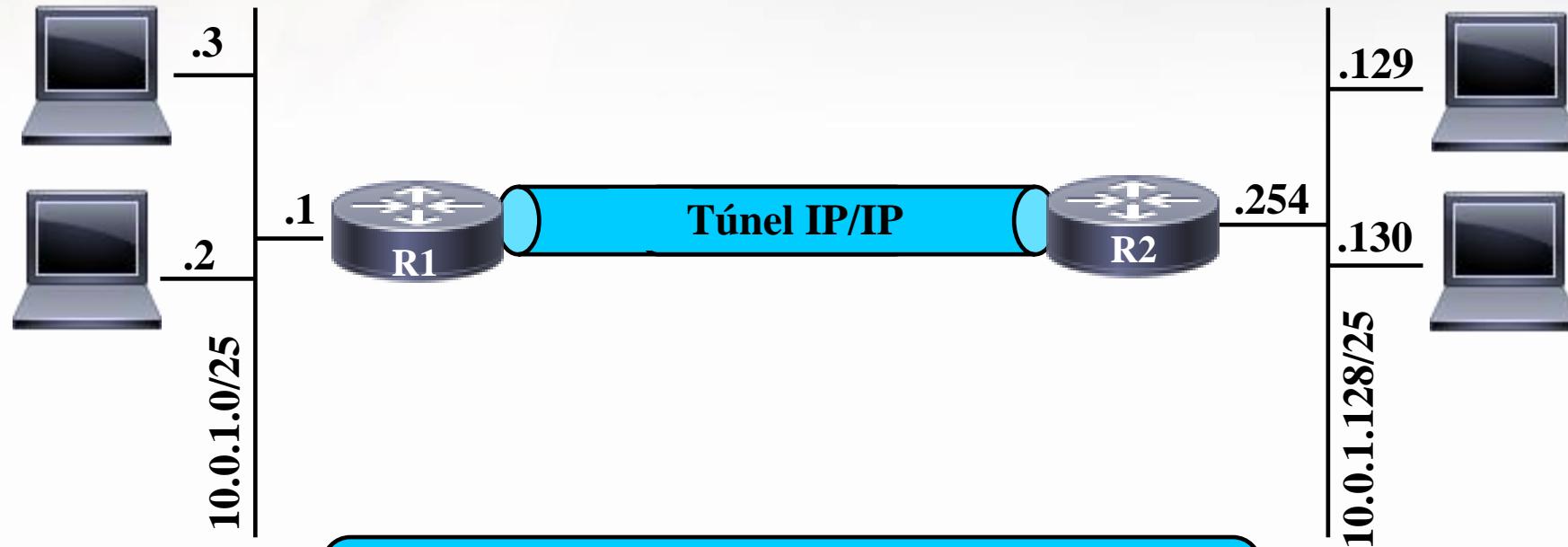


Recordemos el paquete IP



# Túneles IP-en-IP

Supongamos que tenemos dos redes IP y queremos enviar tráfico desde una hacia otra como si estuvieran conectadas directamente.



¿Podría resolverlo utilizando NAT?

# Túneles IP-en-IP



## Cómo configurar el túnel en Linux

En R1 (IP privada: 10.0.1.1, IP pública: 24.10.11.12)

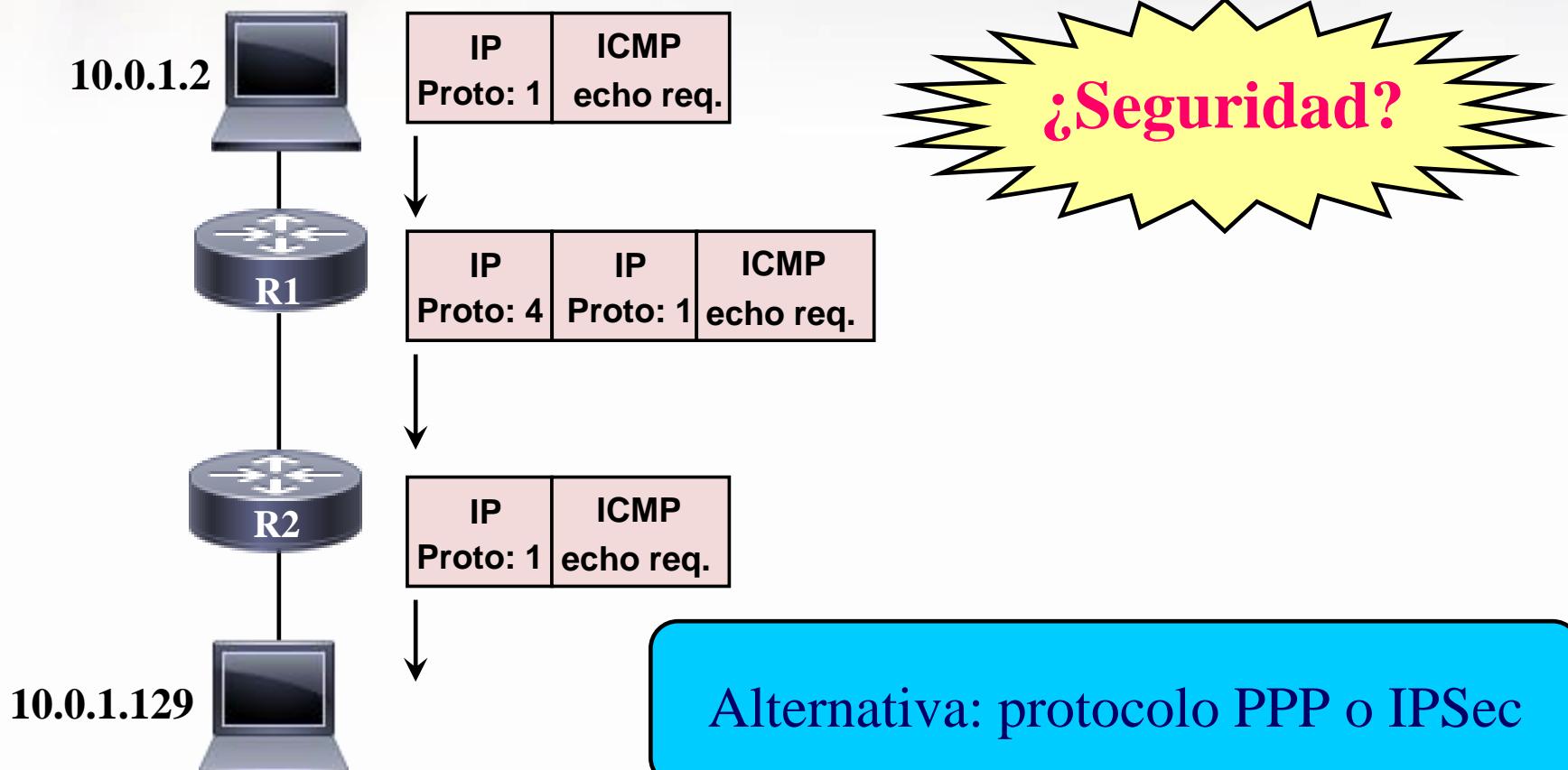
```
$ ifconfig tun0 10.0.1.1 pointopoint 24.13.14.15  
$ route add -net 10.0.1.128 netmask 255.255.255.128 dev tun0
```

En R2 (IP privada: 10.0.1.254, IP pública: 24.13.14.15)

```
$ ifconfig tun0 10.0.1.254 pointopoint 24.10.11.12  
$ route add -net 10.0.1.0 netmask 255.255.255.128 dev tun0
```

# Túneles IP-en-IP

¿Cuál sería el flujo de paquetes si el host 10.0.1.2 envía un ping al 10.0.1.129?



# Material de lectura

Capítulos 4.1 a 4.4 inclusive de la bibliografía

# IPSec



El objetivo de IPSec es proveer servicios de seguridad para los paquetes IP en el nivel de red.

□ Estos servicios incluyen:

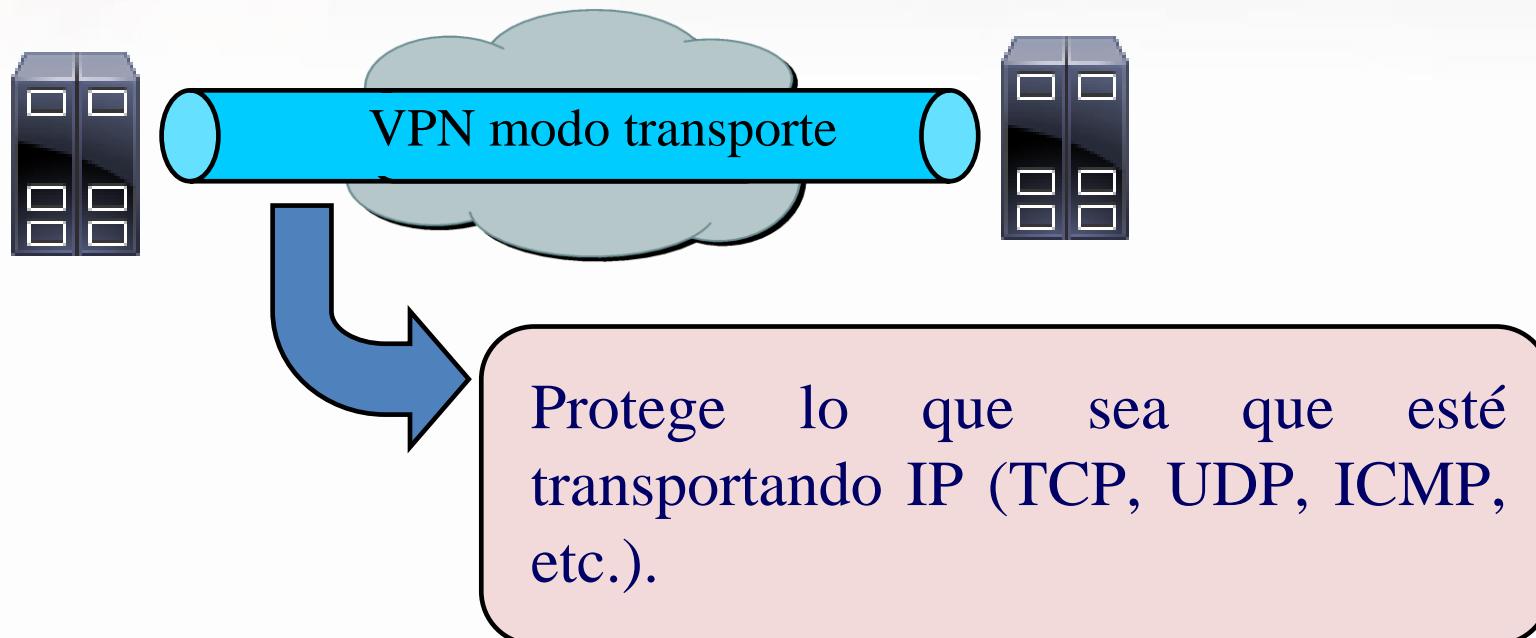
- Control de acceso
- Integridad de datos
- Autenticación
- Confidencialidad de datos

□ IPSec trabaja en dos modos:

- Túnel
- Transporte

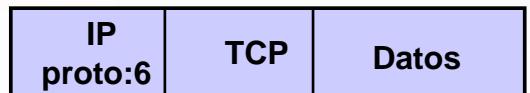
# IPSec: modo transporte

Es usado entre dos hosts en forma directa. No puede ser usado para conectar dos redes o un host a una red.



# IPSec: modo transporte

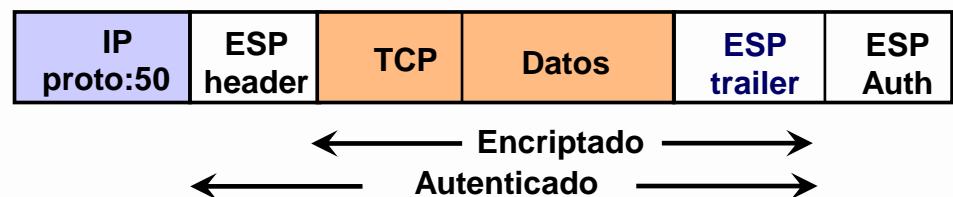
En este modo se agrega un header AH (*Authentication Header*) o ESP (*Encapsulating Security Payload*)



Paquete original



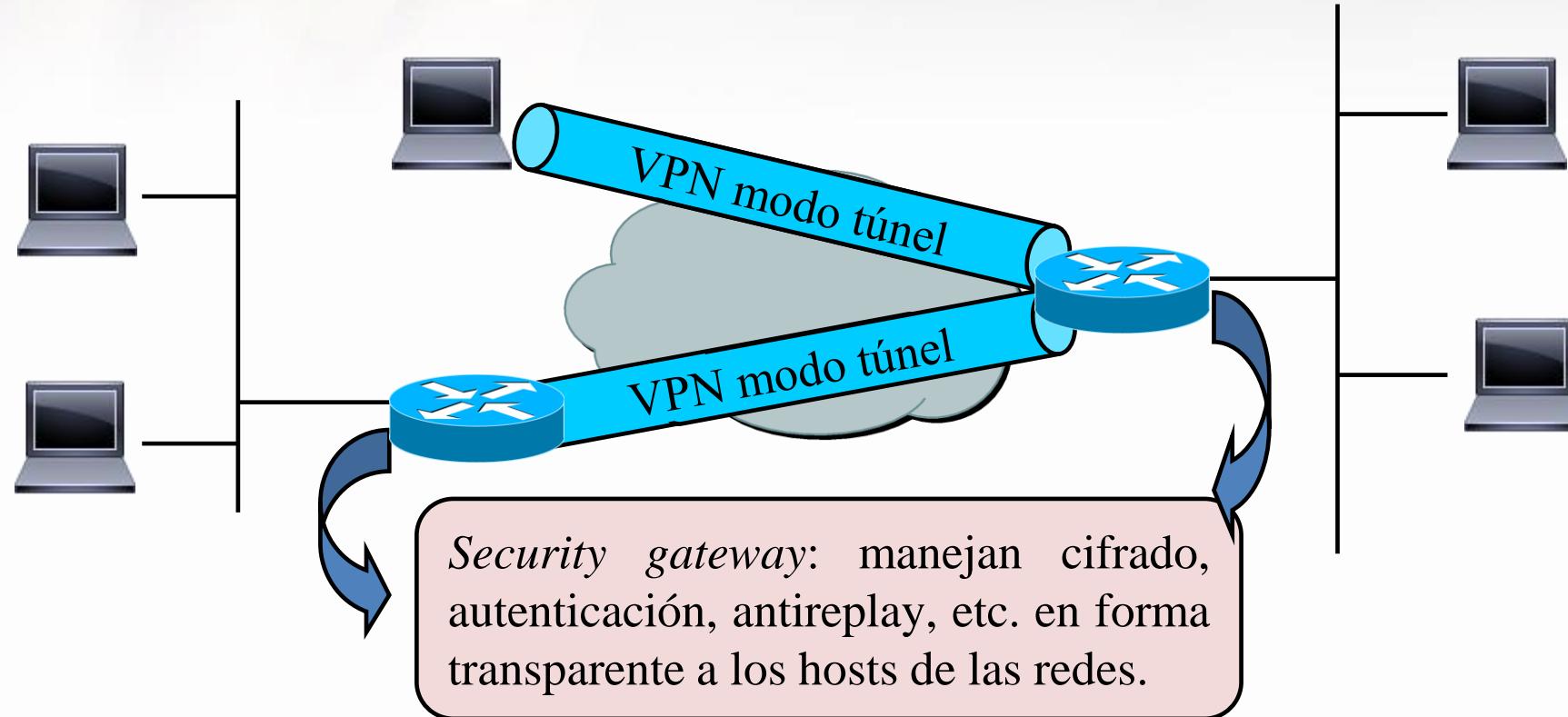
Autenticado



Autenticado y  
encriptado

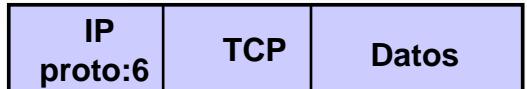
# IPSec: modo túnel

En modo túnel es posible formar una VPN con dos redes o un host y una red.

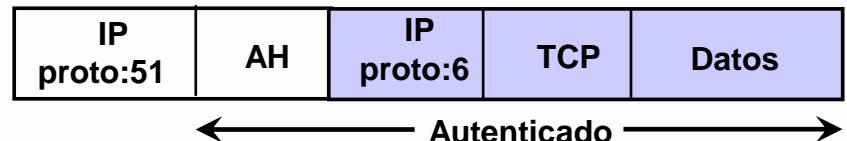


# IPSec: modo túnel

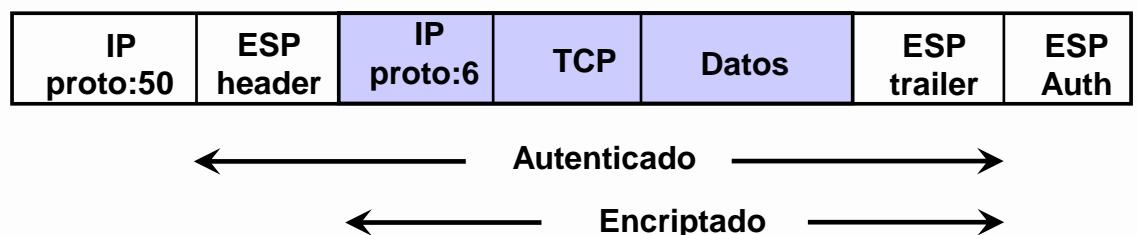
El paquete IP original es encapsulado en otro datagrama IP, insertando un header (AH o ESP)



Paquete original



Autenticado



Autenticado y  
encriptado

# IpTables e Ip6Tables

- Filtrado de paquetes y NAT
- Configurar, mantener e inspeccionar las tablas usadas por el kernel para filtrar paquetes
- Un paquete va pasando por distintas tablas, las cuales a su vez contienen una o más listas

# IPTables



Hay 5 tablas independientes que utiliza ipTables:

- ❑ **raw**: filtra los paquetes antes que otras tablas. Se utiliza principalmente para configurar exenciones de seguimiento de conexiones. Dos cadenas:
  - ❑ PREROUTING:
  - ❑ OUTPUT
- ❑ **filter**: es la usada por defecto. Cadenas:
  - ❑ INPUT: tráfico entrante
  - ❑ OUTPUT: tráfico saliente generado localmente
  - ❑ FORWARD: tráfico enrutado

```
~$ iptables -A INPUT -p tcp --dport 17500 -j REJECT  
--reject-with icmp-port-unreachable
```

# IPTables

Hay 5 tablas independientes que utiliza ipTables:

- ❑ **nat**: es consultada cuando un paquete crea una nueva conexión. Se usa para traducir direcciones de red y puertos. No se debe usar para reglas de filtrado. Cadenas:
  - ❑ PREROUTING: cambia paquetes antes de ser “ruteados”
  - ❑ POSTROUTING: cambia paquetes luego de ser “ruteados”
  - ❑ OUTPUT: aplica NAT a paquetes generados localmente
- ❑ Acciones:
  - ❑ DNAT
  - ❑ SNAT
  - ❑ MASQUERADE
  - ❑ REDIRECT

# Tabla nat



Utilizada para traducir el IP origen o destino de una secuencia de paquetes.

Posibles *targets*:

- ◆ DNAT
- ◆ SNAT
- ◆ MASQUERADE

```
~$ iptables -t nat -A PREROUTING -p tcp --dport 80  
      -i eth1 -j DNAT --to 10.0.1.80:8080
```

# IpTables: Ejemplos

Listar reglas y ver tabla NAT actual

```
~$ iptables -t nat -L  
~$ netstat-nat -n
```

Aplicar SNAT (eth1: interfaz WAN)

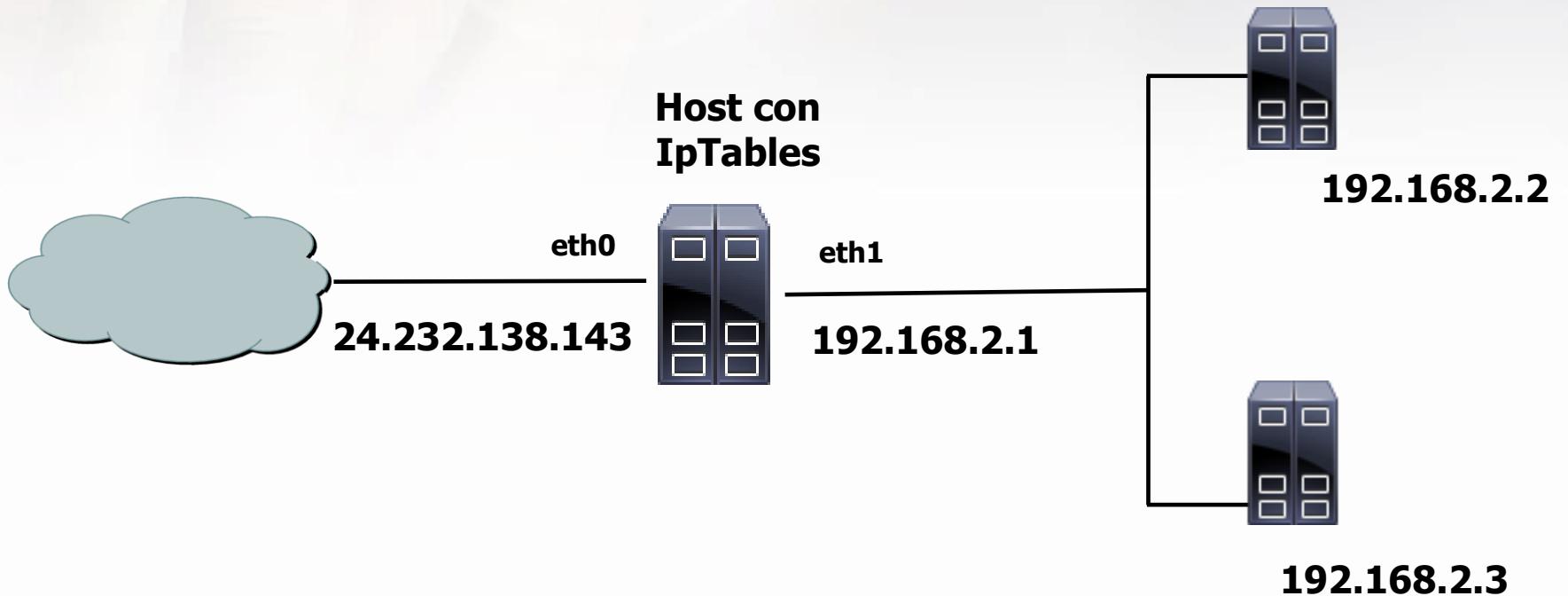
```
~$ iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE
```

```
~$ iptables -t nat -A POSTROUTING -s 192.168.1.0/24  
-o eth1 -j SNAT --to 20.28.101.14
```

```
~$ iptables -t nat -A POSTROUTING -p tcp  
-o eth1 -j SNAT --to 20.28.101.14:1-1023
```

# Ejemplo

Sea el siguiente escenario



# Ejemplo

```
# Reenviar todos los paquetes de eth1 (red interna)
# a eth0 (Internet).
iptables -A FORWARD -i eth1 -o eth0 -j ACCEPT

# Habilitar SNAT para IP pública fija:
iptables -t nat -A POSTROUTING -o eth0 -j SNAT
           -to-source 24.232.138.143

# No aceptar IP Spoofing
iptables -A INPUT -i eth0 -s 24.232.138.143/32 -j DROP
iptables -A INPUT -i eth0 -s 192.168.0.0/24      -j DROP
iptables -A INPUT -i eth0 -s loopback/8 -j DROP
```

# Ejemplo

```
# Aceptamos lo que venga de localhost
iptables -A INPUT -i lo -j ACCEPT

# Implementamos Web Server en un host local
iptables -t nat -A PREROUTING -d 24.232.138.143
-p tcp -dport 80 -j DNAT
--to-destination 192.168.2.3
```



## Protocolos de routing

- ◆ Routing estático
- ◆ Routing dinámico
- ◆ RIP
- ◆ OSPF

# Protocolos de Routing

Routing permite que el tráfico de una red pueda ser enviado hacia otra red en cualquier parte del mundo. Hay que diferenciar entre:

- **Routing o forwarding:** es el acto de reenviar paquetes a un host basado en la información almacenada en las tablas de ruteo.
- **Protocolos de routing:** son programas que intercambian información entre routers para construir las tablas de ruteo.

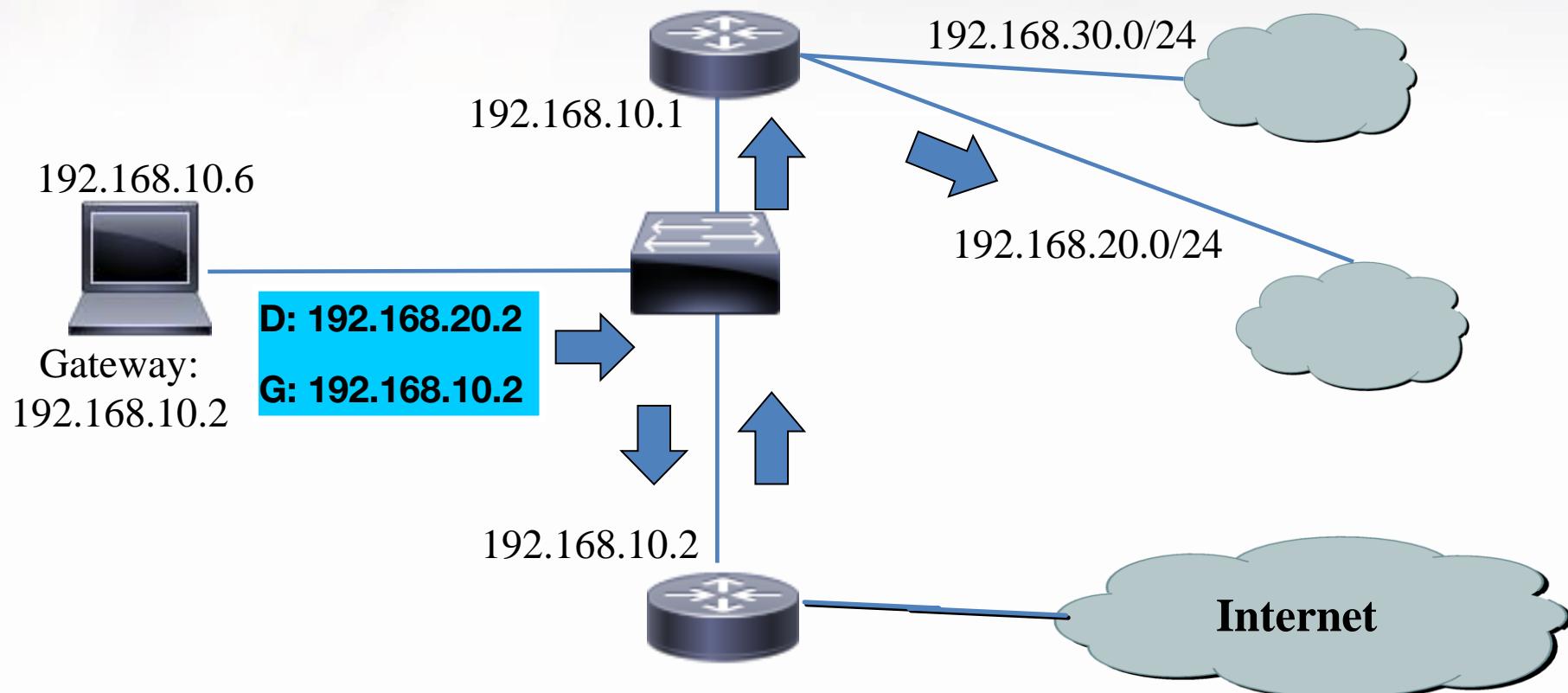
# Protocolos de Routing

Las configuraciones de routing más comunes son:

- ◆ **Routing mínimo:** una red aislada de otras redes TCP/IP
- ◆ **Routing estático:** la tabla de ruteo es construída por un administrador en forma estática.
- ◆ **Routing dinámico:** la tabla de ruteo es construída en base a información intercambiada por protocolos de routing.

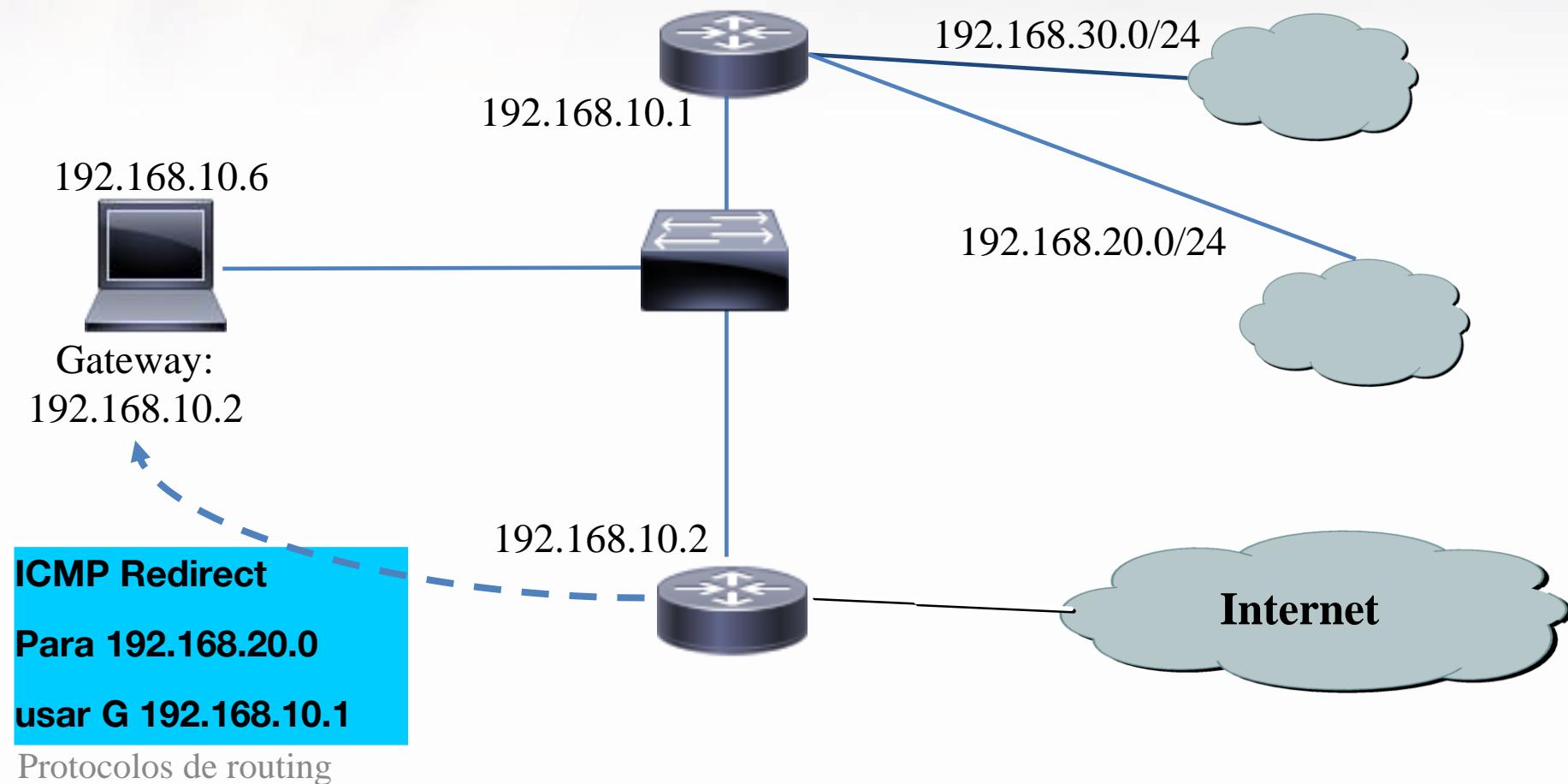
# Protocolos de Routing

Las tablas de ruteo también se pueden actualizar en base a paquetes **ICMP Redirect**.



# Protocolos de Routing

Las tablas de ruteo también se pueden actualizar en base a paquetes **ICMP Redirect**.



# ICMP redirect



Formato de un paquete ICMP Redirect

Type=5	Code	Checksum
Gateway Internet address (192.168.10.1)		
Internet header + 64 bits Data		

- C      0 = Redirect datagrams for the Network.  
O      1 = Redirect datagrams for the Host.  
D      2 = Redirect datagrams for the ToS and Network.  
E      3 = Redirect datagrams for the ToS and Host.

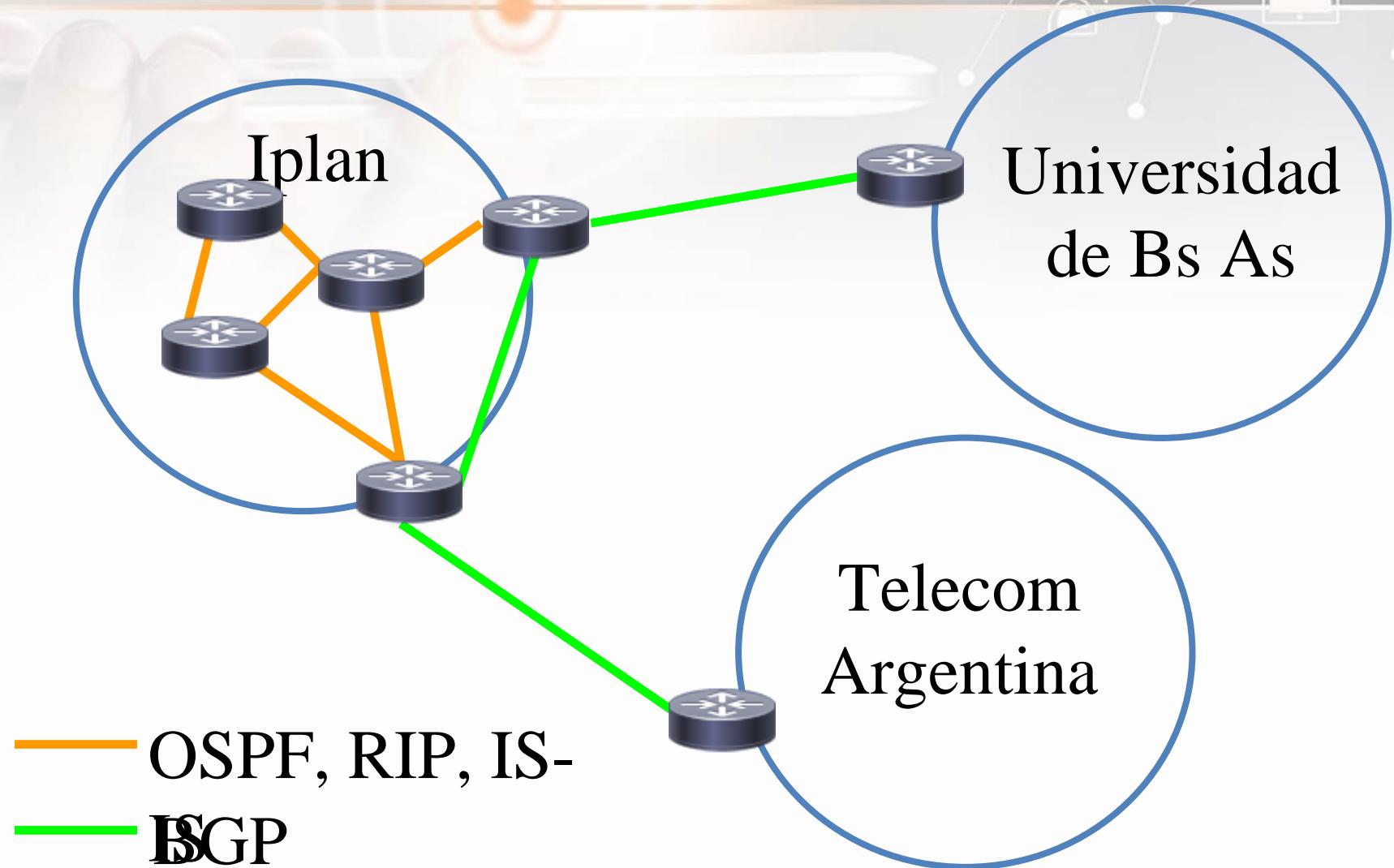


# Routing dinámico

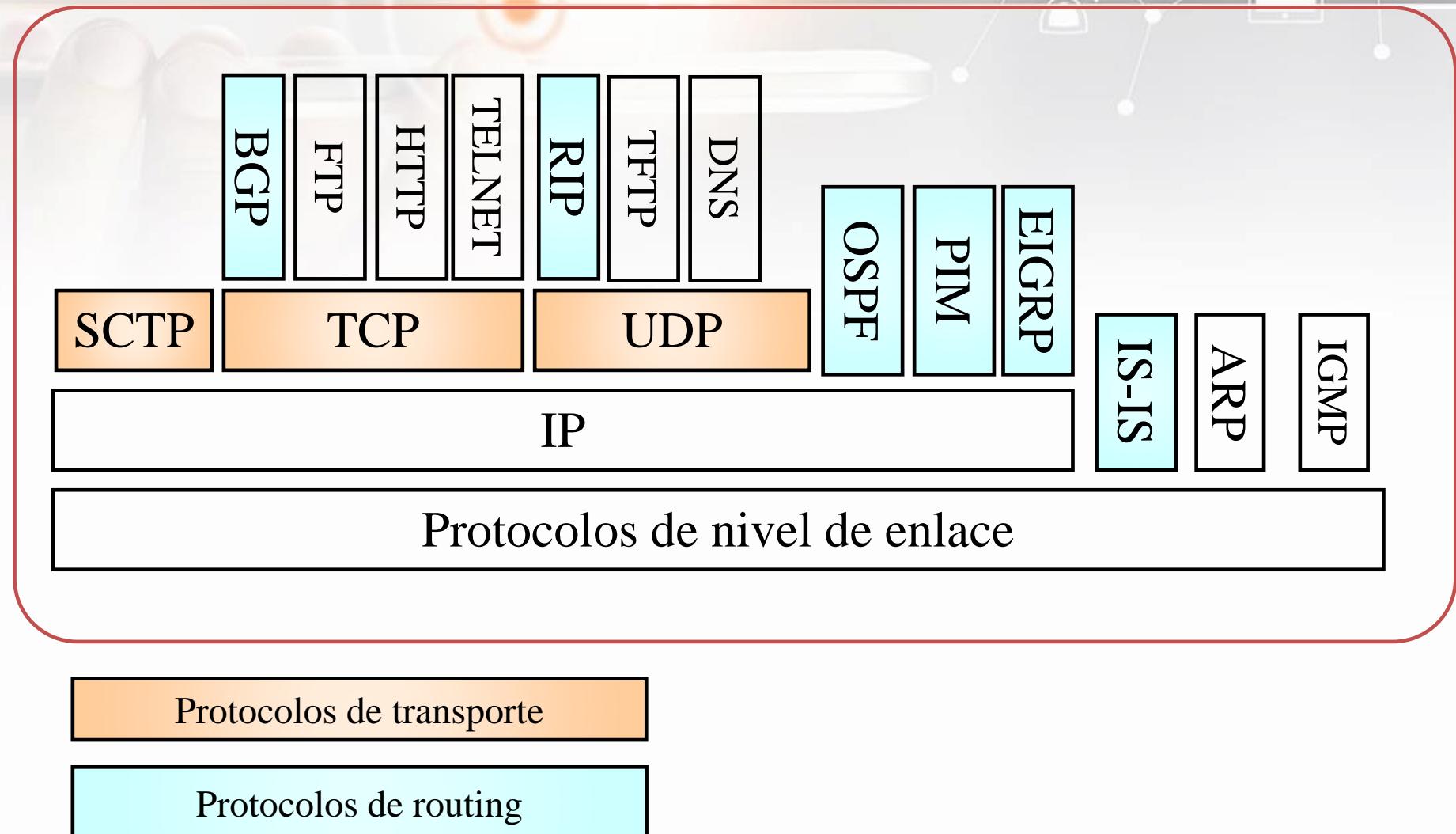
Se dividen en dos categorías:

- **Routing interno (IGP)** : diseñados para una red controlada por una sola organización. Los más usados son RIP y OSPF.
- **Routing externo (EGP)** : diseñados para intercambiar información entre redes controladas por distintas organizaciones (sistemas autónomos). Se usa BGP.

# Routing dinámico



# Routing dinámico



# Routing dinámico

Utiliza algoritmos adaptativos

- Vector de distancia (*Distance vector*)
- Estado de enlace (*Link state*)

# Distance vector

Cada router recibe de sus vecinos una tabla de enrutamiento.  
En base a ella actualiza su tabla y la envía a sus vecinos.

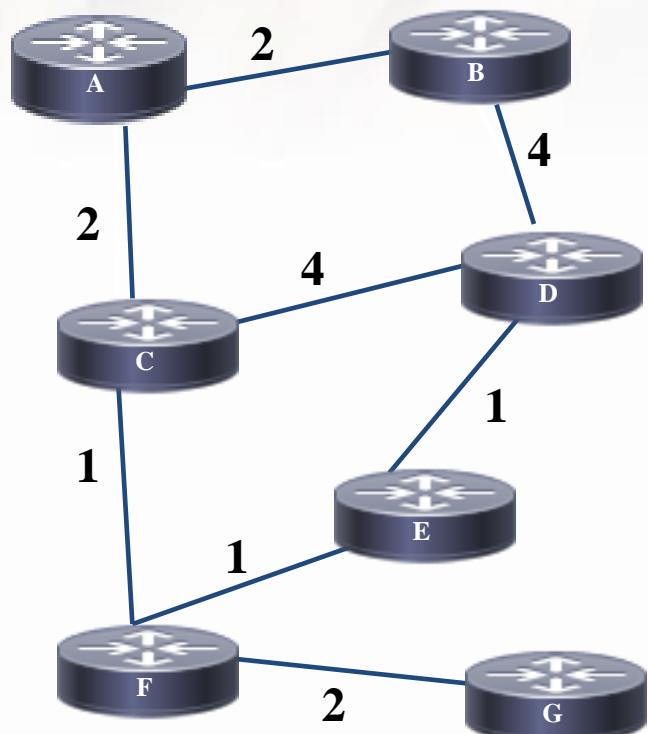


Tabla de A

R	D	Sale
A	0	-
B	2	B
C	2	C

Informa B

R	D
A	2
B	0
D	4

Informa C

R	D
A	2
C	0
D	4
F	1

# Distance vector

Cada router recibe de sus vecinos una tabla de enrutamiento.  
En base a ella actualiza su tabla y la envía a sus vecinos.

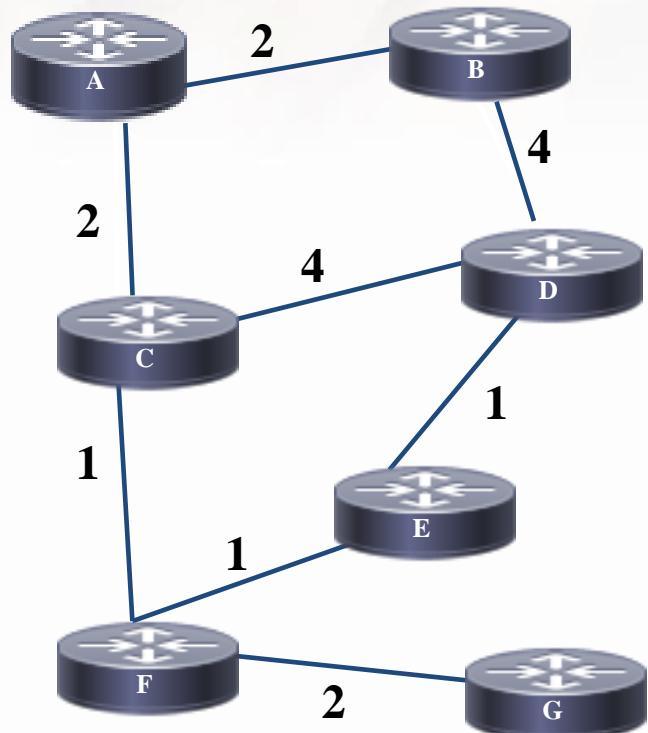


Tabla de A

R	D	Sale
A	0	-
B	2	B
C	2	C
D	6	B
F	3	C

Informa B

R	D
A	2
B	0
C	4
D	4
E	5

Informa C

R	D
A	2
B	4
C	0
D	4
E	2
F	1
G	3

# Distance vector

Cada router recibe de sus vecinos una tabla de enrutamiento.  
En base a ella actualiza su tabla y la envía a sus vecinos.

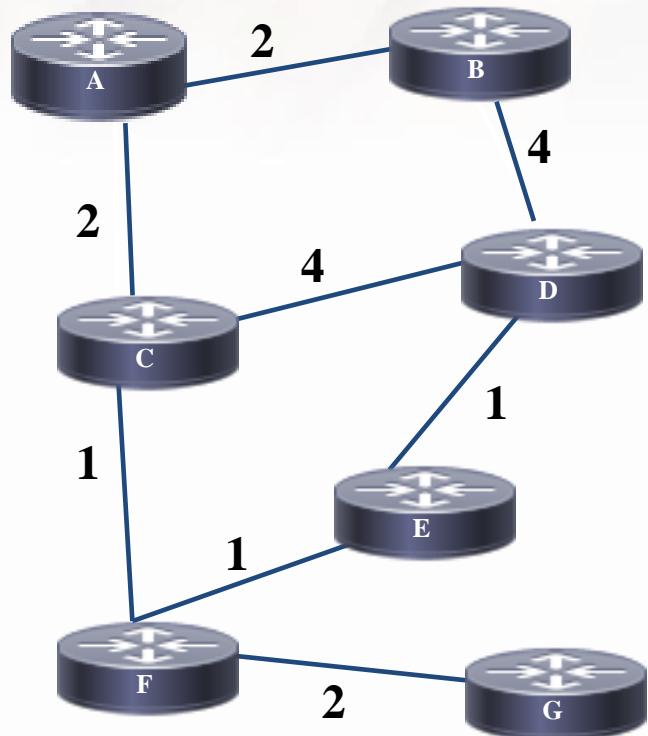


Tabla de A

R	D	Sale
A	0	-
B	2	B
C	2	C
D	6	B
E	4	C
F	3	C
G	5	C

Informa B

R	D
A	2
B	0
C	4
D	4
E	5
F	5

Informa C

R	D
A	2
B	4
C	0
D	3
E	2
F	1
G	3

# Distance vector

Cada router recibe de sus vecinos una tabla de enrutamiento.  
En base a ella actualiza su tabla y la envía a sus vecinos.

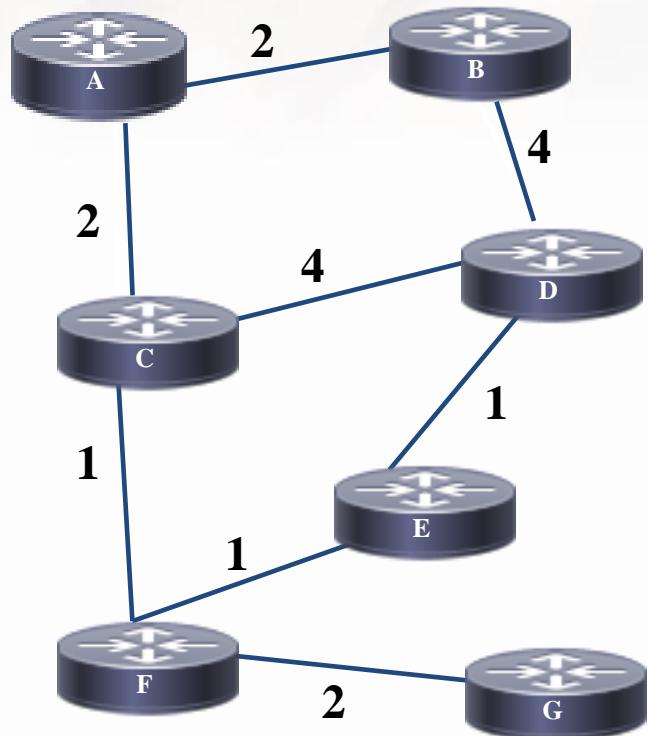


Tabla de A

R	D	Sale
A	0	-
B	2	B
C	2	C
D	5	C
E	4	C
F	3	C
G	5	C

Informa B

R	D
A	2
B	0
C	4
D	4
E	5
F	5
G	7

Informa C

R	D
A	2
B	4
C	0
D	3
E	2
F	1
G	3

# Distance vector



## Problemas: Contar hasta infinito

Supongamos que A estaba inactivo y vuelve a la actividad



Intercambios	Dist(B-A)	Dist(C-A)	Dist(D-A)	Dist(E-A)
0	$\infty$	$\infty$	$\infty$	$\infty$
1	1	$\infty$	$\infty$	$\infty$
2	1	2	$\infty$	$\infty$
3	1	2	3	$\infty$
4	1	2	3	4

Las buenas noticias viajan rápido.

# Distance vector

**Problemas: Contar hasta infinito**

Supongamos que A estaba activo y pierde conectividad con B



Intercambios				
		$\text{Dist}(C-A) + \text{Dist}(B-C)$		
		1	2	3
1	3	2	3	4
2	3	4	3	4
3	5	4	5	4
4	5	6	5	6
5	7	6	7	6
...		...	...	...

¡ Incrementa hasta infinito !

# Distance vector

## Problemas: Contar hasta infinito

Posibles soluciones:

- ◆ *Split horizon*: no enviarle a mi vecino lo que mi vecino me está enviando.
- ◆ Ruta envenenada: lo que mi vecino me envía se lo envío con salto  $\infty$ , para que no actualice.
- ◆ Enviar rápido las malas noticias: si un router detecta un problema en el vínculo, enviar en forma inmediata esa entrada “envenenada”.



## RIP (*Routing Information Protocol*)

- ◆ Pertenece a los algoritmos *distance vector*
- ◆ Selecciona una ruta en base a la cantidad de saltos a usar para llegar a destino.
- ◆ Simple de configurar.
- ◆ Adecuado para redes pequeñas
- ◆ El demonio en Linux es **routed**.
  - ◆ Cuando se inicia envía un pedido de actualizaciones de ruteo.
  - ◆ Cada router envía en forma periódica paquetes con información basada en su table de ruteo.

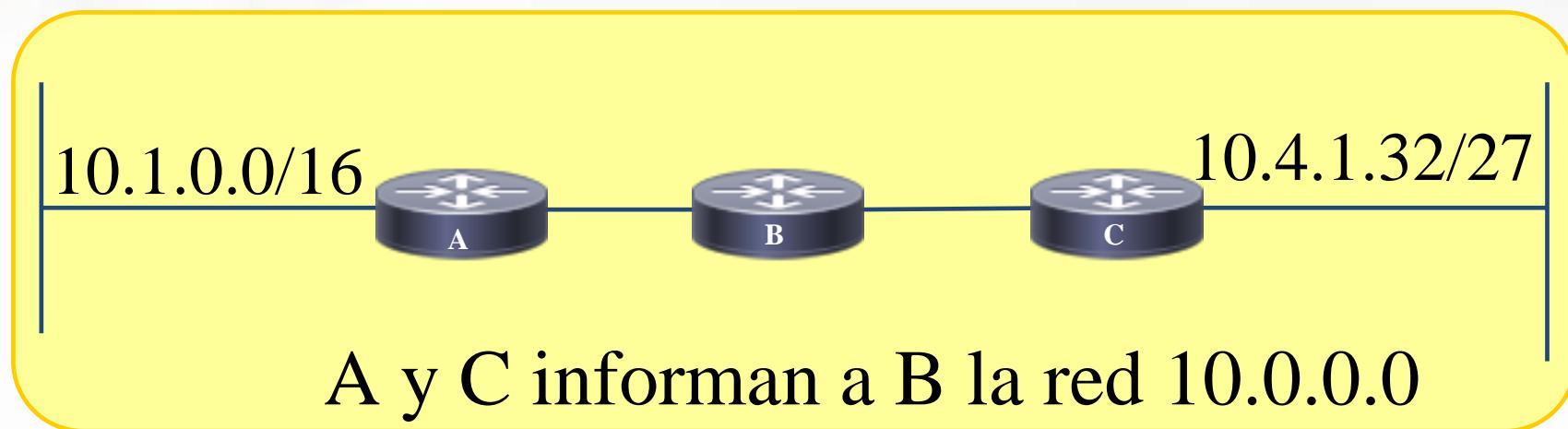
# RIP v1

## Características de RIP v1

- ◆ Intercambia información con sus vecinos cada 30 segundos
- ◆ Limitado a un diámetro de red de 15 saltos
- ◆ Los paquetes son broadcast
- ◆ Usa notación *classfull*

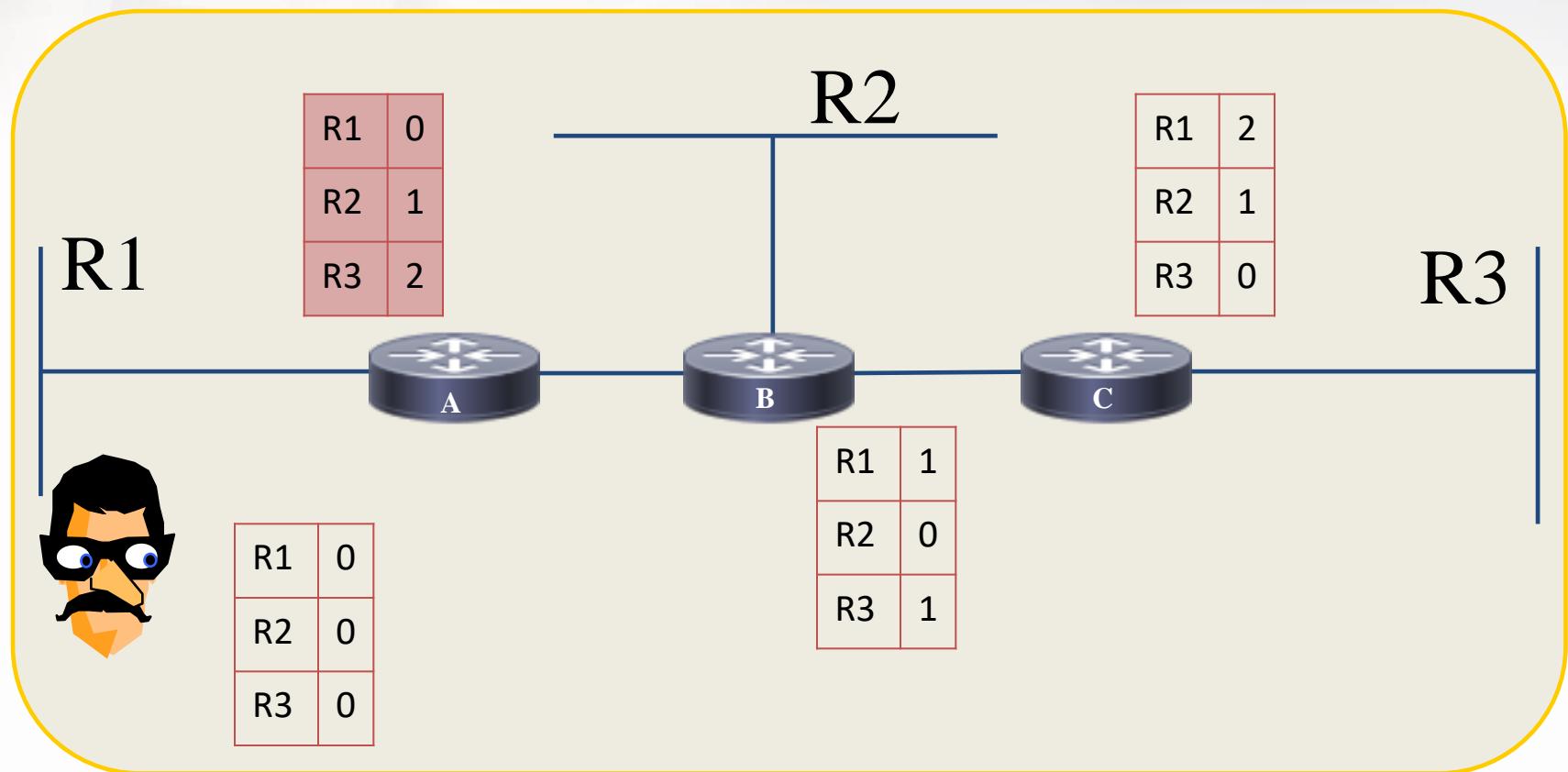
# RIP v1

Direcciones IP de red con máscara natural (*classful*)



# RIP v1

Los paquetes no tienen autenticación



# RIP v2 (RFC 2453)

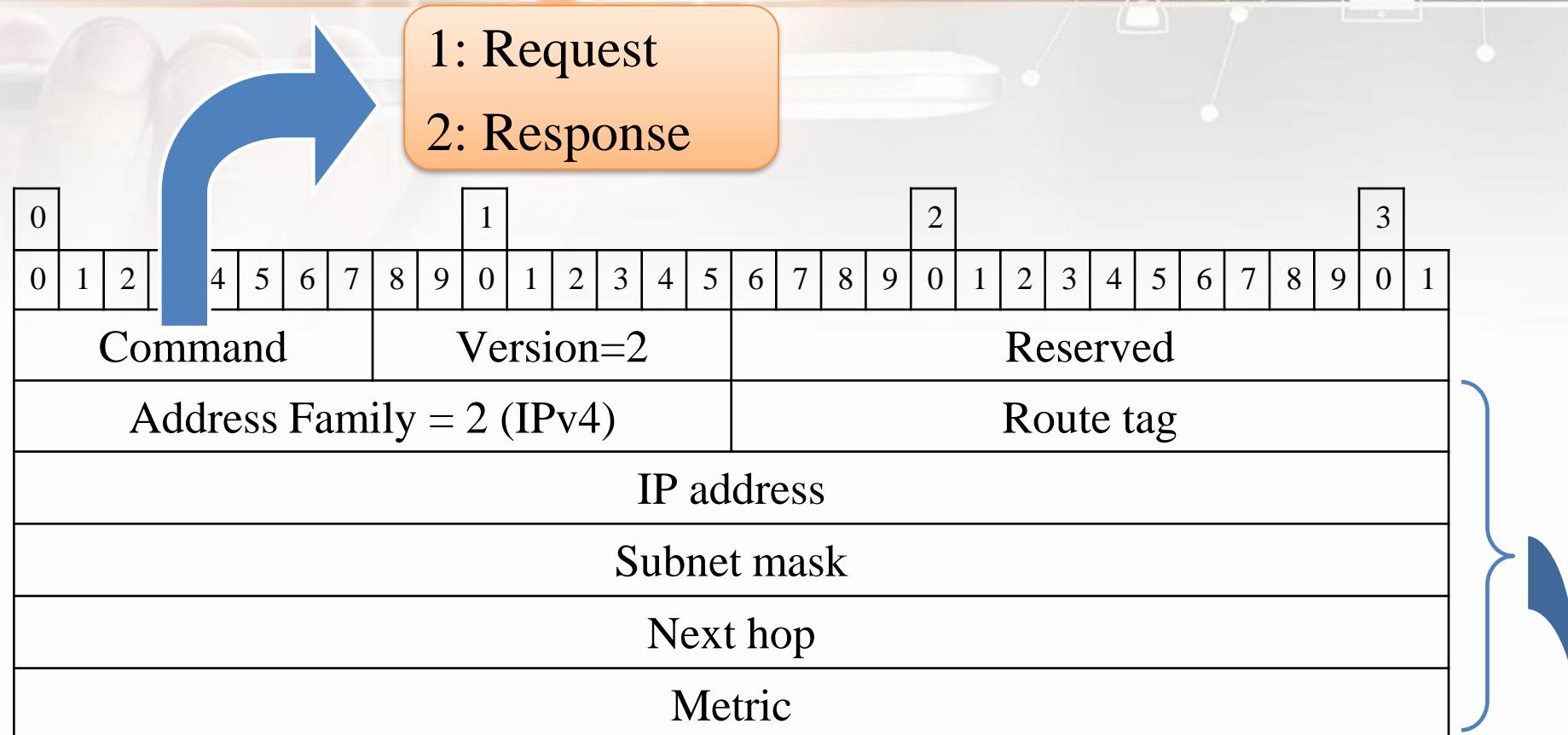


- ❑ Informa direcciones CIDR.
- ❑ Updates en forma multicast
- ❑ Updates con autenticación (opcional)
- ❑ Compatible con RIP v1



A informa la red 10.1.0.0/16 y C 10.4.1.32/27

# RIP v2: sin autenticación



1 a 25 entradas

# RIP v2: con autenticación

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1																													
Command										Version=2										Reserved																																								
Address Family = 0xFFFF (Auth)																Authentication type																																												
Authentication information ( 16 bytes)																																																												
Address Family = 2 (IPv4)																Route tag																																												
IP address																																																												
Subnet mask																																																												
Next hop																																																												
Metric																																																												
Address Family = 0xFFFF (Auth)																Authentication type																																												
Authentication information (16 bytes)																																																												

## Configuración de RIP

- ◆ Implementado con el demonio **routed**.
- ◆ No necesita parámetros ni archivos de configuración.
- ◆ Puede crearse en forma opcional el archivo **/etc/gateways** que provee seteos iniciales.

Cada línea de **/etc/gateways** contiene una entrada similar a la tabla de ruteo, por ejemplo

```
net 192.160.14.0 gateway 172.14.88.12 metric 1 passive
```

# Link state



Cada router recibe del resto de los routers información sobre sus vecinos. En base a ella arma un grafo de la red y calcula el o los caminos más cortos.

Cada router debe hacer lo siguiente:

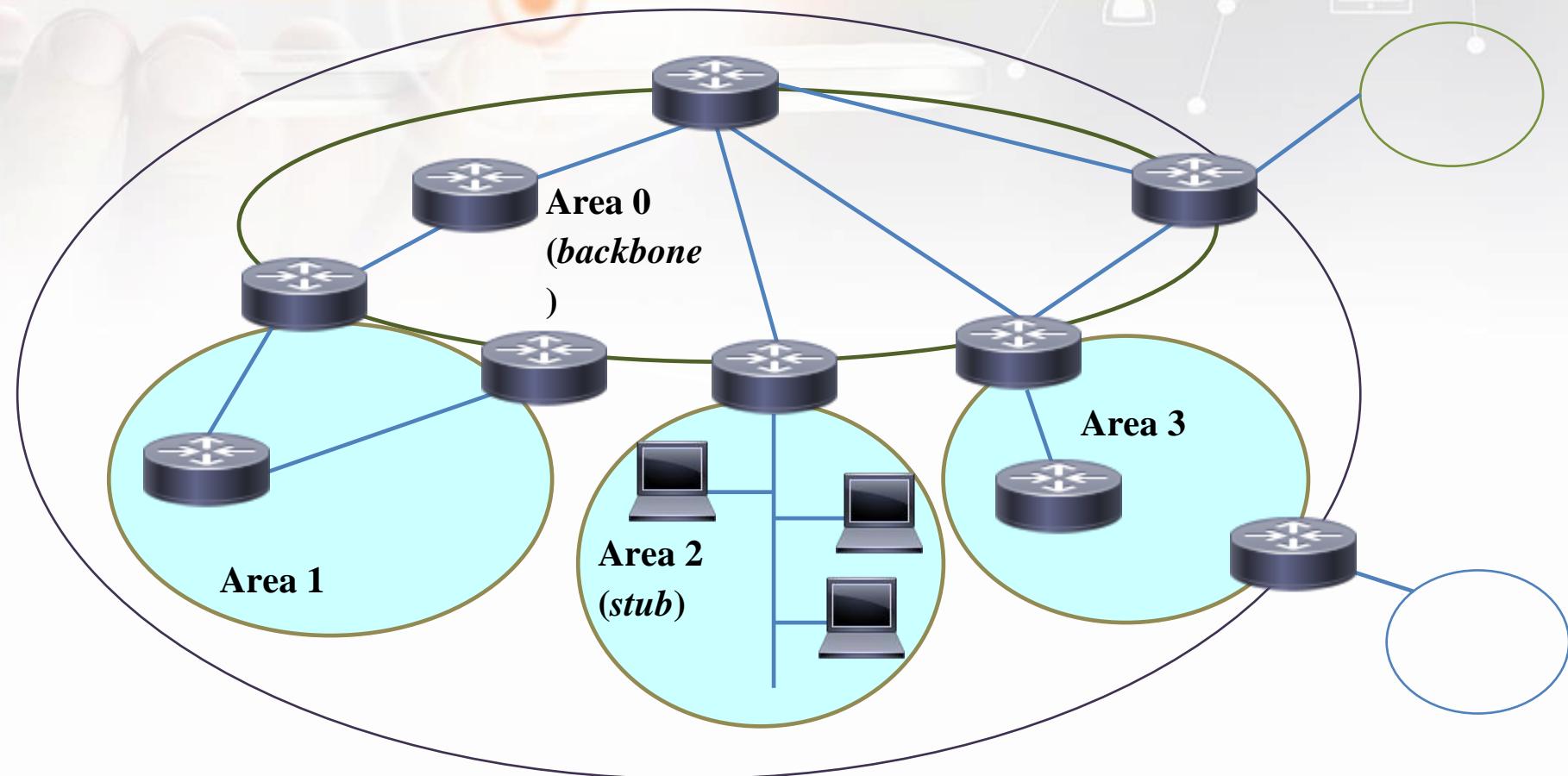
1. Descubrir sus vecinos
2. Medir la distancia (demora o costo) a cada vecino
3. Construir un paquete con lo que aprendió
4. Enviar el paquete a todos los routers de la red
5. Calcular el camino más corto a cada router

# OSPF

OSPF (open shortest path first) RFC 2328

- ◆ **Modelo jerárquico**
- ◆ Intercambia información sobre sus vecinos con toda la red
- ◆ Se envía información cuando se detecta un cambio en la topología.
- ◆ Permite asignar peso a los enlaces que unen áreas o routers
- ◆ Cada router construye un grafo de la red y utiliza Dijkstra para construir el camino más corto.
- ◆ Permite definir áreas. Un área es una colección interconectada de redes. Cada área intercambia información con otras áreas a través de un *router de borde*.

# OSPF: definición de áreas



# OSPF: definición de áreas

- ◆ Cada área está conectada con el área cero (*área backbone*) a través de los routers de borde (ABR)
- ◆ Los routers de borde summarizan las redes de un área
- ◆ Reduce las entradas en las tablas de ruteo
- ◆ El impacto por los cambios de topología está localizado

Las áreas “stub” no requieren un router de borde con gran capacidad, generalmente su tabla de ruteo se configuran en forma estática y con información sobre una subred local y una ruta por defecto.

# OSPF

## Configuración de OSPF

OSPF es implementado con el demonio **gated**, que soporta múltiples protocolos (RIP v1, RIP v2, OSPF, BGP). Para configurar gated se debe usar el archivo **/etc/gated.conf**.

# Routing dinámico



Distance vector	Link state
Visualiza la topología de red en base a la visión de sus vecinos	Visualiza la topología completa de la red
Suma distancias de router a router	Calcula la ruta más corta hacia otros routers
Actualizaciones frecuentes y en forma periódica	Actualizaciones activadas por cambios
Envía copia de la tabla de enrutamiento a los vecinos	Envía estado de enlace a todos los routers
Limita el diámetro de la red (15 “saltos” en RIP).	No tiene límite para el diámetro de la red.

# Material de lectura

Capítulos 4.5 y 4.6 de la bibliografía



## Capa de enlace

- ◆ Nivel de enlace
- ◆ Protocolos de acceso al medio
- ◆ Ethernet

# Nivel de enlace



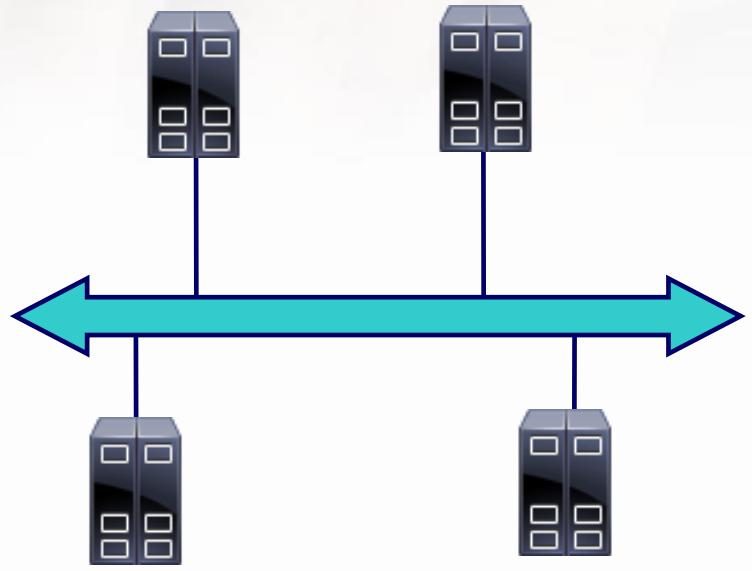
- ◆ Trama: un paquete de nivel de enlace, encapsula datagramas
- ◆ Transferencia confiable de datagramas entre dos nodos adyacentes
- ◆ Cada interface tiene su número MAC único, otorgado por IEEE

# Nivel de enlace: servicios

- ◆ Control de flujo
- ◆ Detección de errores
- ◆ Corrección de errores
- ◆ Half-duplex o Full-duplex

# Protocolos de enlace

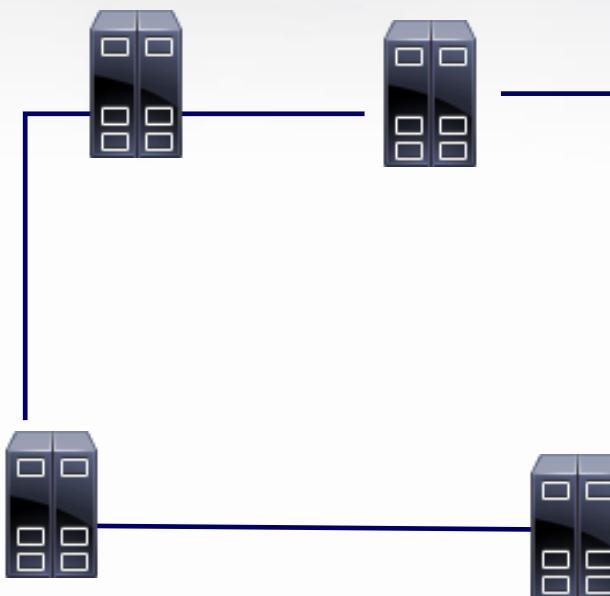
Existen dos tipos de comunicaciones a nivel de enlace



Red broadcast



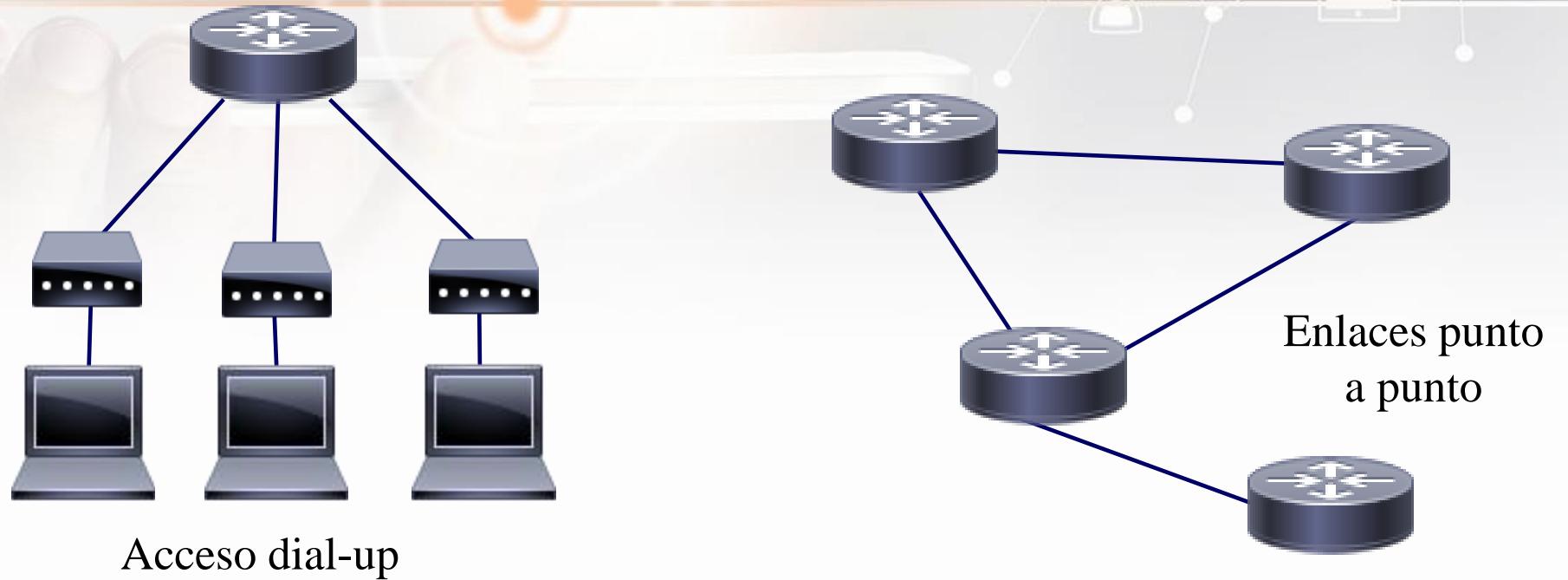
Nivel de enlace



Comunicación punto a punto



# Protocolos punto a punto



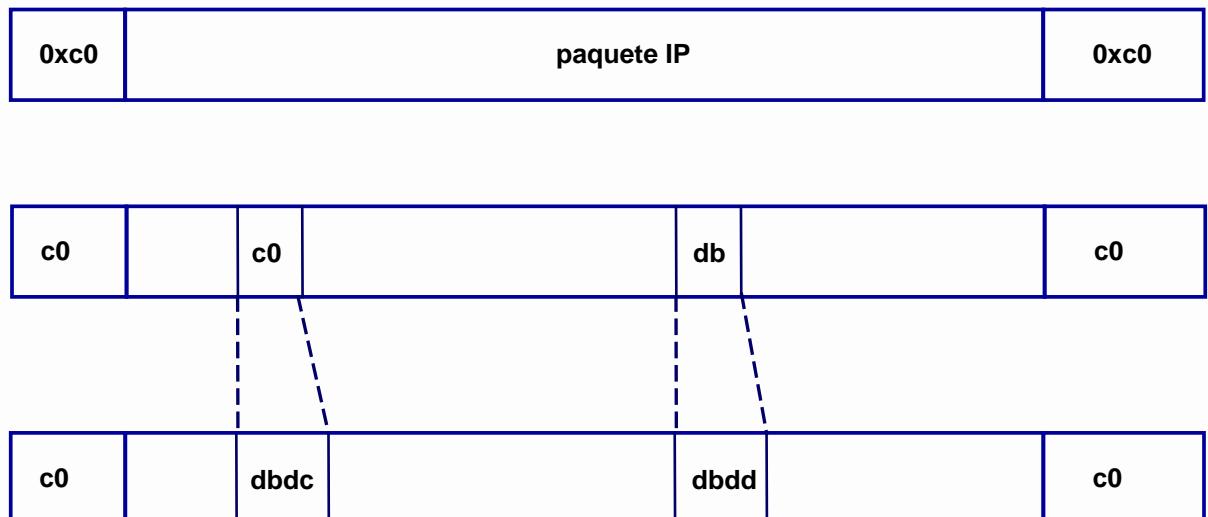
Los protocolos de nivel de enlace para links punto a punto son más simples que para enlaces broadcast:

- Generalmente usados para encapsular datagramas IP
- No se necesita control de acceso al medio

# Protocolos punto a punto: SLIP

## SLIP (RFC 1055, 1988)

- Extremadamente simple.
- Usado para encapsular datagramas IP sobre líneas seriales



# Protocolos punto a punto: PPP

## PPP (RFC 1661, 1992)

- ◆ Sucesor de SLIP
- ◆ Soporta encapsulamiento de varios protocolos
- ◆ Detección de errores
- ◆ Autenticación
- ◆ Notificación de direcciones
- ◆ Negociación de opciones
- ◆ Medición de calidad del enlace

# Redes broadcast



- ◆ Un canal compartido
- ◆ Posibilidad de interferencia
  - ◆ Colisión: si el nodo recibe dos o más señales al mismo tiempo
  - ◆ Debe existir una norma de cómo se comparte el canal
  - ◆ La coordinación se hace usando el mismo canal de datos

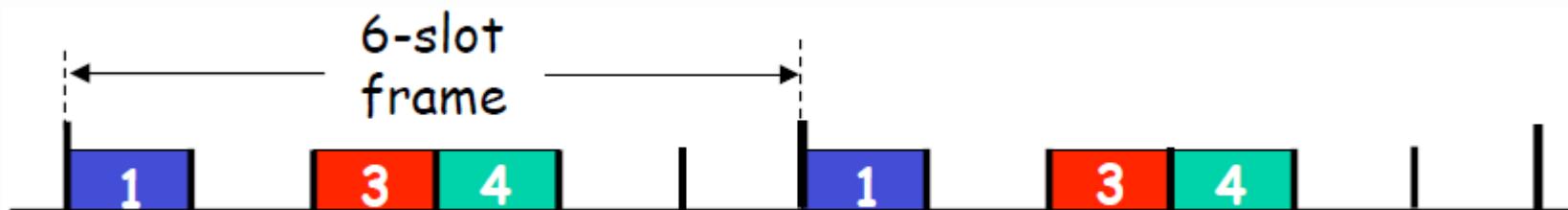
# Redes broadcast



- ◆ Tres clases de protocolos para acceder al medio compartido
  - ◆ Protocolos de «particionamiento de canal»
    - ◆ Dividir el canal ( slots de tiempo, frecuencia)
    - ◆ Cada «partecita» es de uso exclusivo
  - ◆ Por turnos
  - ◆ Acceso aleatorio

# Particionamiento del canal

- ◆ TDMA (*Time Division Multiple Access*)
  - ◆ Acceso al canal en turnos
  - ◆ Cada nodo tiene un tamaño fijo de *slot* por turno
  - ◆ *Slots* no usados se desperdician

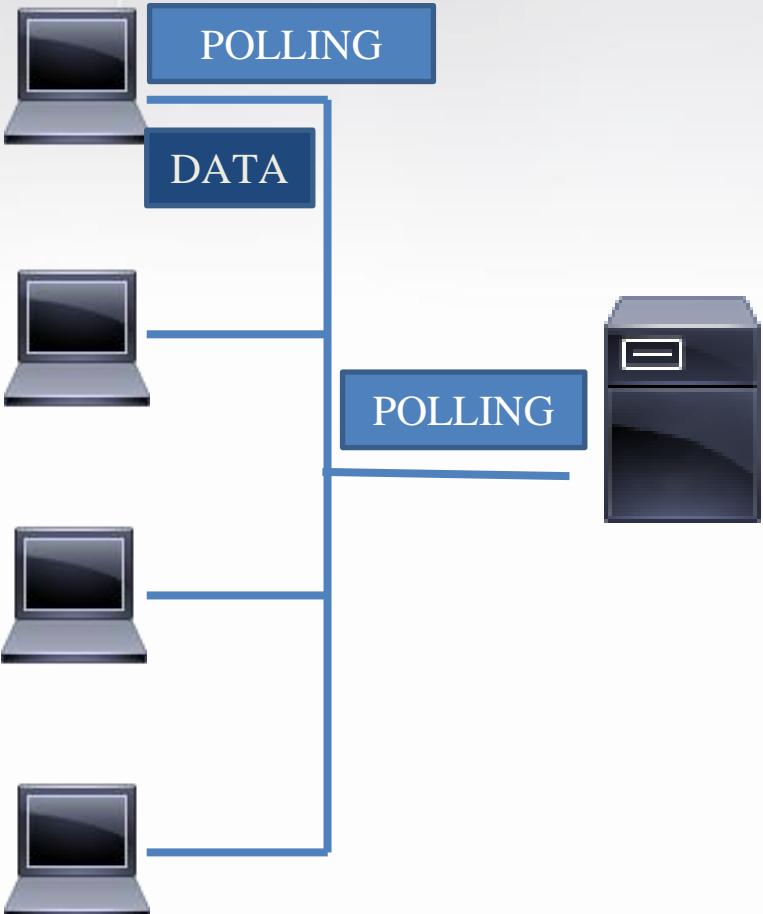


Nivel de enlace

# Protocolos por turnos

## ♦ Polling

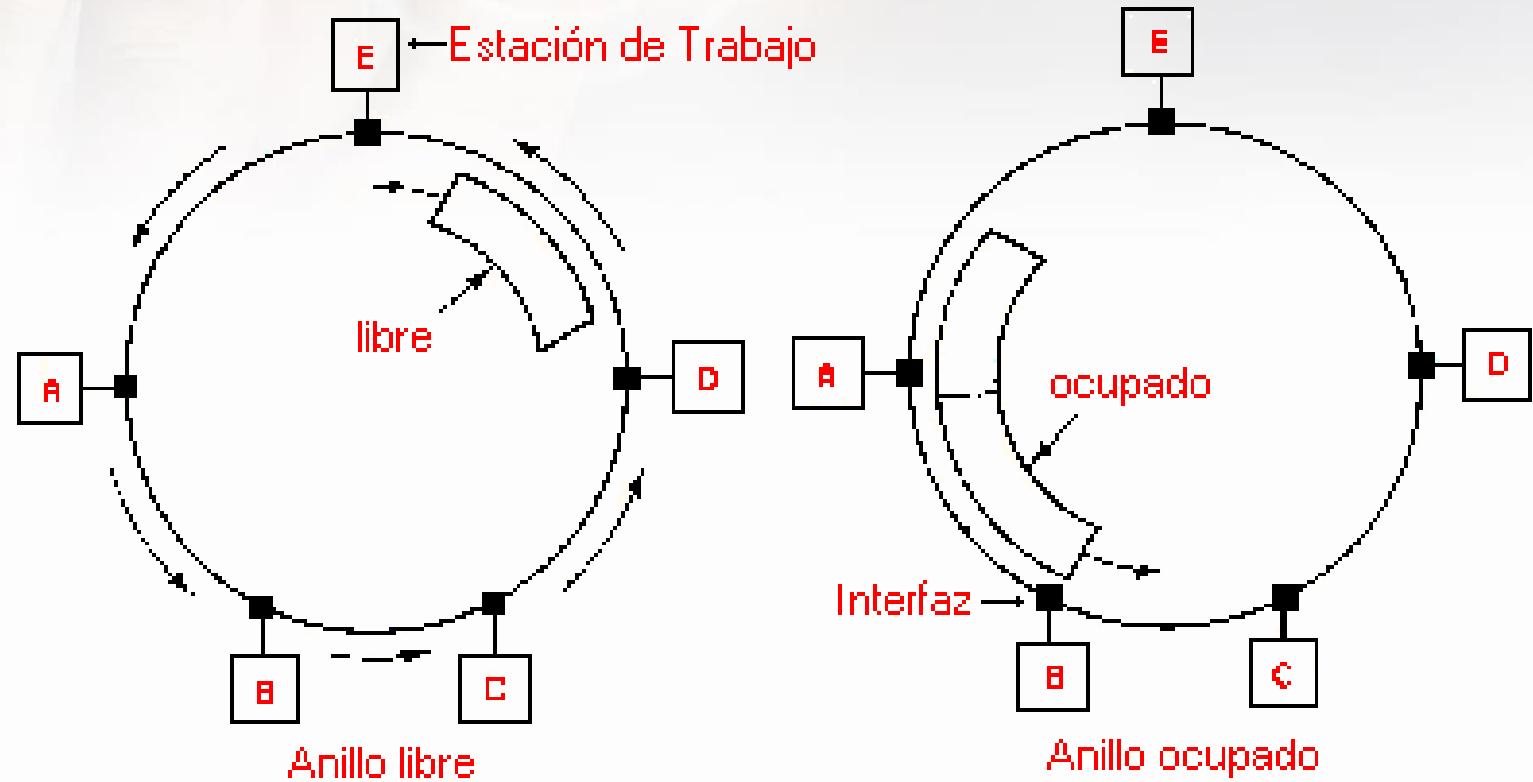
- ♦ Un nodo «master» invita al nodo 1 a transmitir
- ♦ Una vez que el nodo termina le avisa al nodo 2
- ♦ Pro: no hay colisiones
- ♦ Contras:
  - ♦ *Polling overhead*
  - ♦ Latencia
  - ♦ Único punto de falla



# Protocolos por turnos: Token Ring

- ◆ Conjunto de nodos conectados en forma de anillo
- ◆ Los datos circulan en un único sentido
- ◆ Cada nodo recibe un «token» que lo habilita a transmitir
- ◆ El anillo es compartido pero sólo transmite el que recibió el «token» durante cierto tiempo

# Token ring



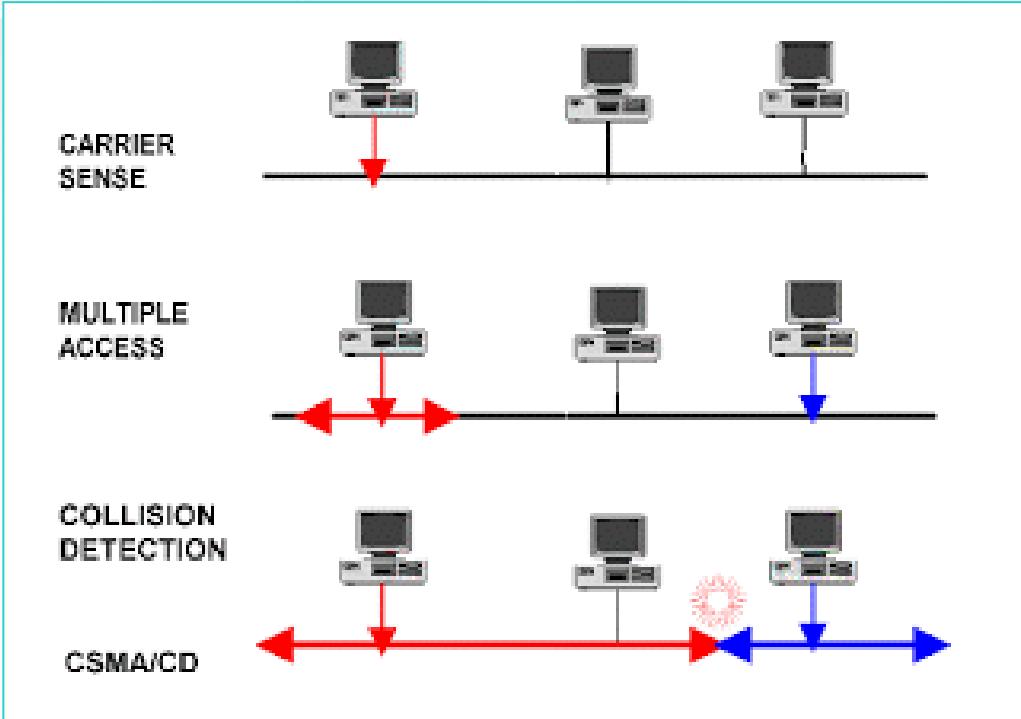
Nivel de enlace

# Protocolos de acceso aleatorio

- ◆ Cuando un nodo quiere transmitir: transmite
- ◆ Si dos nodos transmiten al mismo tiempo: «colisión»
- ◆ El protocolo a usar debe definir
  - ◆ Cómo detectar colisiones
  - ◆ Cómo actuar frente a las colisiones

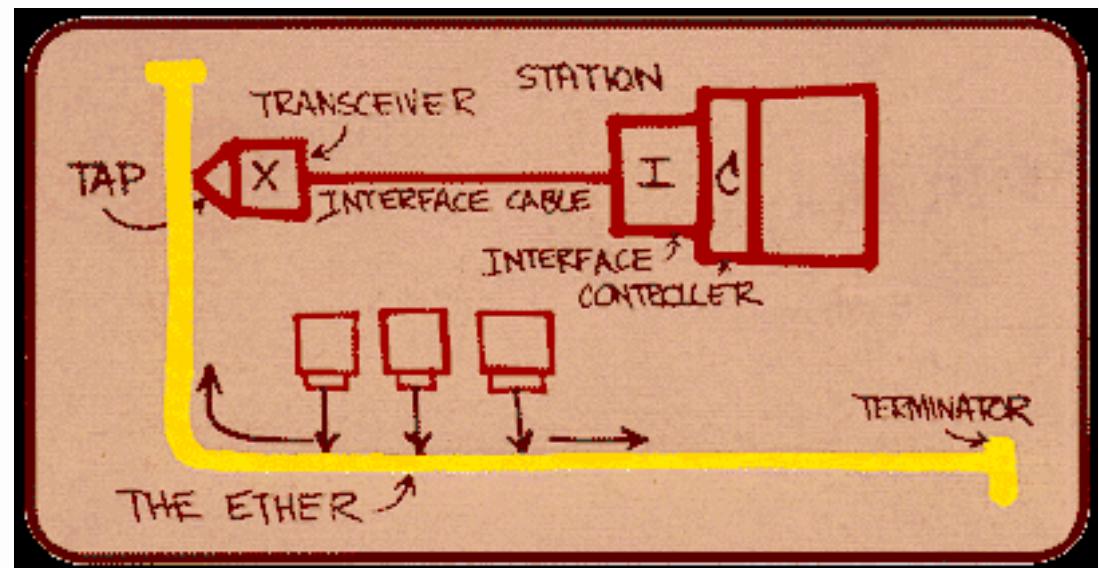
# CSMA (Carrier Sense Multiple Access)

- ◆ Sensar el canal antes de transmitir
- ◆ Si está ocupado, esperar
- ◆ Si está libre, transmitir la trama completa
- ◆ Sensar el medio para verificar si hubo colisión
- ◆ ¿Puede fallar?



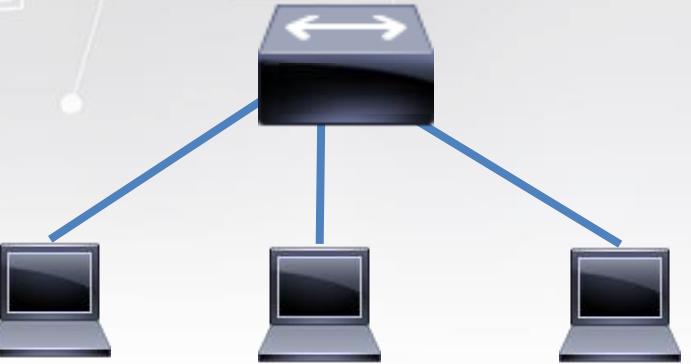
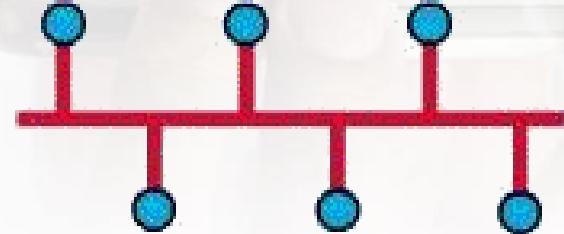
# Ethernet

- ◆ El más usado en LANs cableadas
- ◆ Muy económico
- ◆ Simple
- ◆ Velocidades de 10 Mbps a 10 Gbps



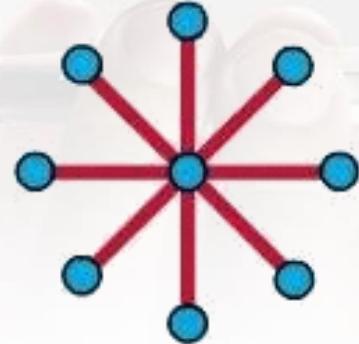
Nivel de enlace

# Ethernet: Topología de bus



- ◆ Todos los hosts en la misma línea, conectados a un hub
- ◆ Cada host se puede conectar o desconectar en cualquier momento
- ◆ Gran posibilidad de colisiones
- ◆ Si la red crece, se la divide en segmentos (dominios de colisión)

# Ethernet: topología de estrella



- ◆ Cada computadora se conecta a un **switch**.
- ◆ Cada host se puede conectar o desconectar sin afectar al resto de la red.
- ◆ Dirige los datos por el puerto al que está conectado el host, pudiendo encolar distintas tramas (**frames**).
- ◆ Se basan en el número de placa (**MAC address**)
- ◆ Se deben evitar los ciclos.

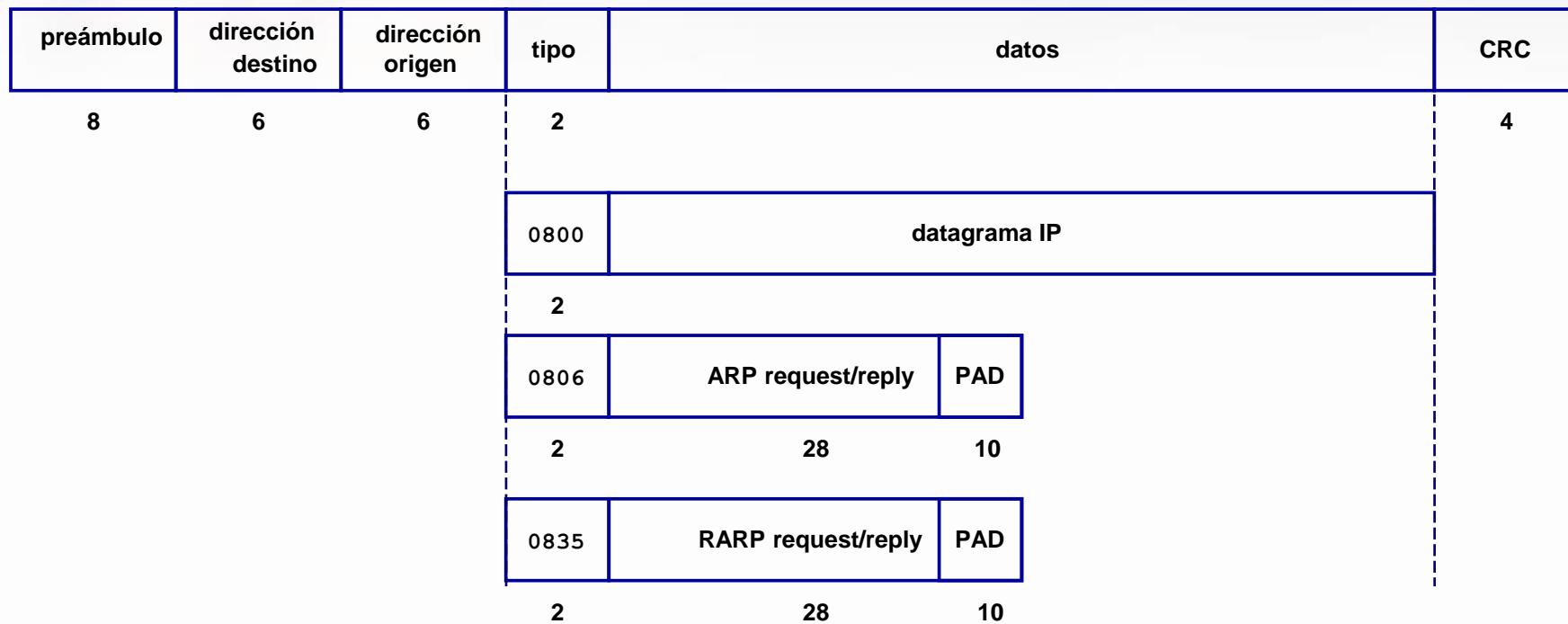
# MAC address



- ◆ 48 bits divididos en 4 secciones
  - ◆ bit 1 (receiver) 0 unicast, 1 para multicast
  - ◆ bit 2 (registry)
    - ◆ 0 si es válida universalmente (asignada a un fabricante)
    - ◆ 1 si es válida sólo localmente (por ejemplo una máquina virtual)
  - ◆ bits 3 a 24: identifican al fabricante, asignado por la IEEE
  - ◆ bits 25 a 48: número único asignado por cada fabricante

# Ethernet

## Estructura de una trama Ethernet



Nivel de enlace

# Ethernet



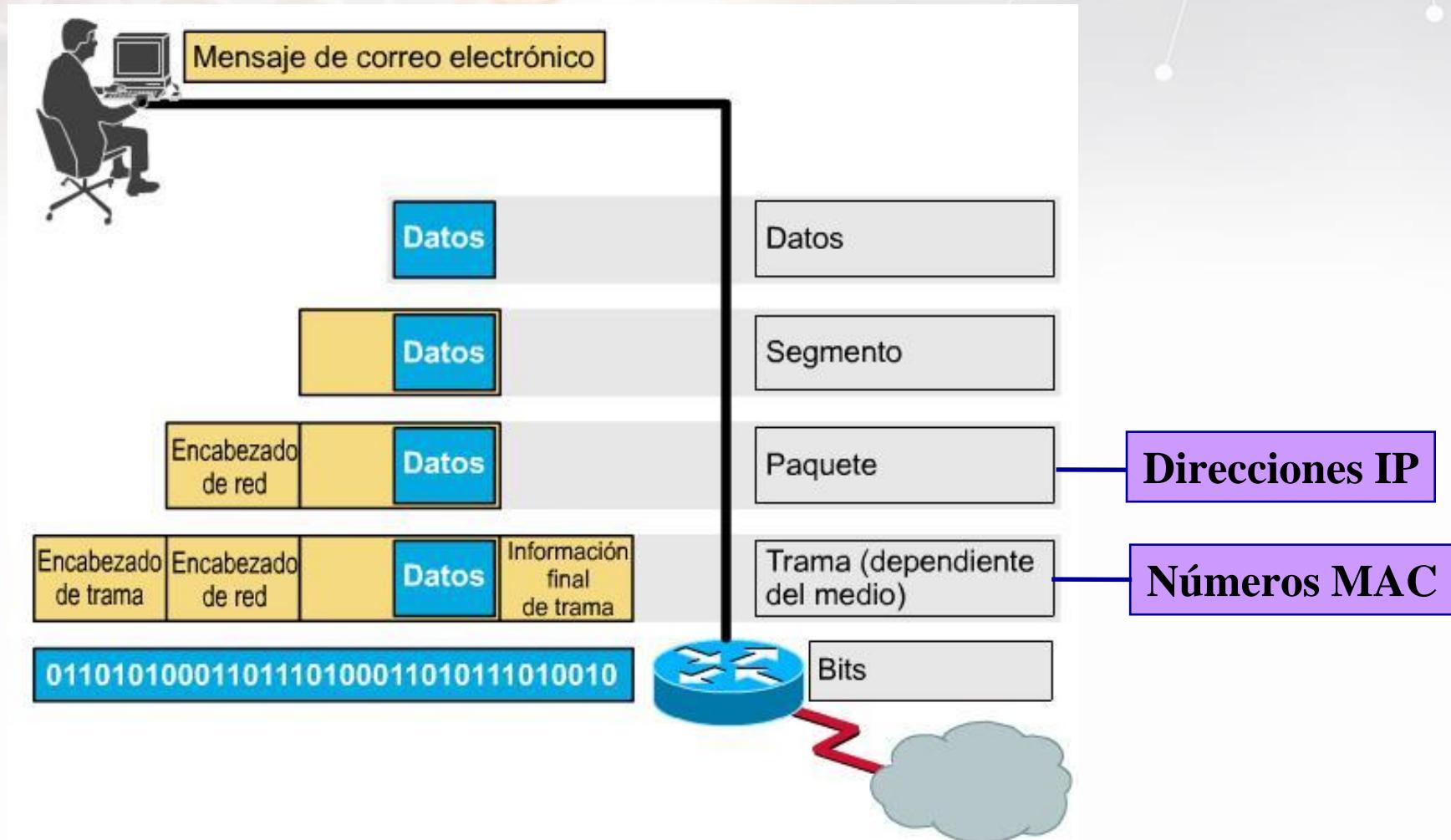
- ◆ Preámbulo: 7 bytes con patrón 10101010 y uno 10101011
- ◆ Dirección destino: si la NIC recibe una trama y la dirección destino no es la de ella ni broadcast, lo ignora
- ◆ Tipo: código del protocolo encapsulado o longitud de la trama
- ◆ CRC: usada por el receptor, si detecta errores ignora la trama
- ◆ Datos: longitudes mínimas y máximas

# Ethernet: velocidades y límites

Tipo	Medio	Ancho de banda máximo	Longitud máxima de segmento	Topología Física	Topología Lógica
10Base5	Coaxial grueso	10 Mbps	500 m	Bus	Bus
10Base-T	UTP Cat 5	10 Mbps	100 m	Estrella; Estrella Extendida	Bus
10Base-FL	Fibra óptica multimodo	10 Mbps	2.000 m	Estrella	Bus
100Base-TX	UTP Cat 5	100 Mbps	100 m	Estrella	Bus
100Base-FX	Fibra óptica multimodo	100 Mbps	2.000 m	Estrella	Bus
1000Base-T	UTP Cat 5	1000 Mbps	100 m	Estrella	Bus

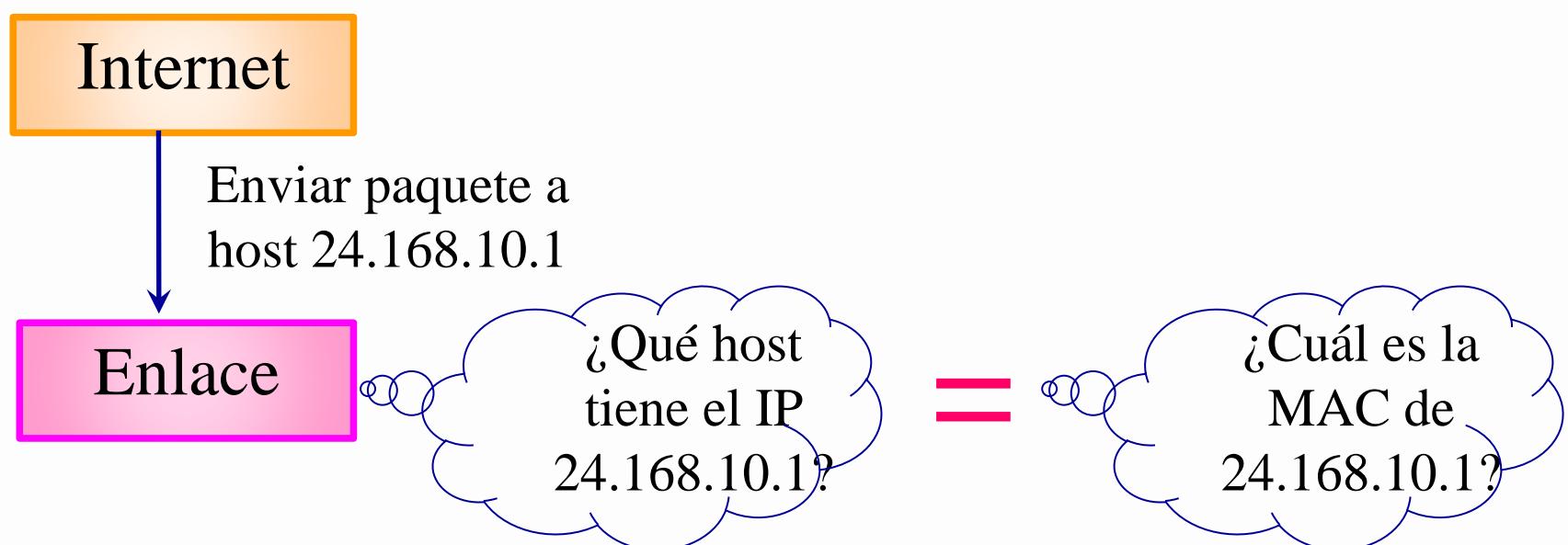
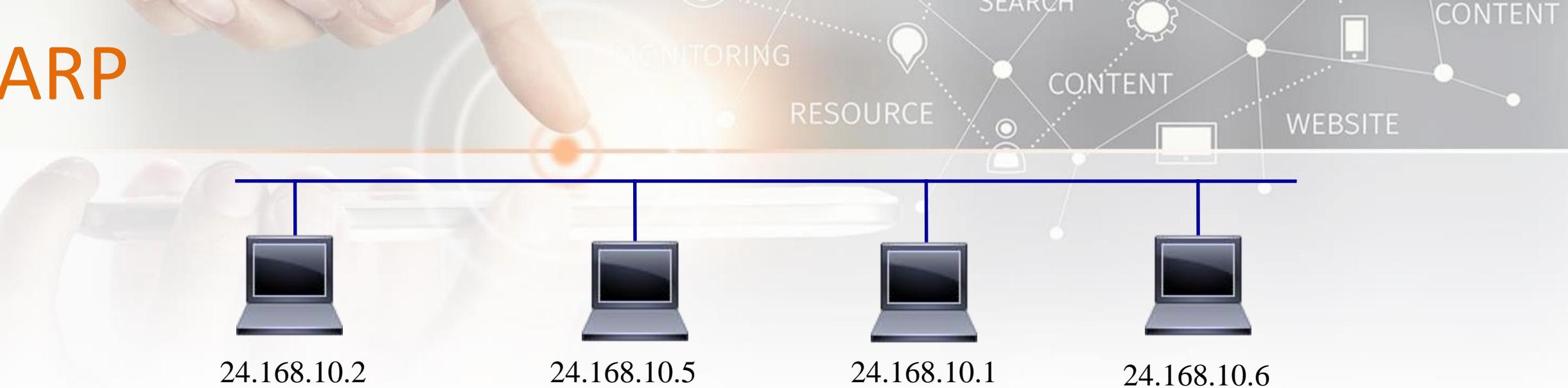
Nivel de enlace

# ARP



Nivel de enlace

# ARP



Nivel de enlace

# ARP



Formato genérico de una trama:

Inicio de trama	Dirección	Tipo / longitud	Datos	FCS	Fin de trama
-----------------	-----------	-----------------	-------	-----	--------------

Trama Ethernet 802.3

Preámbulo	Delimitador de trama	MAC destino	MAC origen	Long / Tipo	Datos	Pad	FCS
-----------	----------------------	-------------	------------	-------------	-------	-----	-----

# ARP



Para poder enviar tramas de un host a otro, se debe conocer el número MAC del host destino. ¿Cómo sabe el host, dado un número IP, a qué MAC address corresponde?

El comando **arp** nos muestra una tabla que mantiene cada host con la relación entre números IP y números MAC. También permite modificar esa tabla.

Ejemplo:

```
user@server:~$ arp -a
Interface: 192.168.2.20 --- 0x10003
Internet Address      Physical Address          Type
 192.168.2.1           00-03-0a-00-d8-d0    dynamic
 192.168.2.4           00-00-21-fe-77-03    dynamic
```

# ARP



Ejemplo:

```
user@server:~$ arp -s 192.168.2.99 00-00-21-fe-77-03
user@server:~$ arp -a
Interface: 192.168.2.20 --- 0x10003
          Internet Address          Physical Address      Type
          192.168.2.1                00-03-0a-00-d8-d0  dynamic
          192.168.2.4                00-00-21-fe-77-03  dynamic
          192.168.2.99               00-00-21-fe-77-03  static
```

¿Qué sucederá cuando se desee enviar un paquete IP al host 192.168.2.99 ?

# ARP

## Trama ARP request

**Ethernet II, Src: 00:0c:76:b8:47:dd, Dst: ff:ff:ff:ff:ff:ff**

Destination: ff:ff:ff:ff:ff:ff

Source: 00:0c:76:b8:47:dd

Type: ARP (0x0806)

### Address Resolution Protocol (request)

Hardware type: Ethernet (0x0001)

Protocol type: IP (0x0800)

Hardware size: 6

Protocol size: 4

Opcode: request (0x0001)

Sender MAC address: 00:0c:76:b8:47:dd

Sender IP address: 192.168.2.20

Target MAC address: 00:00:00:00:00:00

Target IP address: 192.168.2.1

# ARP

## Trama ARP reply

**Ethernet II, Src: 00:03:0a:00:d8:d0, Dst: 00:0c:76:b8:47:dd**

Destination: 00:0c:76:b8:47:dd

Source: 00:03:0a:00:d8:d0

Type: ARP (0x0806)

### Address Resolution Protocol (reply)

Hardware type: Ethernet (0x0001)

Protocol type: IP (0x0800)

Hardware size: 6

Protocol size: 4

Opcode: reply (0x0002)

Sender MAC address: 00:03:0a:00:d8:d0

Sender IP address: 192.168.2.1

Target MAC address: 00:0c:76:b8:47:dd

Target IP address: 192.168.2.20

# ARP

El protocolo ARP se utiliza de 4 formas

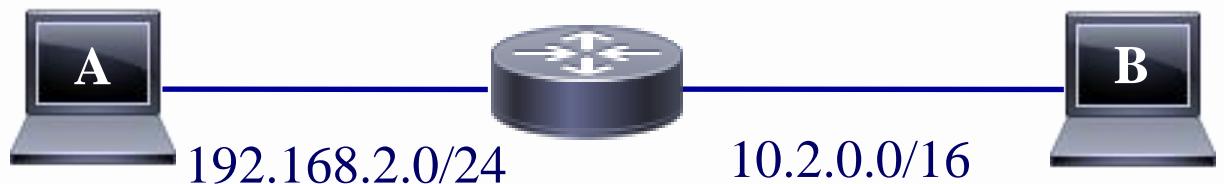
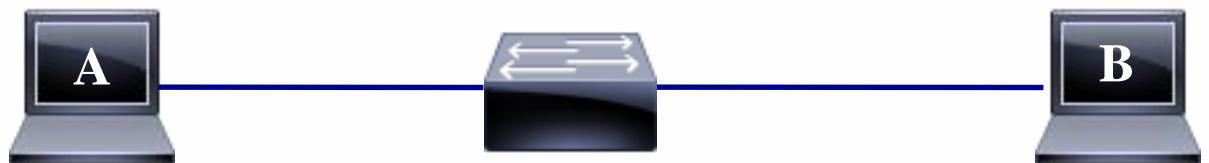
- ◆ **ARP:** un host con dirección IP necesita conocer la dirección MAC de otro host del cual conoce la dirección IP.
- ◆ **RARP:** un host sin almacenamiento (*diskless workstation*) conoce su dirección MAC pero “olvidó” su dirección IP.
- ◆ **InARP:** un host conoce la dirección MAC de otro host pero no su IP.
- ◆ **Proxy ARP:** un host responde a un *request* para otro host.

Ver RFCs 826 y  
903

# ARP



En ambos casos el host A envía un paquete IP al host B.  
Suponiendo que las tablas ARP están vacías, ¿cuántos paquetes ARP se envían en cada caso?



# ARP



A envía paquete IP a  
B



- 1.- A examina su tabla de ruteo y determina que el gateway para llegar a B es 192.168.2.1

# ARP



A envía paquete IP a  
B



2.- A envía ARP request

## Ethernet

Destination:

ff:ff:ff:ff:ff:ff

Source:

MAC A

## ARP

Sender IP address:

192.168.2.2

Sender MAC address:

MAC A

Target MAC address:

00:00:00:00:00:00

Target IP address:

192.168.2.1

# ARP



A envía paquete IP a  
B



3.- El router envía ARP reply

## Ethernet

Destination:	MAC A
Source:	MAC Router

## ARP

Sender IP address:	192.168.2.1
Sender MAC address:	MAC Router
Target MAC address:	MAC A
Target IP address:	192.168.2.2

# ARP



A envía paquete IP a  
B



4.- A envía paquete IP

**Ethernet**

Destination: MAC Router

Source: MAC A

**IP**

Sender IP address: 192.168.2.2

Target IP address: 10.2.0.2

# ARP

Nivel de enlace

A envía paquete IP a

B



- 5.- El router examina su tabla de ruteo y determina que B está en la misma red 10.2.0.0/16

# ARP

A envía paquete IP a

B



6.- El router envía ARP request

### Ethernet

Destination: ff:ff:ff:ff:ff:ff

Source: MAC Router

### ARP

Sender IP address: 10.2.0.1

Sender MAC address: MAC Router

Target MAC address: 00:00:00:00:00:00

Target IP address: 10.2.0.2

# ARP



A envía paquete IP a  
B



7.- B envía ARP reply

## Ethernet

Destination: MAC Router  
Source: MAC B

## ARP

Sender IP address: 10.2.0.2  
Sender MAC address: MAC B  
Target MAC address: MAC Router  
Target IP address: 10.2.0.1

# ARP



A envía paquete IP a  
B



8.- El router envía paquete IP a B

## Ethernet

Destination: MAC B

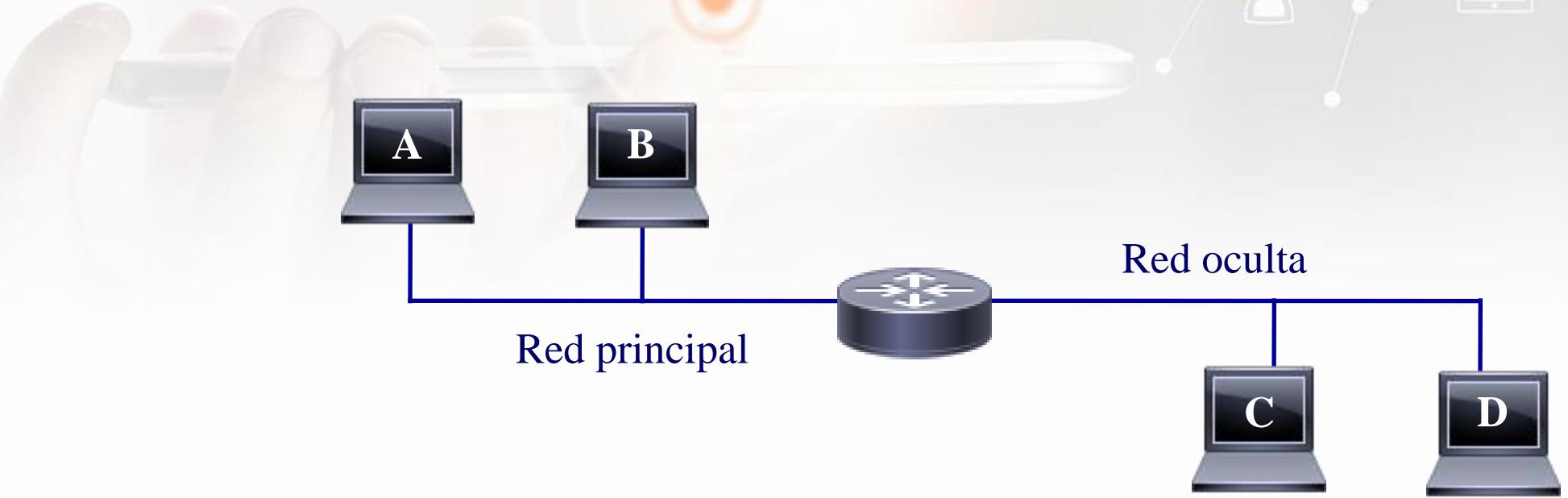
Source: MAC Router

## IP

Sender IP address: 192.168.2.2

Target IP address: 10.2.0.2

# ARP: Proxy ARP

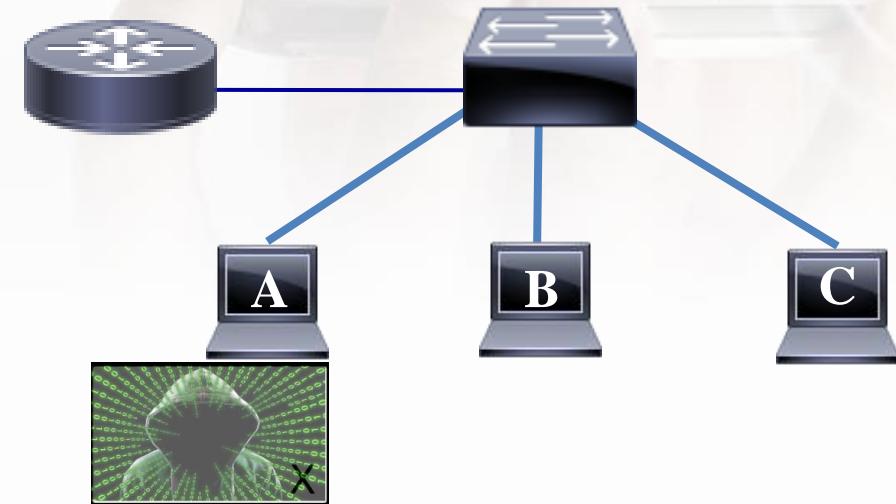


Si el host A solicita nro MAC de C o D, el router responde con su propio nro MAC, por lo que la relación entre número de MAC y nro de IP no es 1 a 1.

# ARP spoofing

- ◆ ARP no contempla autenticación, cualquiera puede responder un ARP request para relacionar un IP con un MAC, lo que permite distintos tipos de ataque
  - ◆ “Man-in-the-middle”
  - ◆ “Denial-of-service”
  - ◆ Session hijacking
- ◆ Algunos programas para experimentar
  - ◆ Arpspoof, Arpoison, Cain & Abel, Ettercap

# ARP spoofing



El host A, una vez que aprende las direcciones IP, envía ARP reply en forma periódica con las IPs conocidas y su MAC

Los hosts comenzarán a enviar paquetes con IP correcta pero con la MAC del atacante.

¿Qué debe hacer el atacante para negarle servicio a B y C?

¿Cómo se pueden defender los hosts?

# ARP spoofing



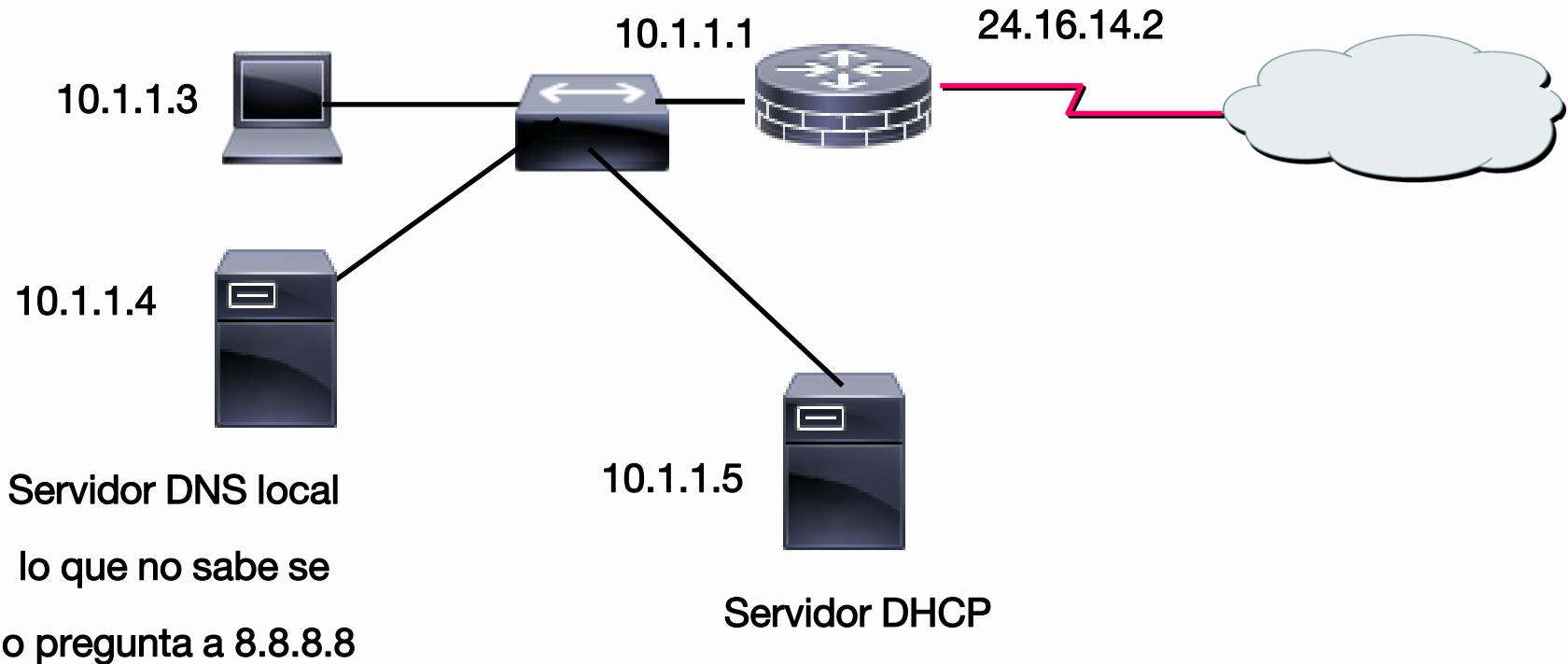
- ◆ Usos legales de ARP spoofing
- ◆ Redireccionar equipos no registrados a una página de registro
- ◆ Acceso a Internet en hoteles a dispositivos móviles
- ◆ Implementar redundancia en servicios de red

# ARP spoofing: defensas

- ◆ Añadir entradas estáticas en la tabla ARP
- ◆ DHCP snooping (dispositivos Cisco, Allied Telesis, otros)
- ◆ Arpwatch: programa Unix que escucha respuestas ARP y envía mail al administrador si una entrada cambia
- ◆ Enviar RARP request. Si hay más de una respuesta puede ser un ataque

# Síntesis

Asumiendo que las tablas ARP están vacías, detallar los paquetes que se generan en la red privada si se enciende el host A y se conecta con HTTP a campus.itba.edu.ar



# Material de lectura

Capítulos 5.1 a 5.4.2 inclusive de la bibliografía



# SSH

## Introducción OpenSSH Túneles Autenticación

# SSH



SSH es un protocolo pero también puede ser visto o analizado como una herramienta para encriptar, una aplicación cliente/servidor o una interfaz de comandos.

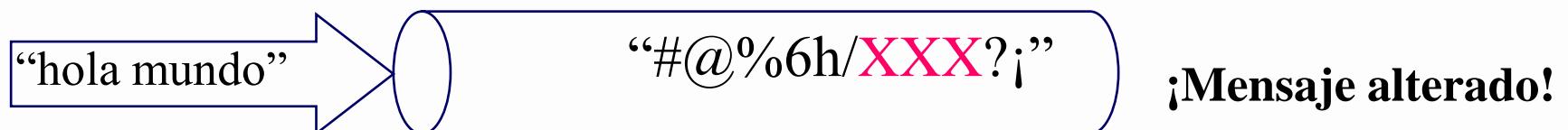
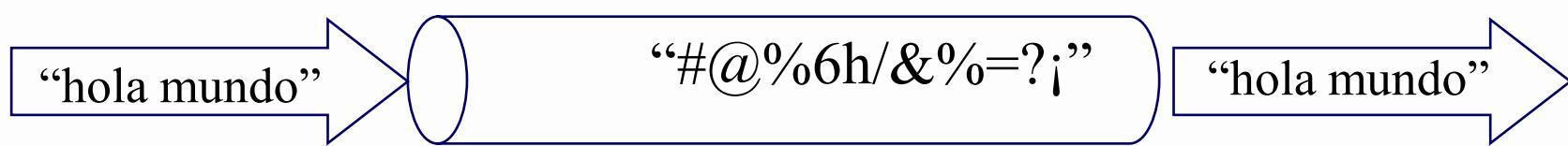
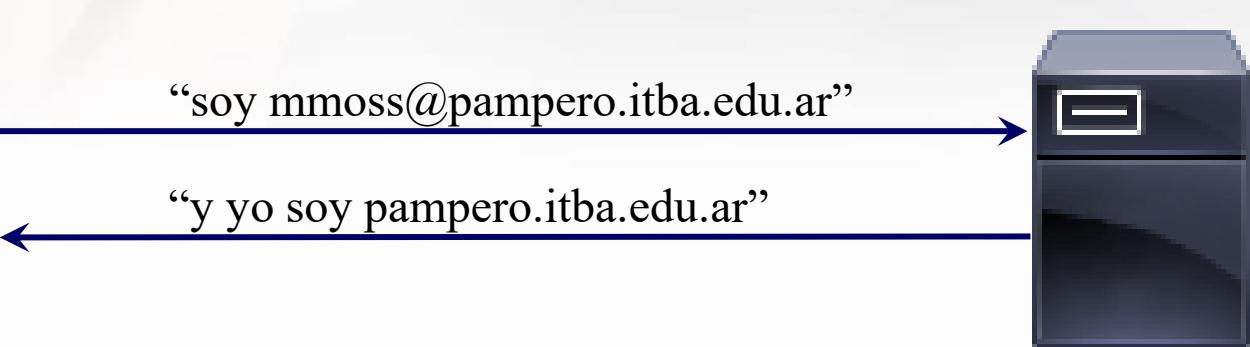
En su forma más habitual o común SSH puede ser visto como un reemplazo seguro para Telnet, ya que tanto los comandos como los datos viajan encriptados.

Aunque SSH es mucho más potente y versátil que un mero login remoto encriptado.

# SSH

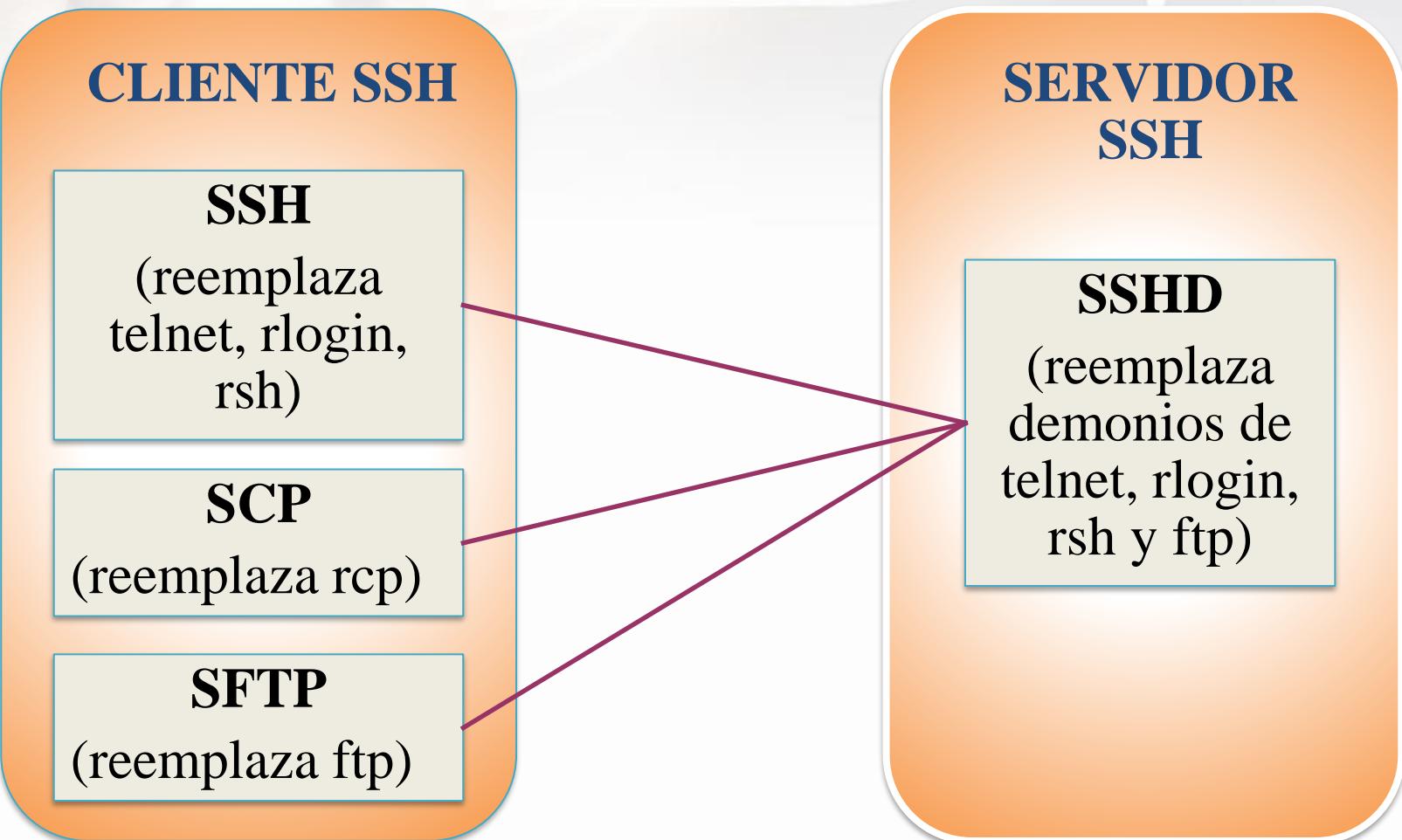


El protocolo SSH ofrece: autenticación , encriptación e integridad



# SSH

SSH es el reemplazo seguro de varias aplicaciones



# SSH



Los principales servidores SSH son:

- OpenSSH: Windows, Unix y Linux
- SSH Communications: Windows, Unix y Linux
- VanDyke's Vsphere, BitVise: Windows.

Nosotros utilizaremos OpenSSH, que viene instalado por defecto en las distribuciones Linux.

El demonio que utiliza es **sshd** y escucha, por defecto, en el puerto 22.

# OpenSSH

## Componentes básicos de OpenSSH

- sshd: Es el demonio que maneja las conexiones entrantes
  - No debe ser iniciado por inetd o xinetd
- ssh: Comando que reemplaza a otros como rsh y rlogin, para loguearse en forma remota o enviar un comando a un servidor remoto.
- scp: Versión segura de rcp, que permite copiar archivos
- sftp: Versión segura de FTP
- ssh-keygen: generador de claves públicas y privadas.

# OpenSSH



El archivo principal de configuración es `/etc/sshd_config`.  
Algunas de las opciones generales son:

```
#Port 22
#Protocol 2,1
#ListenAddress 0.0.0.0
#ListenAddress ::

#LoginGraceTime 120
PermitRootLogin no
#StrictModes yes
```

# Ejemplos de uso

## Sesión de terminal remota

```
user@server:~$ ssh pi@pampero.itba.edu.ar
```

```
Password: *****
```

```
Last Login Wed May 4 16:12:22 from ....
```

```
Linux 2.4.25-lids1.2.0rc2
```

```
user@server:~$ ...
```

```
user@server:~$ logout
```

La misma  
interacción que con  
telnet

## Ejecución remota de un comando

```
user@server:~$ ssh user@host rm /tmp -r
```

# Ejemplos de uso



## Sesión de terminal remota

Algunas de las opciones para ssh son:

- ◆ -1 Fuerza ssh a usar SSH versión 1.
- ◆ -2 Fuerza ssh a usar SSH versión 2.
- ◆ -C Comprime los datos
- ◆ -c blowfish | 3des | des Elegir método de encriptación
- ◆ -v Muestra información de depuración
- ◆ -4 Forzar a IP v4
- ◆ -6 Usar únicamente IP v6
- ◆ -E *log\_file* Info de debug a un archivo
- ◆ -p *port* Usar un puerto distinto a 22
- ◆ -N Sólo conecta, no ejecuta comando remoto

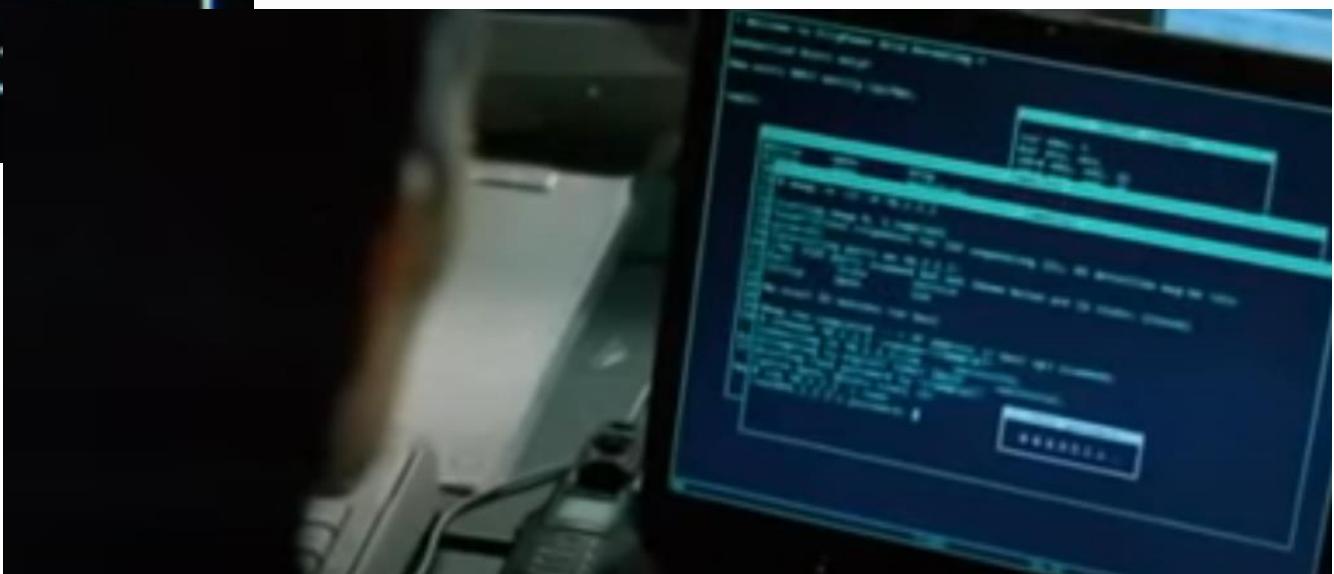
# SSH Versión 1

Las primeras versiones de OpenSSH 1 contenían errores de implementación que la hacían vulnerable, como bien lo sabía Trinity

```
Port      State      Service
22/tcp    open       ssh

No exact OS matches for host

Nmap run completed -- 1 IP address (1 host up) scanned
# sshnuke 10.2.2.2 -rootpw="Z10H0101"
Connecting to 10.2.2.2:ssh ... successful.
Attempting to exploit SSHv1 CRC32 ... successful.
Resetting root password to "Z10H0101".
System open: Access Level <9>
# ssh 10.2.2.2 -l root
root@10.2.2.2's password: 
```



# Ejemplos de uso



## Copiar archivos

```
mgarbe@pampero:~$ scp yo@casa.no-ip.info:*.pdf .
```

Password: \*\*\*\*\*

parcial1.pdf	100%	9428	9.2KB/s	00:00
parcial2.pdf	100%	8508	1.5KB/s	00:01
final.pdf	100%	7099	1.1KB/s	00:00

Además de las opciones vistas para ssh, scp incluye:

- -q                      Deshabilita porcentaje de progreso
- -r                      Copia recursivamente un directorio

# Ejemplos de uso

## FTP seguro

```
user@server:~$ sftp mgarbe@pampero.itba.edu.ar
```

```
Password: *****
```

```
sftp> ll
```

```
sftp> lpwd
```

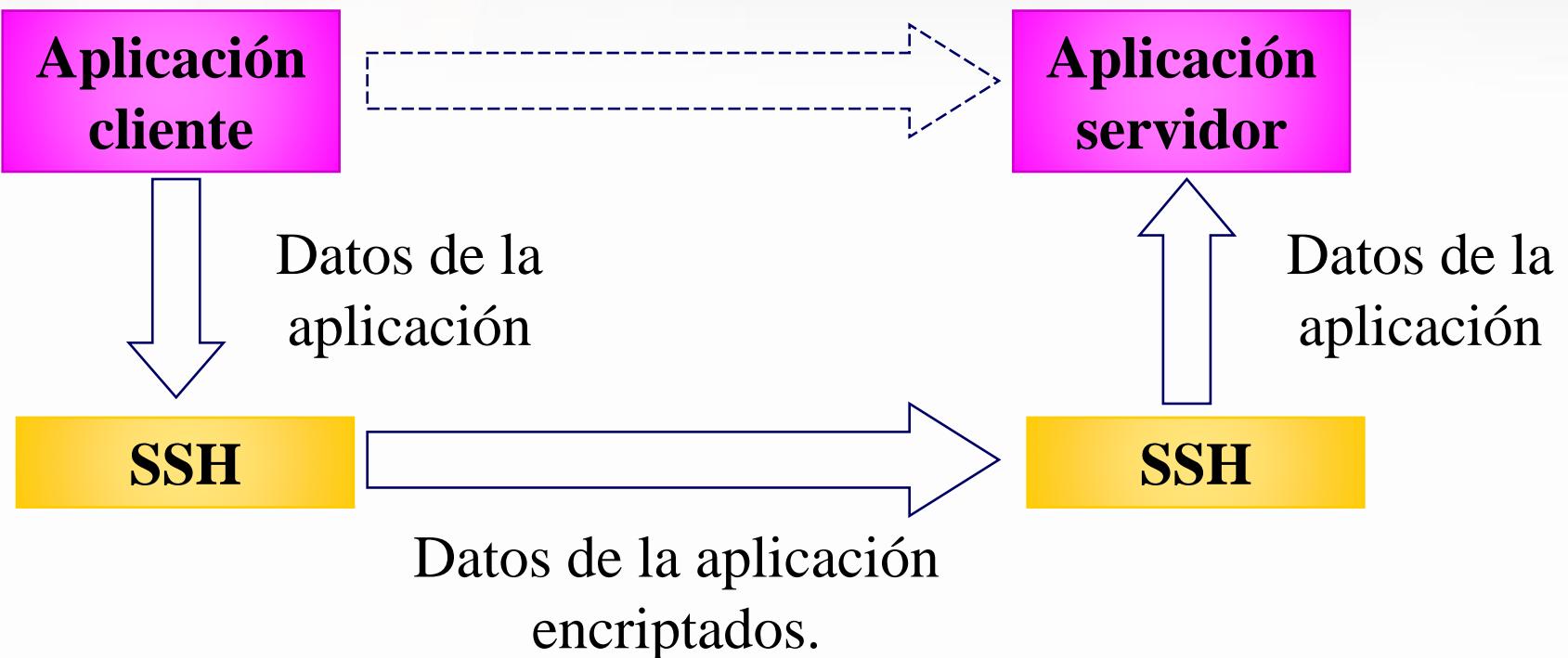
```
sftp> quit
```

Alguna opción particular de sftp:

- **-B *buffer\_size*** Buffer de transferencia.
- **-b *batch\_file*** Lee los comandos desde un archivo y no desde stdin.

# Port forwarding

SSH puede encriptar en forma transparente el flujo de datos de otra aplicación (*tunneling*)



# Port forwarding



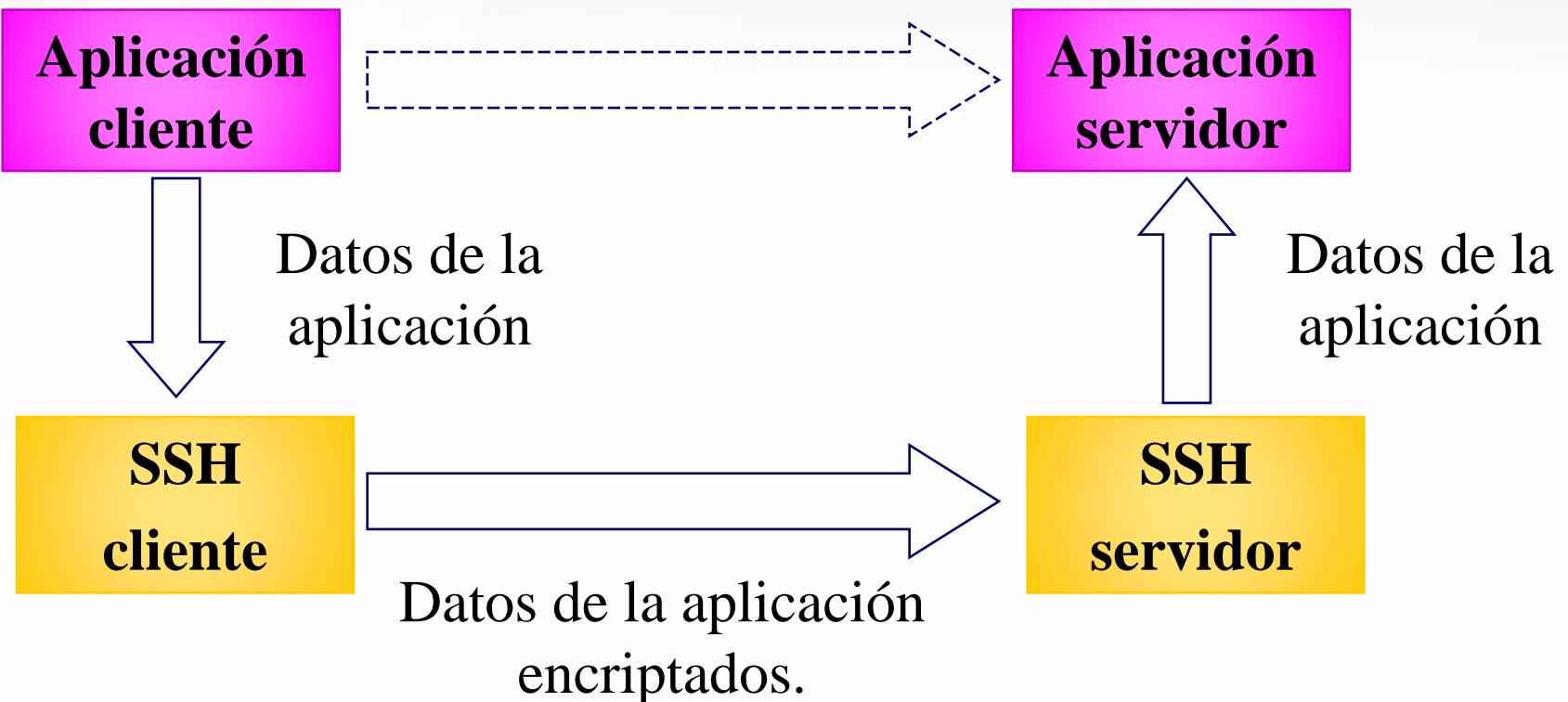
SSH port forwarding protege conexiones TCP redirigiéndolas a través de una conexión SSH. Una vez establecida la conexión se le pueden dar diversos usos, como por ejemplo:

- Acceder a servicios TCP (SMTP, IMAP, POP, LDAP, etc.) a través de un firewall que no permita conexiones a ellos.
- Protege la sesión con estos servicios, encriptando claves y datos.
- Enviar mensajes usando un servidor SMTP que no permita relaying, trabajando en otra red.

Notar que funciona sólo con TCP, no funciona con protocolos basados en UDP como DNS, DHCP, NFS, NetBIOS

# Local port forwarding

En local forwarding la aplicación cliente está localizada en el cliente SSH.  
El socket pasivo para la boca del túnel se abre local (en el host del cliente SSH)



# Port forwarding

Local forwarding.

La sintaxis para indicar que se va a redirigir lo enviado a un puerto local a otro puerto en otro host es

```
$ ssh -Lport:host:hostport SSHserver
```

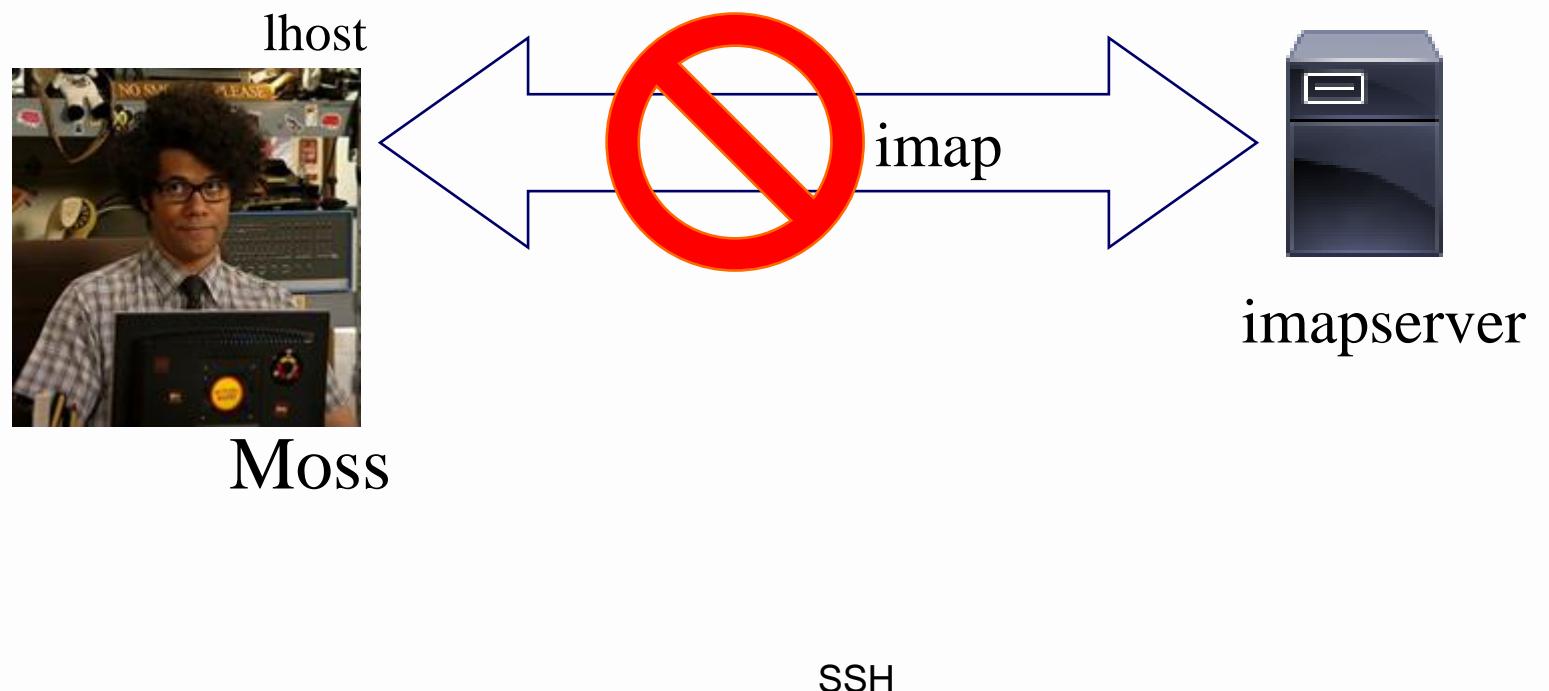
puerto del socket pasivo local

Host remoto, visible desde el servidor SSH

Puerto de la aplicación en el host remoto

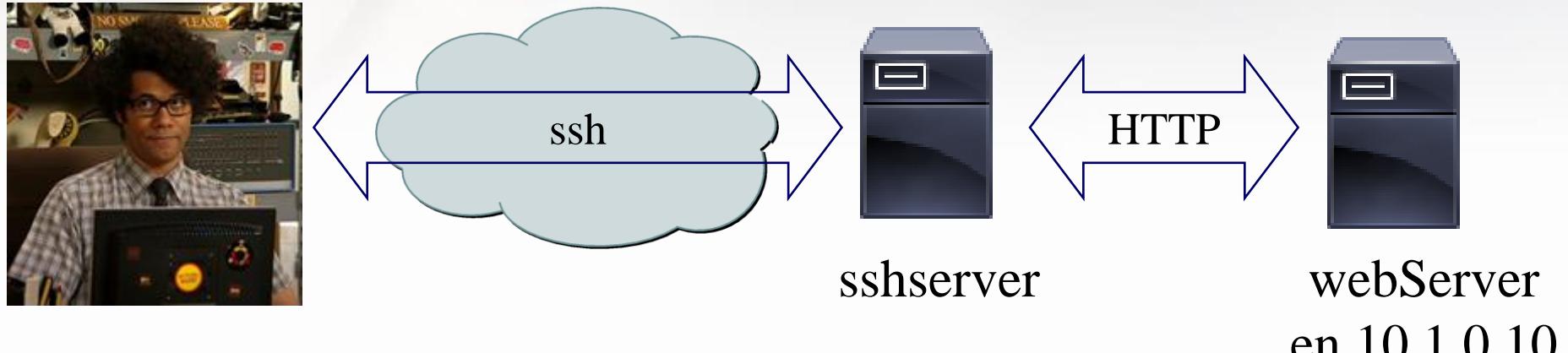
# Local Port forwarding: ejemplo

Moss quiere conectarse a un servidor IMAP remoto pero por alguna razón no puede o no quiere hacerlo directamente



# Local Port forwarding: ejemplo

Moss accede a un servidor web que está "escondido" en una red privada.  
Pero tenemos acceso a un servidor SSH dentro de esa red.



Paso 1: establecer la conexión

```
moss@reynholm~$ ssh -L2001:10.1.0.10:80 -l user sshserver
```

Se establece una sesión remota con el servidor *sshserver* pero además hará que todo lo enviado al puerto local 2001 sea enviado a través de la sesión SSH al puerto 80 de *webServer*.

# Local Port forwarding: ejemplo

Para acceder a un recurso del webserver podemos hacer

```
moss@reynholm~$ curl localhost:2001 .....
```

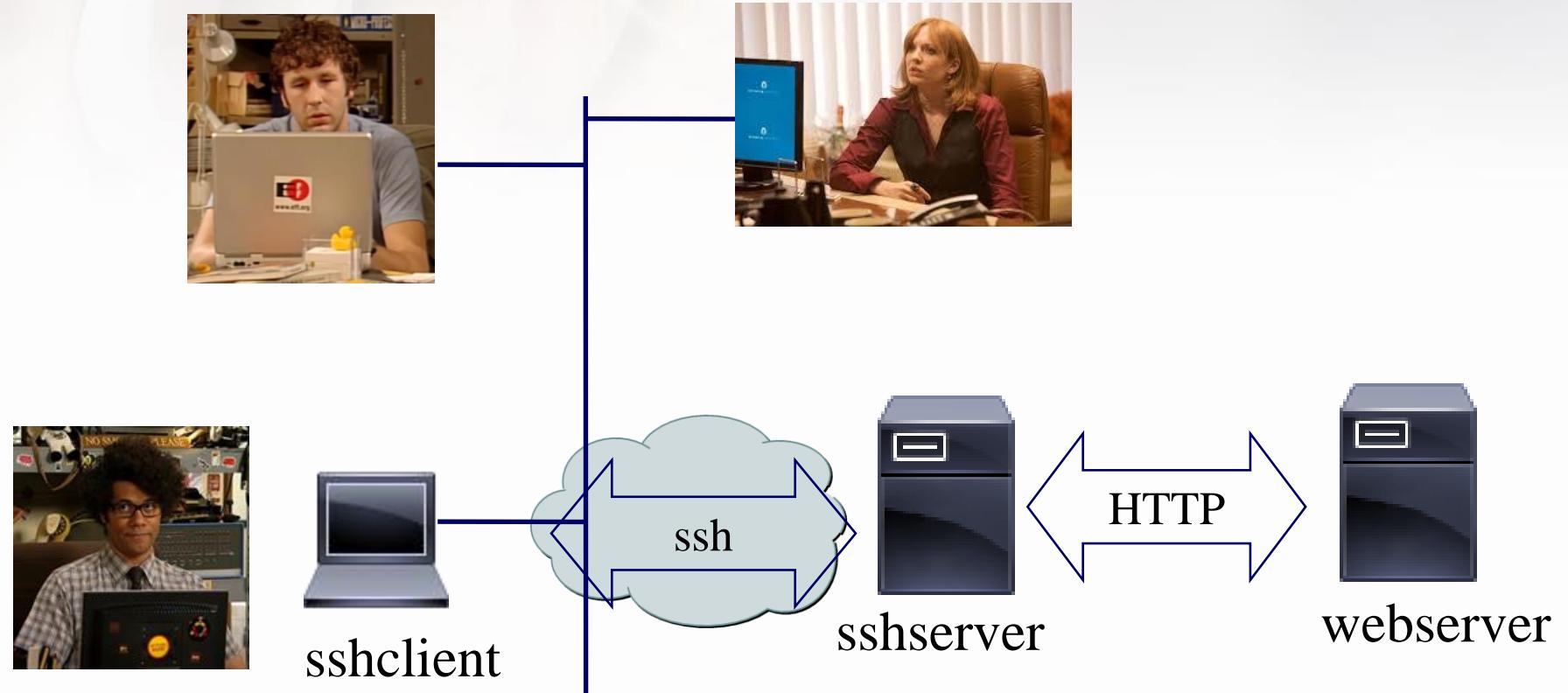
1. curl se conecta con TCP a (localhost,2001)
2. envía los requests
3. El cliente local SSH lee el puerto 2001, encripta los datos, y los envía al servidor SSH.
4. El servidor SSH desencripta los datos y los envía a 10.1.0.10:80.
5. Las respuestas son enviadas por el mismo túnel.

¿Qué debería cambiar si el servidor web funciona en el mismo host que el servidor SSH?

Para el web server ¿quién hizo el request?

# Local Port forwarding: gateway

Los otros hosts de su red también quieren usar un túnel seguro al server remoto

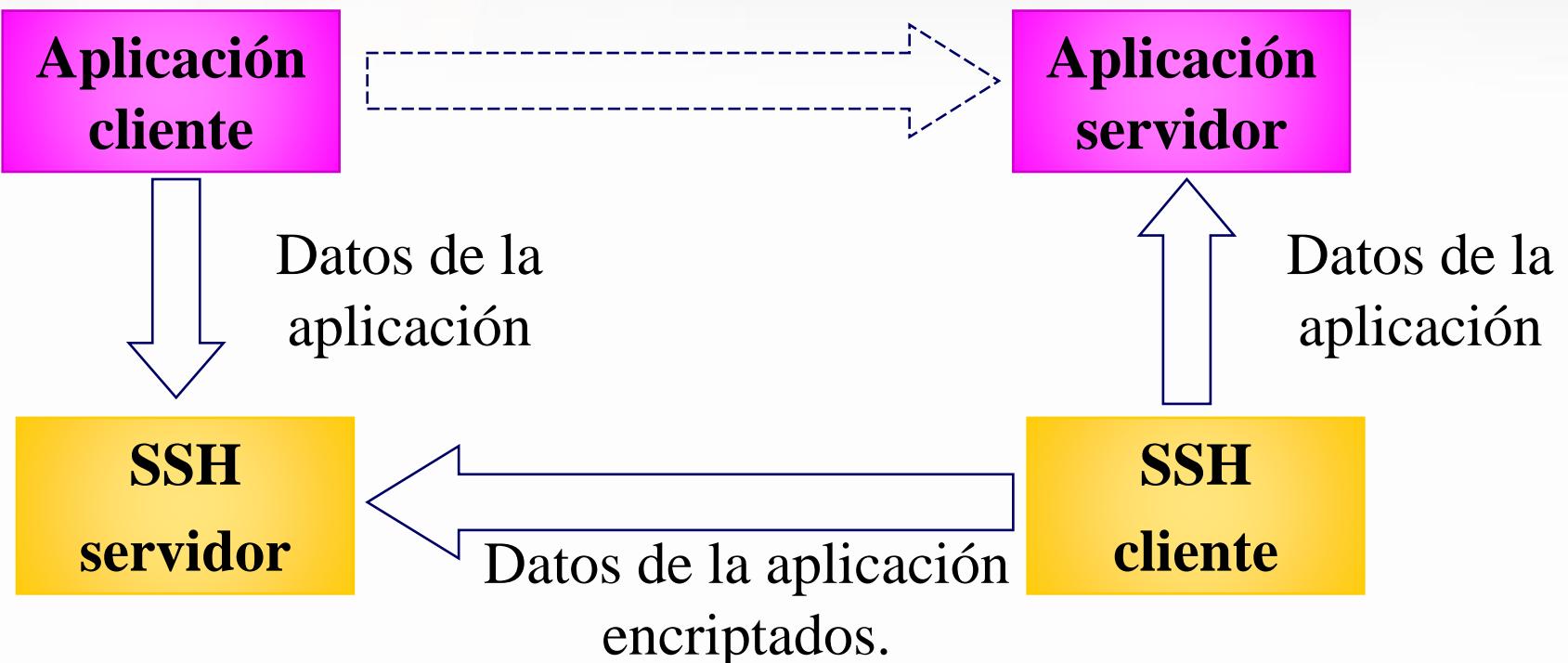


```
sshclient:~$ ssh -g -L2001:webserver:80 -l user sshserver
```

SSH

# Remote port forwarding

En remote forwarding la aplicación cliente está localizada en el servidor SSH.



# Port forwarding

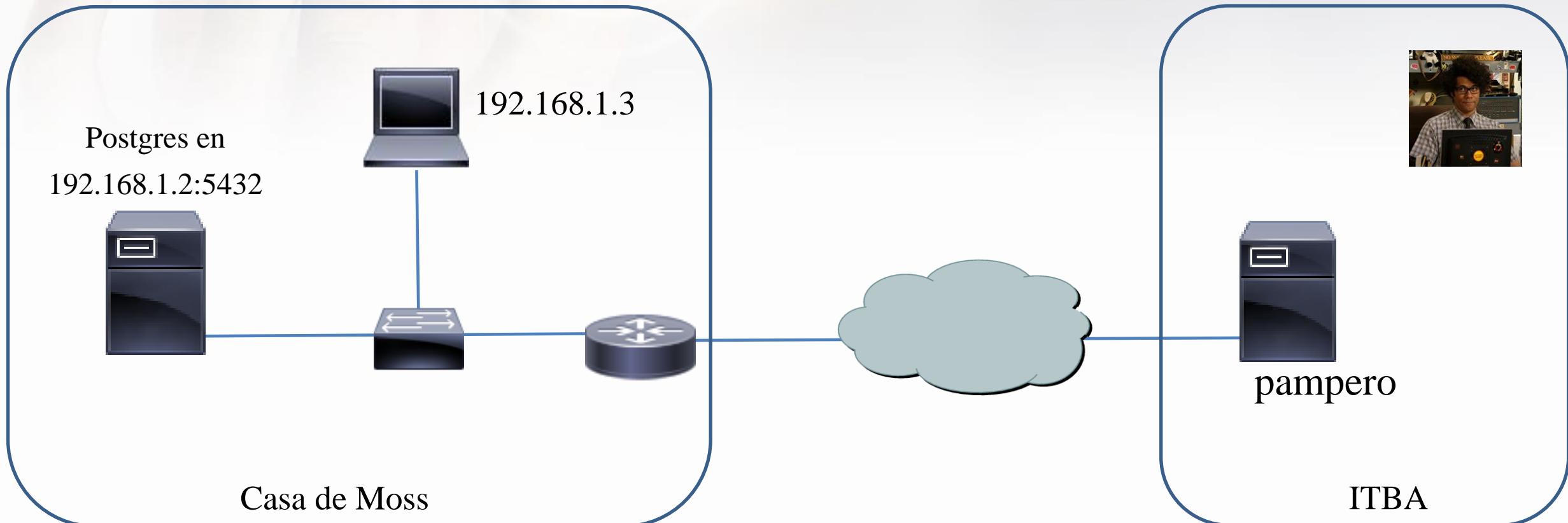
Remote forwarding.

Similar a local forwarding pero el cliente ssh es el que tiene acceso al servidor de la aplicación.

```
$ ssh -Rport:host:hostport sshServer
```

# Remote Port forwarding: ejemplo

Moss tiene en su casa un servidor Postgres y quiere conectarse desde el ITBA.

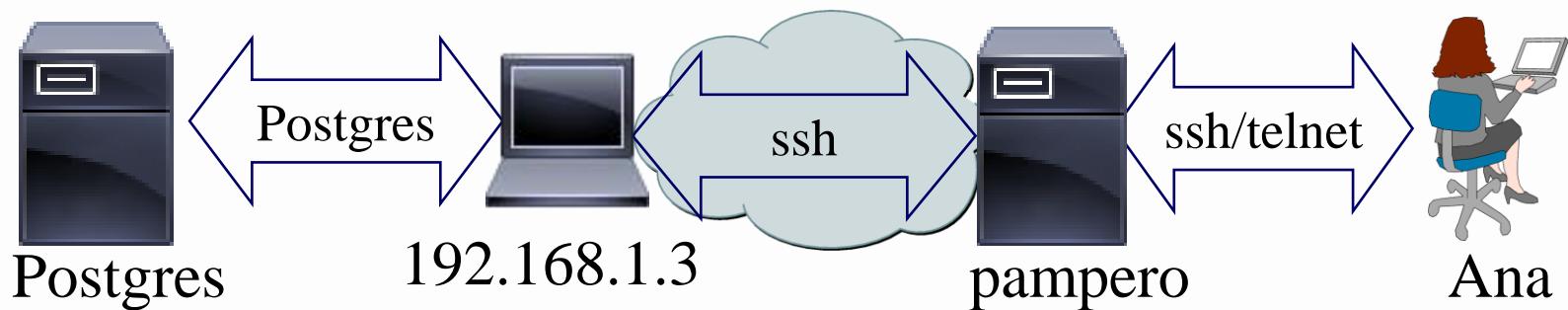


```
192.168.1.3:~$ $ ssh -R2001:192.168.1.2:5432 moss@pampero
```

SSH

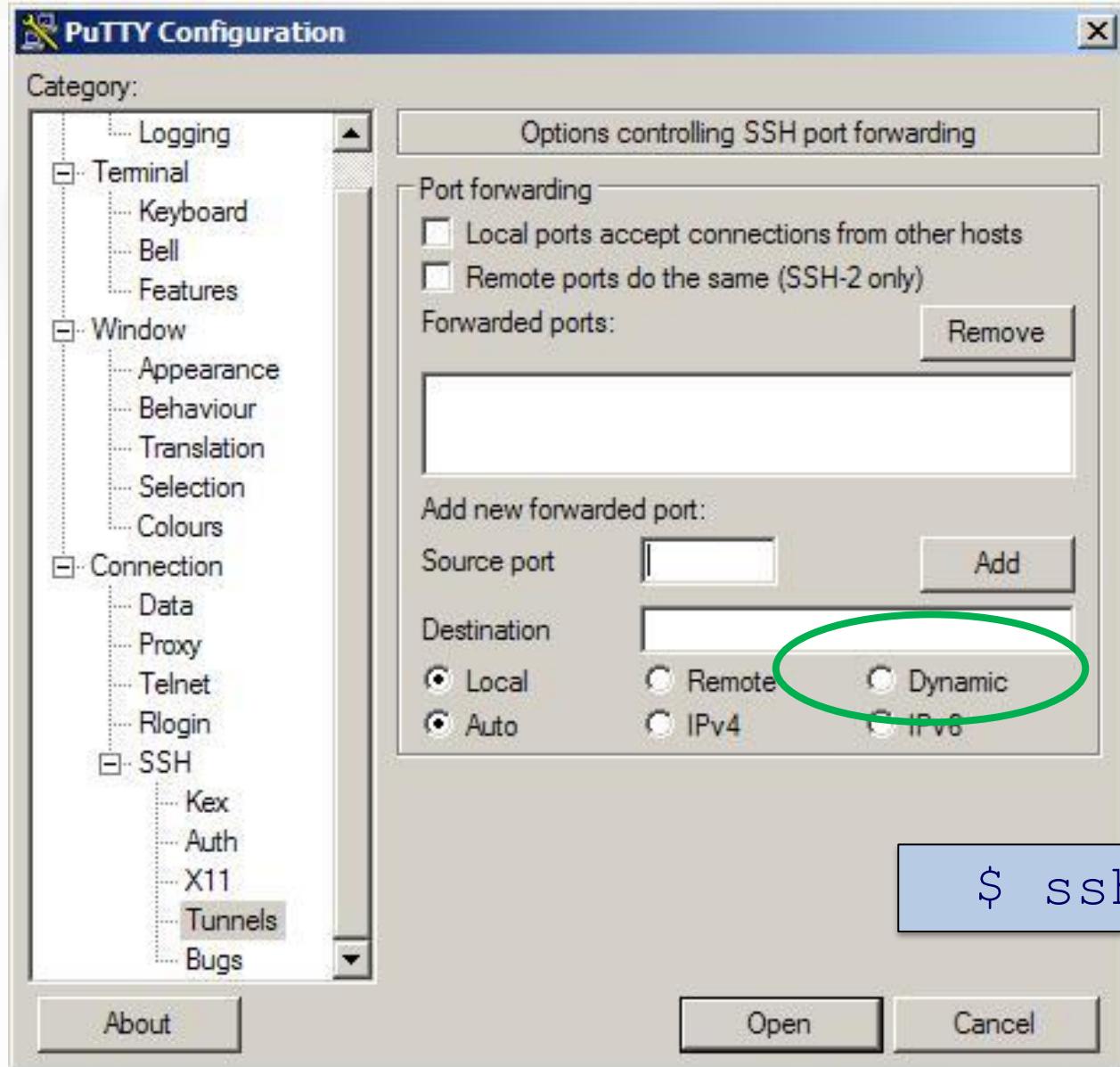
# Port forwarding remoto

El puerto remoto será el 2001, el puerto local el 5432.  
Una vez establecida la sesión, se ha creado un túnel desde el puerto 2001 del host remoto al puerto 5432 en el servidor Postgres.

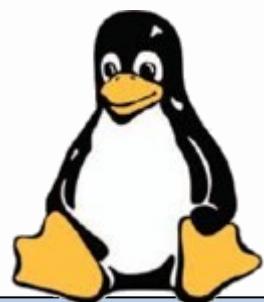


Por defecto todos los usuarios en pampero podrán usar el túnel

# Dynamic Port forwarding



```
$ ssh -D 1080 pampero.itba.edu.ar
```



# Port forwarding: BitVise

The screenshot shows the BitVise software interface. On the left is a sidebar with icons for profile management: Load profile, Save profile, Save profile as, New profile, and Close profile. The main window has a tab bar at the top with Login, Options, Terminal, Remote Desktop, SFTP, Services, C2S (selected), S2C, SSH, and About. Below the tab bar is a table titled "Forwarded ports" with columns: Enabled, Listen Interface, List. Port, Destination Host, Dest. Port, and Comment. One row is listed: Enabled (checked), Listen Interface (127.0.0.1), List. Port (8088), Destination Host (localhost), Dest. Port (8080). At the bottom are buttons for Add, Edit, Remove, and a checkbox for "Accept server-configured port forwardings". A "Help" link is also present.

Enabled	Listen Interface	List. Port	Destination Host	Dest. Port	Comment
<input checked="" type="checkbox"/>	127.0.0.1	8088	localhost	8080	

Add Edit Remove  Accept server-configured port forwardings [Help](#)

SSH

# Port forwarding: Termius

The screenshot shows the 'Edit Rule' screen in the Termius app. The top navigation bar includes a back arrow, the text 'Edit Rule', and a green 'SAVE' button. Below the navigation is a tab bar with three options: 'Local' (which is selected and highlighted in blue), 'Remote', and 'Dynamic'. The main content area contains the following fields:

- Label:** A text input field containing the value 'Belog'.
- Host from \***: A text input field containing the value 'Belog'.
- Hosts →**: A blue button with a right-pointing arrow, indicating a relationship or mapping.
- Port From \***: A text input field containing the value '15432'.
- Host to \***: A text input field containing the value 'localhost'.
- Port to \***: A text input field containing the value '5432'.
- Bind address, 127.0.0.1 by default**: A text input field containing the value '127.0.0.1'.

# Autenticación



Existen tres formas de autenticar un cliente SSH:

- Claves
- Claves públicas
- Basada en el host

# Autenticación por claves



Es la forma más común y controla que el nombre de usuario y clave sean válidos, generalmente usando el archivo /etc/passwd o /etc/shadow.

Para deshabilitar este método editar /etc/ssh/sshd\_config

```
PasswordAuthentication no
```

# Autenticación

## Claves públicas

Requiere que cada usuario genere un archivo con su clave pública y otro con su clave privada.

La clave pública debe ser copiada al servidor y la clave privada debe estar en el host cliente.

En el archivo sshd\_config debe figurar

```
PubkeyAuthentication yes  
AuthorizedKeysFile .ssh/authorized_keys
```

# Autenticación

Pasos a seguir en el host cliente Linux

```
user@server:~$ ssh-keygen -d
```

Generating public/private dsa key pair.

Enter file in which to save the key

(/home/.../.ssh/id\_dsa):

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Your identification has been saved in id\_dsa

Your public key has been saved in id\_dsa.pub

Ahora el usuario dispone de archivos con clave pública y clave privada. Debe conservar la clave privada en el o los hosts origen y almacenar la clave pública en el o los hosts destino.

# Autenticación



Pasos a seguir en el host servidor: **Copiar el archivo .pub**

```
user@server:~$ cd  
user@server:~$ cat id_dsa.pub >>.ssh/authorized_keys  
user@server:~$ rm id_dsa.pub
```

# Autenticación

## Usar clave pública con Putty y OpenSSH

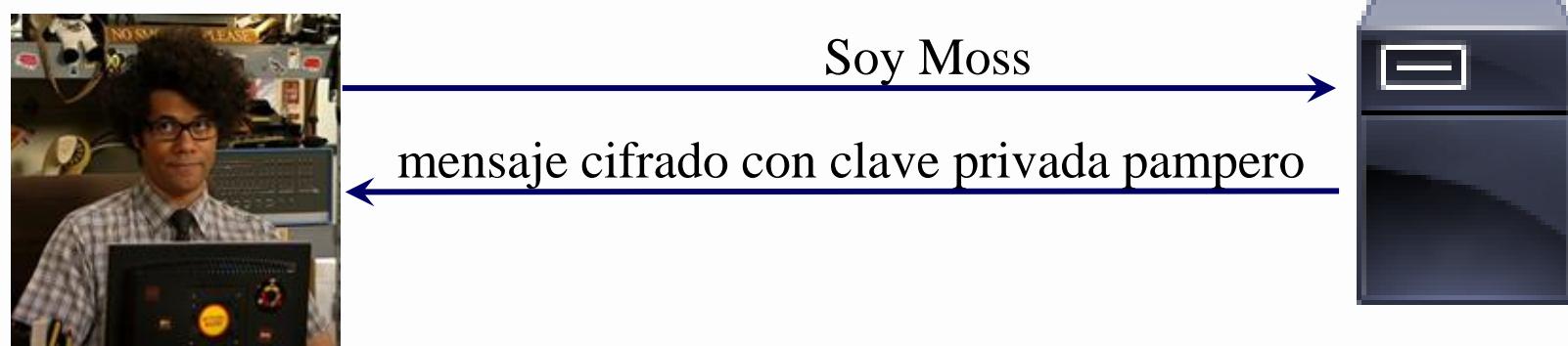
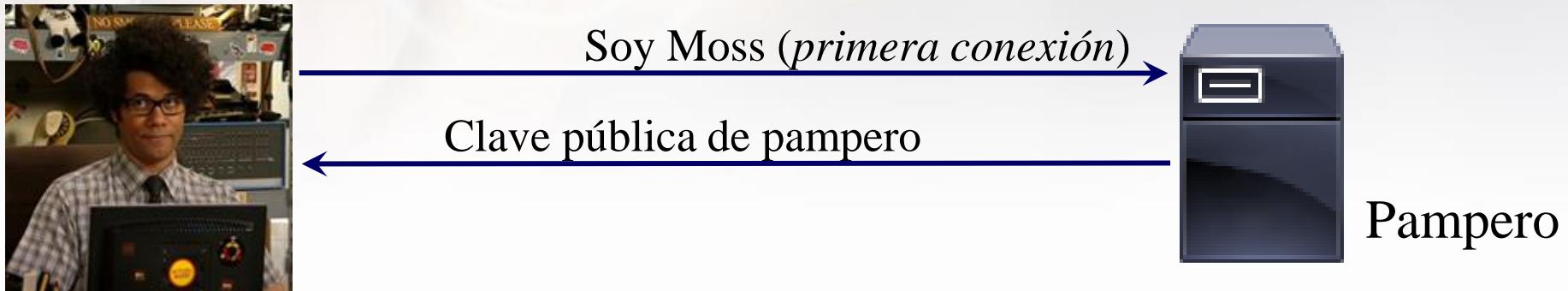
1. Ejecutar el programa puttygen para generar las claves en el host cliente
2. Indicar en Putty, dentro de la opción Connection / SSH / Auth el archivo que contiene la clave privada
3. Agregar la clave en el servidor, de la misma forma que en el ejemplo anterior.

En caso de poseer archivos de clave privada generada por ejemplo con OpenSSH se puede utilizar puttygen para importar dicha clave.

# Autenticación



El cliente debe autenticar al servidor (*Server authentication*).



# Modelo de capas SSH



# Modelo de capas SSH

- ◆ Transporte (SSH Transport Layer Protocol)
  - ◆ Provee autenticación de servidor, privacidad e integridad
  - ◆ Opcionalmente puede comprimir datos
  - ◆ Utiliza los servicios de TCP
- ◆ Autenticación (SSH User Authentication Protocol)
  - ◆ Provee autenticación del lado del cliente
- ◆ Conexión (SSH Connection Protocol)
  - ◆ Se encarga de multiplexar el túnel seguro en los distintos canales lógicos
  - ◆ Algunos canales lógicos posibles
    - ◆ sesiones de shell seguras
    - ◆ TCP port forwarding
    - ◆ Conexiones X11

# Conexión SSH



# Material de lectura

[https://www.climagic.org/tutorials/SSH\\_Tutorial\\_for\\_Linux.php](https://www.climagic.org/tutorials/SSH_Tutorial_for_Linux.php)



# Sockets bloqueantes

# Aplicaciones de red

- ◆ Comunicación entre procesos
  - ◆ En un mismo host: definido por el S.O.
  - ◆ En diferentes hosts: intercambio de mensajes
- ◆ Proceso cliente: proceso que inicia la conexión
- ◆ Proceso servidor: proceso que espera conexiones

# Aplicaciones de red

- ◆ Se debe poder identificar los procesos
- ◆ Cada host se identifica por IP
- ◆ Muchos procesos corriendo en un host
- ◆ Para identificar un proceso se agrega un número de puerto
  - ◆ 80: HTTP server
  - ◆ 5432: PostgreSQL

# Aplicaciones de red

- ◆ El protocolo de nivel de aplicación debe definir:
  - ◆ Tipos de mensaje que se intercambian
  - ◆ Sintaxis de los mensajes
  - ◆ Semántica de los mensajes
  - ◆ Reacción frente a errores

# Interfaces de protocolos de transporte

El estándar TCP/IP sugiere que las API contemplen las siguientes operaciones básicas:

- ◆ Reservar los recursos locales para la comunicación
- ◆ Especificar *endpoints* origen y destino
- ◆ Iniciar una conexión (desde el cliente)
- ◆ Enviar un datagrama (desde el cliente)
- ◆ Esperar una conexión (desde el servidor)
- ◆ Enviar y recibir datos
- ◆ Poder determinar cuándo arriba información
- ◆ Generar datos urgentes
- ◆ Manejar el arribo de datos urgentes

# Interfaces de protocolos de transporte

Sólo han tenido aceptación unas pocas API para los protocolos TCP/IP:

- ◆ socket API (Berkeley), diseñada para UNIX.
- ◆ Windows Socket (Microsoft).
- ◆ Transport Layer Interface TLI (AT&T), diseñada para System V UNIX.

# Socket API

Berkeley sockets fue diseñado en los 80 para UNIX.

Utiliza system calls ya existentes cuando sea posible.

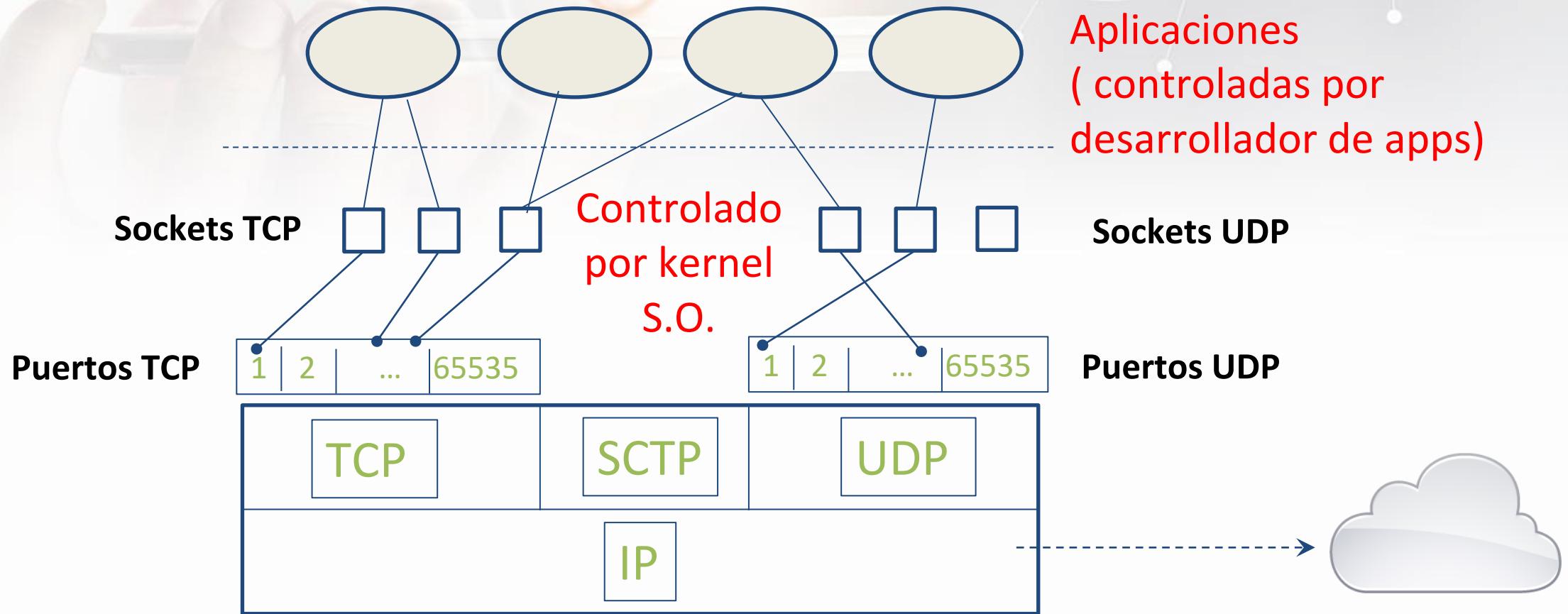
Provee funciones generales que soportan comunicaciones con distintos protocolos, no sólo TCP/IP.

Las funciones trabajan con *familias* de protocolos. Indicando el usuario qué tipo de familia (p.e. TCP/IP) y qué tipo de servicio requiere (con conexión o sin conexión).

# Sockets: conceptos básicos

- ◆ El servidor debe estar en ejecución ANTES de que el cliente intente conectarse
- ◆ El servidor debe tener un socket (como una «puerta») abierto, por el cual recibir y enviar mensajes (segmentos)
- ◆ Los clientes necesitan un socket
- ◆ Los clientes necesitan conocer IP y número de puerto del socket del servidor

# Aplicaciones de red : Sockets



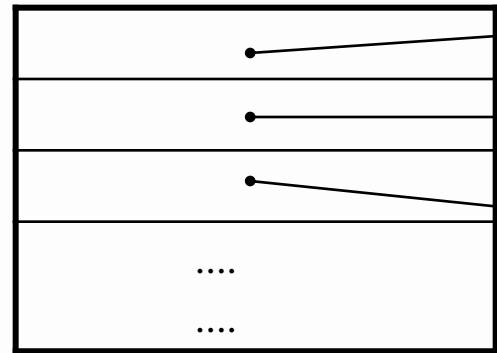
# Socket API



Un proceso que necesita realizar operaciones de E/S utiliza el *system call open* que crea un *file descriptor*, el cual usa para acceder al archivo. El S.O. implementa *file descriptors* como un vector de punteros a estructuras de datos.

## Espacio del usuario

tabla de descriptores  
(una por proceso)



## Recursos del kernel

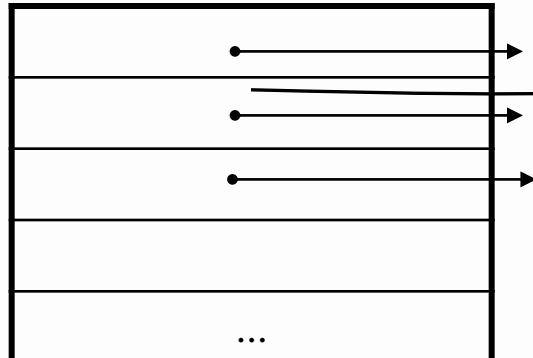
- Estructura de datos para *file id 0*
- Estructura de datos para *file id 1*
- Estructura de datos para *file id 2*

# Socket API

La API de sockets agrega una nueva abstracción para las comunicaciones en red, el *socket*. Al igual que con los archivos, cada socket es identificado por un *handle* y la información relacionada es almacenada en la tabla de *file descriptors*.

## Espacio del usuario

tabla de descriptores  
(una por proceso)



## Recursos del kernel

**family: PF\_INET**

**service: SOCK\_STREAM**

**Local IP:**

**Local port:**

**Remote IP:**

...

# Creación de un socket

```
#include <sys/socket.h>

// Returns: file (socket) descriptor if OK, -1 on error
int socket(int domain, int type, int protocol);
```

domain: naturaleza de la comunicación y formato de la dirección

- AF\_INET
- AF\_INET6
- AF\_UNIX
- AF\_UNSPEC

# Creación de un socket

```
#include <sys/socket.h>

// Returns: file (socket) descriptor if OK, -1 on error
int socket(int domain, int type, int protocol);
```

type: caracteriza el tipo de comunicación

- SOCK\_DGRAM
- SOCK\_RAW
- SOCK\_SEQPACKET
- SOCK\_STREAM

# Creación de un socket

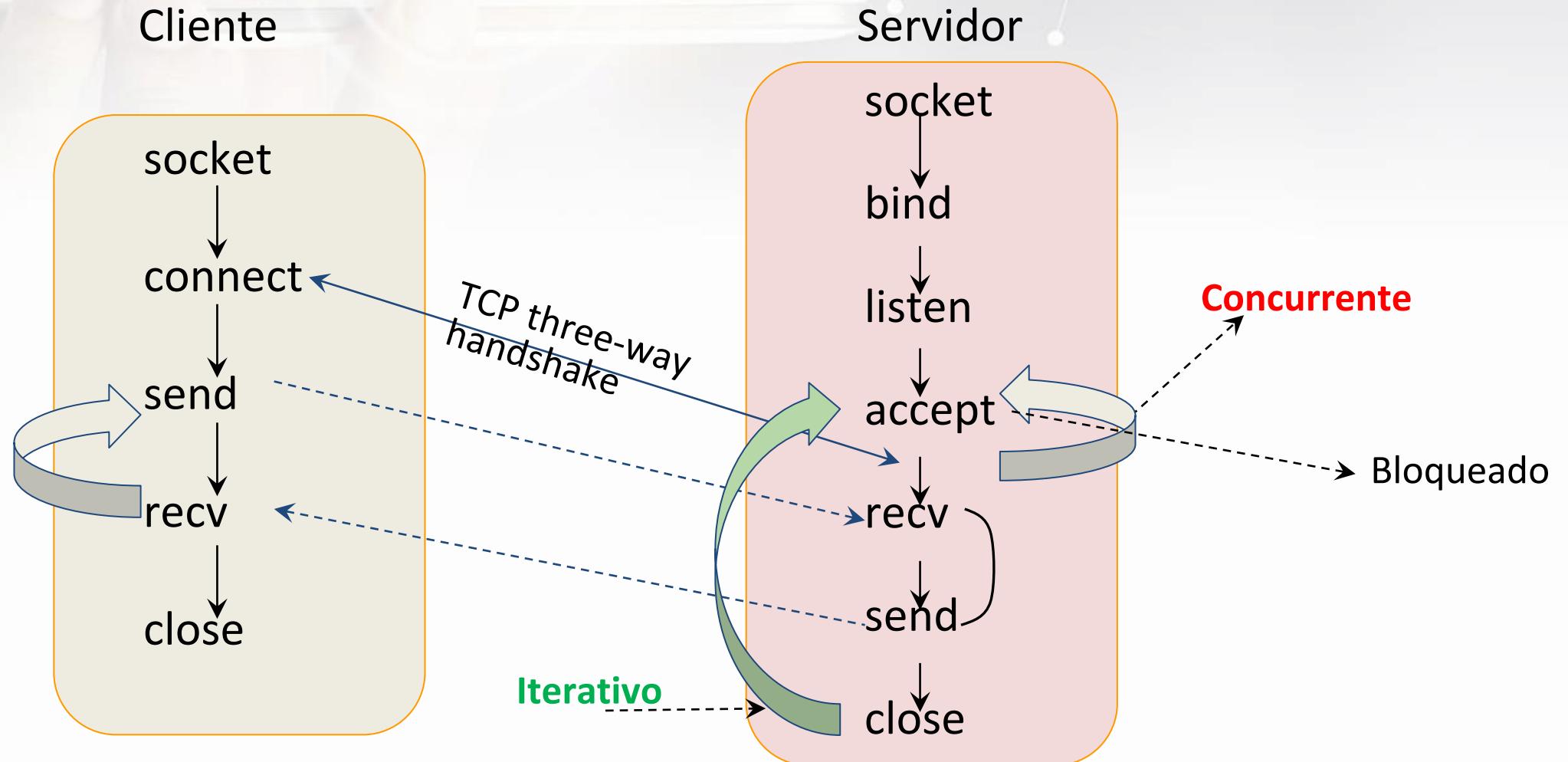
```
#include <sys/socket.h>

// Returns: file (socket) descriptor if OK, -1 on error
int socket(int domain, int type, int protocol);
```

protocol: generalmente cero para que se seleccione el protocolo por defecto de acuerdo a *domain* y *type*

- IPPROTO\_IP
- IPPROTO\_IPV6
- IPPROTO\_ICMP
- IPPROTO\_RAW
- IPPROTO\_TCP
- IPPROTO\_UDP
- IPPROTO\_SCTP

# Socket API: ejemplo TCP



# Diseño de servidores



## Algoritmo para servidor iterativo no orientado a conexión

1. Crear un socket y ligarlo (*bind*) a un puerto
2. Leer un datagrama *request* de un cliente
3. Enviar datagrama/s como respuesta
4. Volver al punto 2

Un datagrama es un mensaje auto-suficiente

# Diseño de servidores



## Algoritmo para servidor iterativo orientado a conexión

1. Crear un socket y ligarlo (*bind*) a un puerto
2. Aceptar un pedido de conexión a través del socket
3. Obtener un nuevo socket para la conexión
4. Leer un *request* del cliente
5. Enviar una respuesta
6. Si el cliente no finalizó, volver al punto 4
7. Cerrar el socket creado en punto 3
8. Volver al punto 2

# Socket I/O

La comunicación en un socket es bidireccional. Se puede deshabilitar I/O con *shutdown*

```
#include <sys/socket.h>  
  
int shutdown(int sockfd, int how);
```

Posibles valores para *how*

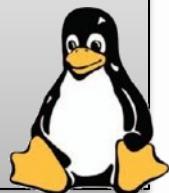
- SHUT\_RD
- SHUT\_WR
- SHUT\_RDWR

# Formato de direcciones

Una dirección identifica un *socket endpoint*. El formato es específico para cada familia, y se castean a un formato genérico.

```
struct sockaddr {  
    sa_family_t sa_family;      /* address family */  
    char     sa_data[...];       /* variable-length  
address */  
    ...  
};
```

```
struct sockaddr {  
    sa_family_t sa_family;  
    char     sa_data[14];  
};
```



```
struct sockaddr {  
    unsigned char sa_len;  
    sa_family_t sa_family;  
    char     sa_data[];  
};
```



FreeBSD

# Formato de direcciones: ejemplo IPv4

```
struct in_addr {  
    in_addr_t s_addr;          /* IPv4 address */  
};  
struct sockaddr_in {  
    sa_family_t sin_family;      /* address family: unsiged short */  
    in_port_t sin_port;          /* port number: uint16_t */  
    struct in_addr sin_addr;      /* IPv4 address: uint32_t */  
    unsigned char sin_zero[8];    /* Linux: relleno */  
};
```

```
int bind(int socket, const struct sockaddr *address, socklen_t address_len);
```

```
struct sockaddr_in myAddress;  
...  
bind( socket, (struct sockaddr *) (&myAddress), sizeof(myAddress));
```

# Formato de direcciones: ejemplo IPv6

```
struct in6_addr {  
    uint8_t s6_addr[16]; /* IPv6address */  
};  
struct sockaddr_in6 {  
    sa_family_t sin6_family; /* address family: unsiged short */  
    in_port_t sin6_port; /* port number: uint16_t */  
    uint32_t sin6_flowinfo; /* traffic class and flow info */  
    struct in6_addr sin6_addr; /* IPv6 address: */  
    uint32_t sin6_scope_id;  
};
```

```
struct sockaddr_in6 myAddress;  
...  
bind( socket, (struct sockaddr *) (&myAddress), sizeof(myAddress));
```

# Resolución de nombres

Un cliente debe especificar la dirección de un servidor utilizando la estructura `sockaddr_in`. ¿Qué sucede si el programa cliente conoce sólo el FQDN del servidor?

**inet\_addr**: convierte de notación con puntos ("200.132.2.15") a decimal.

**gethostbyname**: dado un string que contiene el FQDN de un host retorna una estructura que contiene –entre otras cosas– el IP del host en forma decimal.

**getaddrinfo**: más completa que la anterior.

```
#include <sys/socket.h>
#include <netdb.h>

int getaddrinfo(const char *restrict host,
               const char *restrict service,
               const struct addrinfo *restrict hint,
               struct addrinfo **restrict res);
```

```
void freeaddrinfo(struct addrinfo *ai);
```

# Resolución de nombres

La estructura `addrinfo` contiene al menos los siguientes campos

```
struct addrinfo {  
    int             ai_flags;          /* customize behavior */  
    int             ai_family;         /* address family */  
    int             ai_socktype;       /* socket type */  
    int             ai_protocol;       /* protocol */  
    socklen_t       ai_addrlen;        /* length in bytes of address */  
    struct sockaddr *ai_addr;          /* address */  
    char            *ai_canonname;     /* canonical name of host */  
    struct addrinfo *ai_next;          /* next in list */  
};
```

# Resolución de nombres

## Posibles valores del campo ai\_flags

Flag	Description
AI_ADDRCONFIG	Query for whichever address type (IPv4 or IPv6) is configured.
AI_ALL	Look for both IPv4 and IPv6 addresses (used only with AI_V4MAPPED).
AI_CANONNAME	Request a canonical name (as opposed to an alias).
AI_NUMERICHOST	The host address is specified in numeric format; don't try to translate it.
AI_NUMERICSERV	The service is specified as a numeric port number; don't try to translate it.
AI_PASSIVE	Socket address is intended to be bound for listening.
AI_V4MAPPED	If no IPv6 addresses are found, return IPv4 addresses mapped in IPv6 format.

Figure 16.7 Flags for addrinfo structure

Ver  
addrinfo.c

# Ejemplos TCP

- ◆ Cliente ECHO: `tcpEchoClient.c`
- ◆ Servidor ECHO iterativo: `tcpEchoServer.c`

# Debug de procesos

**Podemos usar comandos para depurar procesos, por ejemplo**

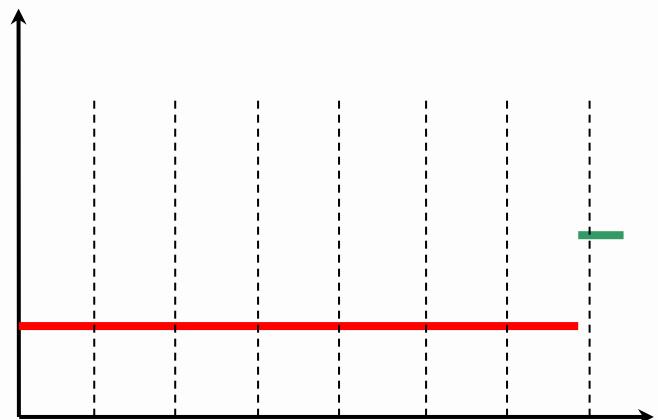
- ◆ truss en Solaris y FreeBSD
- ◆ strace en GNU Linux
- ◆ dtruss en OSX

# Ejemplos UDP

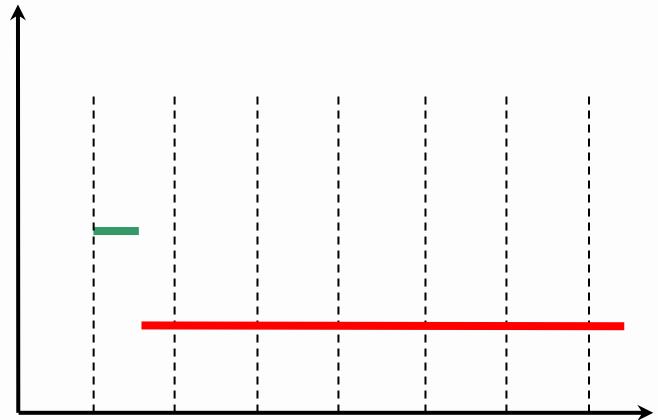
- ◆ Cliente ECHO: udpEchoClient.c
- ◆ Servidor ECHO: udpEchoServer.c

# Servidores concurrentes vs iterativos

Ejemplo: un servidor ftp recibe un pedido para bajar un archivo de 200 MB e inmediatamente después otro pedido para bajar un archivo de 50 bytes.



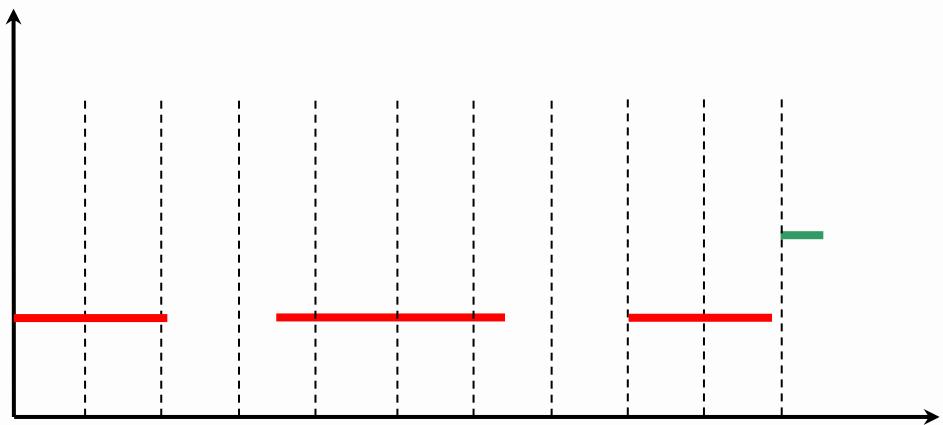
Servidor  
iterativo



Servidor  
concurrente

# Servidores concurrentes vs iterativos

El gráfico anterior contemplaba un uso ideal del “canal” al 100 %, y que el proceso no fuera bloqueado.



Servidor  
iterativo

# Diseño de servidores: multitasking

## Algoritmo para servidor concurrente orientado a conexión

### Master (Proceso Padre)

1. Crear un socket y ligarlo (*bind*) a un puerto
2. Aceptar un pedido de conexión a través del socket
3. Crear un proceso hijo para atender la conexión
4. Volver al punto 2

### Slave (Proceso Hijo)

1. Atender la conexión pasada por el master
2. Interactuar con el cliente
3. Cerrar la conexión y finalizar el proceso

# Multitasking

Ejemplo de servidores concurrentes orientado a conexión:  
tcpEchoServer-Fork.c

# Multiplexing: select

32

- ◆ Todos los ejemplos vistos manejan I/O en un solo canal
- ◆ Un server puede manejar varios canales
  - ◆ escuchar en distintos puertos
  - ◆ sobre TCP, UDP, SCTP
  - ◆ superserver: un único programa para ECHO, Daytime, Chargen
- ◆ `select ()` recibe una lista de descriptores y se bloquea hasta que alguno esté listo para I/O

# Multiplexing: select

```
int select(int n,  
          fd_set *readfds,  
          fd_set *writefds,  
          fd_set *exceptfds,  
          struct timeval *timeout);
```

Cantidad máxima de file descriptors

Conjuntos de *fd* donde se marcará el que esté listo para read/write y condiciones “excepcionales”.

Tiempo máximo de bloqueo

# Multiplexing: select

34

Los conjuntos de *fds* son un “array” de bits y existen macros para manipularlos

- ◆ FD\_ZERO(fd\_set \*set)
- ◆ FD\_SET(int fd, fd\_set \*set)
- ◆ FD\_CLR(int fd, fd\_set \*set)
- ◆ FD\_ISSET(int fd, fd\_set \*set)

# Multiplexing: select

35

- ❑ Ejemplo (no muy afortunado) de servidor TCP ECHO con select
- ❑ TCPEchoSelect original.c extraido de  
<http://www.binarytides.com/multiple-socket-connections-fdset-select-linux/>

# Multiplexing: pselect

36

Usar select implica un problema si queremos esperar a que o bien un file descriptor esté listo o recibamos una señal

```
int main(int argc, char *argv[])
{
    fd_set readfds;
    struct sigaction sa;
    int nfds, ready;

    sa.sa_handler = handler;      /* Establish signal handler */
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = 0;
    sigaction(SIGINT, &sa, NULL);
/* ... */
    ready = select(nfds, &readfds, NULL, NULL, NULL);
/* ... */
```

<https://lwn.net/Articles/176911/>

# Multiplexing: pselect

pselect es similar a select pero además recibe un conjunto de señales que deberían ser bloqueadas durante la llamada a pselect

```
int pselect(int n, fd_set *readfds, fd_set *writefds, fd_set *exceptfds,  
           const struct timespec *timeout, const sigset_t *sigmask);
```

<https://lwn.net/Articles/176911/>

# Multiplexing: poll y epoll

38

A diferencia de select, la función poll recibe un vector que contiene únicamente los files id a "escuchar"

```
int poll (struct pollfd *fds, unsigned int nfds, int timeout);
```

```
struct pollfd {  
    int fd;  
    short events;  
    short revents;  
};
```

Ver ejemplo de uso y diferencias en

<https://devarea.com/linux-io-multiplexing-select-vs-poll-vs-epoll/#.Y0XLy2zMJhE>



# Sockets no bloqueantes

Socket options

Signals

Nonblocking i/o

# Socket options

```
#include <sys/socket.h>

int getsockopt(int socket, int level,
               int optionName,
               void *restrict optionValue,
               socklen_t *restrict optionLen);

int setsockopt(int socket, int level,
               int optionName,
               const void *optionValue,
               socklen_t optionLen);
```

# Socket options: ejemplo

```
/* Do I linger on close? */
int option_len = sizeof(struct linger);
struct linger ling;
int rc = getsockopt(
    s, SOL_SOCKET, SO_LINGER, (void *) &ling, &option_len);
if (rc == 0) {
    if (option_len == sizeof(ling)) {
        if (ling.l_onoff)
            /* yes I linger */
        else
            /* no I do not */
    }
}
```

# Socket options

- Level
  - SOL\_SOCKET: opción independiente del protocolo
  - IPPROTO\_TCP: relativo a TCP
  - IPPROTO\_IP: relativo al protocolo de red IP v4
  - IPPROTO\_IPV6: relativo a IP v6

# Socket options: listado incompleto

- SOL\_SOCKET
  - SO\_BROADCAST: 0 ó 1, si permite broadcast
  - SO\_KEEPALIVE
  - SO\_RCVBUF: tamaño en bytes del buffer de entrada
  - SO\_RCVLOWAT: Mínimo tamaño en bytes para procesar el input.
  - SO SNDLOWAT: ídem para la salida
  - SO\_SNDBUF: tamaño del buffer de salida
  - SO\_RCVTIMEO: máximo timeout para operaciones de lectura
  - SO\_LINGER: 0 para enviar RST al hacer close() -> "*connection reset by peer*"
  - SO\_REUSEADDR: en caso de abortar el programa, reusar dirección y Puerto
  - SO\_REUSEPORT

# Socket options: listado incompleto

- **IPPROTO\_TCP**
  - TCP\_MAX: segundos entre mensajes keepalive
  - TCP\_NODELAY: 0,1
- **IPPROTO\_IP**
  - IP\_TTL: [0,255]
  - IP\_MULTICAST\_LOOP: habilita al socket multicast a recibir paquetes enviados por él.
  - IP\_ADD\_MEMBERSHIP
  - IP\_DROP\_MEMBERSHIP
- **IPPROTO\_IPV6**
  - IPV6\_V6ONLY

# SIGNALS

- Mecanismo para notificar un proceso que ocurrió un evento
  - Timeout
  - Ctrl-c
  - Etc.
- El proceso puede
  - Ignorar la señal
  - Terminar abruptamente
  - Interrumpir su ejecución normal y ejecutar una “rutina de manejo de señal”
  - Bloquear la señal

# SIGNALS



Señal	Evento	Acción por defecto
SIGALARM	Expiró un timer	Termina ejecución
SIGCHILD	“exit” de proceso hijo	Ignorar
SIGINT	Carácter interrupción ( CTRL-C)	Terminar
SIGIO	Socket listo para I/O	Ignorar
SIGPIPE	Intento de escribir en socket cerrado	Terminar
SIGSEGV	“Segmentation fault”	Terminar
SIGTERM	Terminación “amable” (kill)	Terminar
SIGKILL	Terminación forzosa	Terminar

# SIGNALS: manejar el comportamiento

```
int sigaction(int signum, const struct sigaction *act,  
             struct sigaction *oldact);
```

```
struct sigaction {  
    void        (*sa_handler)(int);  
    sigset_t   sa_set; // Añadido  
    int         sa_flags;  
};
```

SIG\_IGN, SIG\_DFL o función que “capture” la señal

Señales a bloquear mientras se ejecuta sa\_handler

# SIGNALS

- int sigemptyset(sigset\_t \* set);
- int sigfillset(sigset\_t \* set);
- int sigaddset(sigset\_t \* set, int signal);
- int sigdelset(sigset\_t \* set, int signal);

```
sigset_t mask;  
sigemptyset (&mask);  
sigaddset (&mask, SIGTERM);  
sigaddset (&mask, SIGINT);  
struct sigaction handler = {SIG_IGN, &mask, 0};  
if (sigaction(SIGINT, &handler, NULL) < 0) {  
    perror (" failed trying to ignore SIGINT ");  
    return 1;  
}
```

# SIGNALS: ejemplo

```
int main(int argc, char *argv[]) {  
    struct sigaction handler; // Signal handler specification structure  
  
    // Set InterruptSignalHandler() as handler function  
    handler.sa_handler = InterruptSignalHandler;  
  
    // Create mask that blocks all signals  
    if (sigfillset(&handler.sa_mask) < 0)  
        ERROR("sigfillset() failed");  
    handler.sa_flags = 0; // No flags  
  
    // Set signal handling for interrupt signal  
    if (sigaction(SIGINT, &handler, 0) < 0)  
        ERROR("sigaction() failed for SIGINT");
```

# SIGNALS: ejemplo (cont.)

```
for (;;)  
    pause(); // Suspend program until signal received  
  
exit(0);  
}  
  
void InterruptSignalHandler(int signalType) {  
    puts("Interrupt Received.  Exiting program.");  
    exit(1);  
}
```

# SIGNALS: anidamiento

- ¿Qué sucede si se produce una señal y se está atendiendo otra?
- El manejo es pospuesto hasta que se termina de manejar la señal actual
- ¡Las señales no se encolan !
- Ejemplo:

```
void InterruptSignalHandler(int signalType) {  
    puts("Interrupt Received.\n");  
    sleep(3);  
}
```

# SIGNALS y sockets

Si una señal es enviada mientras el programa está bloqueado en un “socket call” (`recv()`, `connect()`, etc.) y para esa señal se especificó un “handler”, entonces cuando el “handler” termina

- el “socket call” retorna -1
- `errno = EINTR`

```
while (1) {  
    int rc = recv (sock, buf, 1, 0);  
    if (rc == -1 && errno == EINTR)  
        continue;  
    ...  
}
```

# Blocking y timeouts

15

- ◆ Llamadas a E/S pueden bloquear por diversas razones
  - ◆ `recv()`, `recvfrom()`: no hay datos listos
  - ◆ `send()`, `sendto()`: no hay espacio en buffer de TX
  - ◆ `connect()`, `accept()`: hasta conectar
- ◆ ¿Pueden bloquearse los códigos vistos?
  - ◆ TCPEchoClient.c
  - ◆ TCPEchoServer\*.c
  - ◆ UDPEchoServer.c

# Blocking y timeouts

- ◆ ¿Y si un programa quiere hacer algo mientras espera por una conexión?
- ◆ ¿Qué sucede con datagramas UDP perdidos?

```
....  
numBytes = recvfrom(sock, buffer, MAXSTRINGLENGTH, 0,  
                     (struct sockaddr *) &fromAddr, &fromAddrLen);  
if (numBytes < 0)  
    ERROR("recvfrom() failed");  
....
```

# nonblocking i/o

17

- ◆ Para “romper” el bloqueo podemos
  - ◆ usar sockets no bloqueantes
  - ◆ I/O asincrónico
  - ◆ timeouts

# nonblocking i/o: sockets no bloqueantes

18

- ◆ Todas las llamadas a sockets hacerlas no bloqueantes
- ◆ Si la operación es inmediata retorna éxito
- ◆ Sinó, retorna error: se debe distinguir si es por error o recurso no disponible.
- ◆ Se cambia el bloqueo con `fcntl()`

```
int s = socket(PF_INET, SOCK_STREAM, 0);

fcntl(s, F_SETFL, O_NONBLOCK); // set to non-blocking
fcntl(s, F_SETFL, O_ASYNC); // set to asynchronous I/O
```

# nonblocking i/o: sockets no bloqueantes

¿Qué sucede si una función necesita bloquear para completar su acción?

- ◆ Si una función necesitaba bloquear para completarse asigna EWOULDBLOCK a errno
- ◆ UDP: send () y sendto () no bloquean (no hay buffer de salida)
- ◆ En TCP, connect () asigna EINPROGRESS a errno
- ◆ getAddrInfo bloquea hasta resolver el nombre

# nonblocking i/o: sockets no bloqueantes

```
20 numBytesRcvd = recvfrom(servSock, buffer, MAXSTRINGLENGTH, 0,  
                         (struct sockaddr *) &clntAddr, &clntLen);  
  
if (numBytesRcvd < 0) {  
    // Only acceptable error: recvfrom() would have blocked  
    if (errno != EWOULDBLOCK)  
        ERROR("recvfrom() failed");
```

```
do {  
    res = connect(soc, (struct sockaddr *)&addr, sizeof(addr));  
    if (res < 0) {  
        if (errno == EINPROGRESS) {  
            putchar('.'); continue; }  
        ERROR  
    }  
} while ( res != 0 );
```

¿ Performance ?

# nonblocking i/o: I/O asincrónico

- ◆ En vez de “polling”, que el S.O. avise cuando un “socket call” está listo.
  - ◆ El programa realiza otras tareas hasta tener que manejar señal SIGIO.
1. Usar `sigaction()` para capturar SIGIO
  2. Usar `fcntl()` para hacernos owner del socket
  3. Usar `fcntl()` para no bloquear el socket

Ver UDPEchoServer-SIGIO.c