Redes Industriales - Capa de Aplicación

Vicente González Ruiz
Depto de Arquitectura de Computadores y Electrónica
vruiz@ual.es
http://www.ace.ual.es/~vruiz/docencia

23 de octubre de 2008

Contenidos

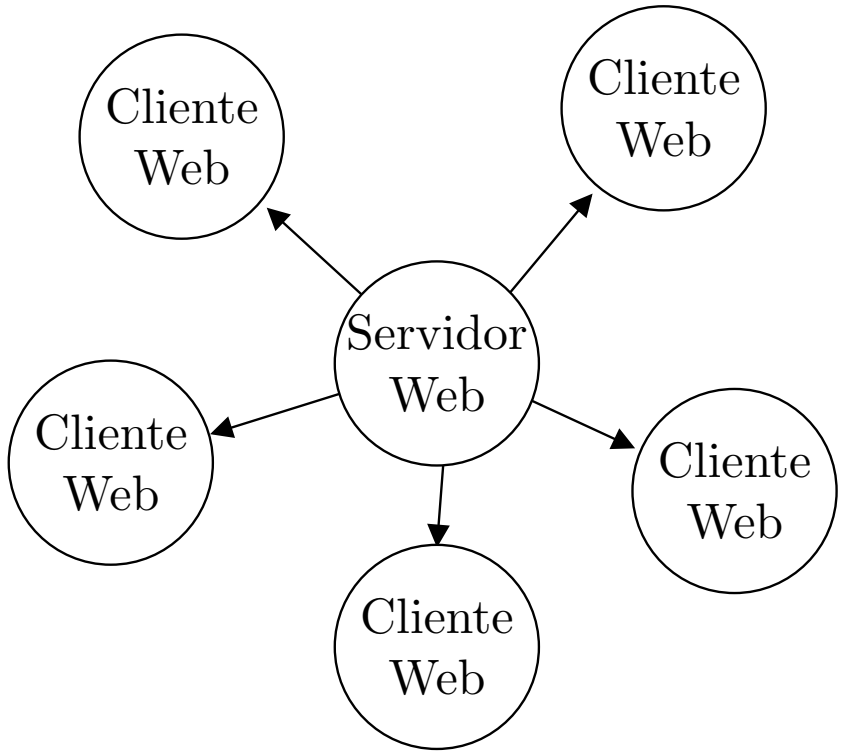
Capítulo 1

La Web

1.1. ¿Qué es la Web?

- La (World Wide) Web es una aplicación distribuida, inventada por Tim Berners-Lee [?] a principios de los 90, que permite la transmisión de información bajo demanda.

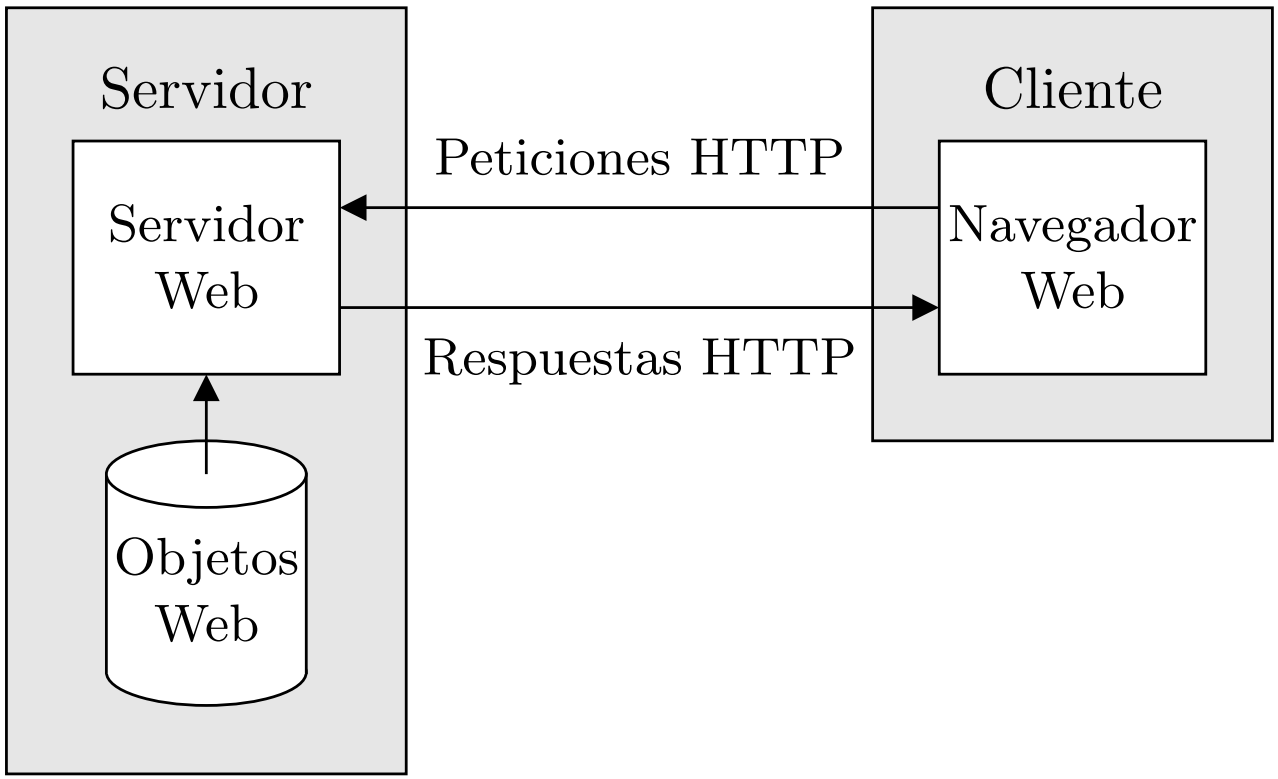
Típicamente, un servidor Web establece una comunicación de forma simultánea con muchos clientes Web. El protocolo que define el intercambio de información es el HTTP (HyperText Transfer Protocol). Este se define en los RFC's 1945 (HTTP/1.0) y 2616 (HTTP/1.1).



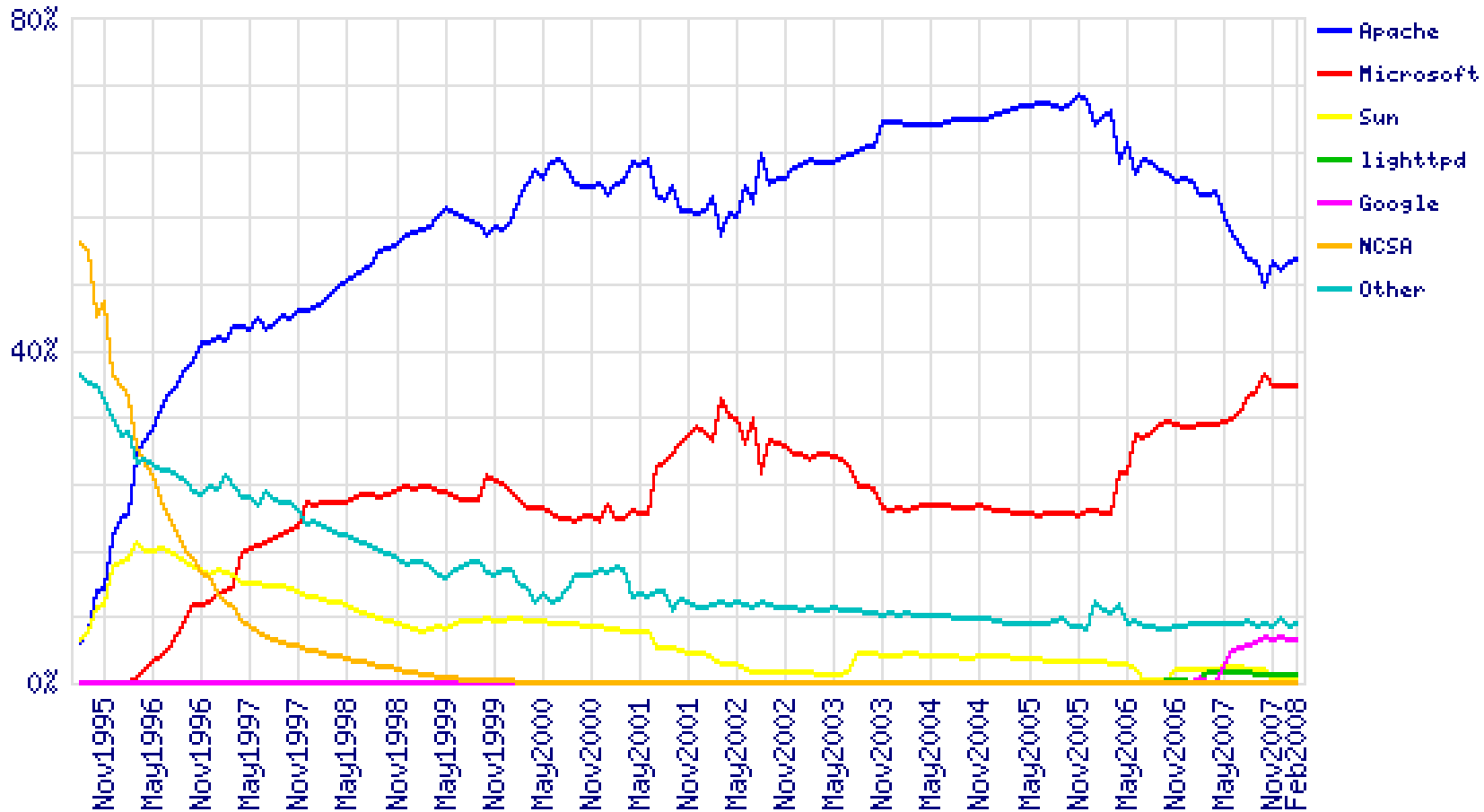
- La Web utiliza el TCP como protocolo de transporte [?].

1.2. La comunicación Web

- En una comunicación Web, un navegador Web (el cliente) realiza peticiones al servidor Web (a través del puerto 80) y éste le entrega objetos Web. Ambos tipos de mensajes se realizan según el HTTP.



- Entre los servidores Web más populares se encuentran Apache y Microsoft Internet Informacion Server (véase la URL http://news.netcraft.com/archives/web_server_survey.html).



- Un objeto Web es cualquier cosa que puede ser referenciada por su URL (Universal Resource Locator). Ejemplos típicos son las páginas Web (código HTML[1]), las imágenes GIF y las imágenes JPEG, aunque actualmente se utiliza el HTTP para transmitir cualquier tipo de archivo (con cualquier contenido).

- La estructura de una URL Web siempre es de la forma:

  http://host/camino_al_objeto_en_el_host

[1] HyperText Markup Language.

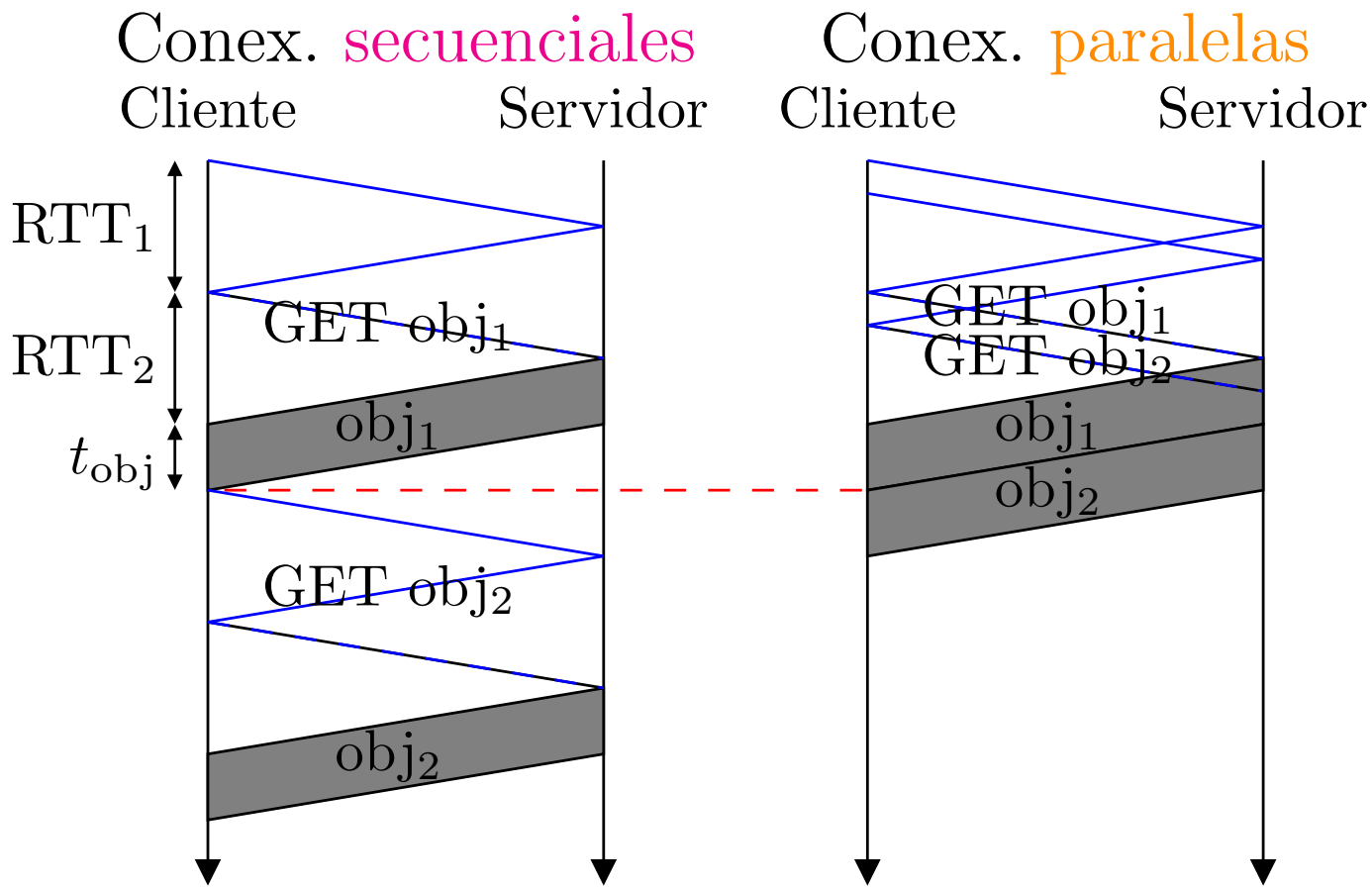
1.3. Las conexiones Web

- Cuando el cliente establece una conexión HTTP con un servidor Web, ésta puede ser de 2 tipos: no persistente o persistente. El HTTP/1.0 utiliza sólo conexiones no persistentes y el HTTP/1.1 puede utilizar además conexiones persistentes.

- En una conexión no persistente, para cada objeto Web transferido se establece una conexión TCP independiente. En una conexión persistente, durante la misma conexión (establecida por un par cliente/servidor) se pueden transmitir muchos objetos Web, lo que generalmente ahorra recursos en el servidor (memoria y CPU) y en la red (ancho de banda).

- Ambos tipos de conexiones pueden ser además secuenciales o paralelas (pipelining). Estas últimas permiten ahorrar tiempo si transmitimos varios objetos Web porque el cliente puede realizar una nueva petición antes de recibir la respuesta de una previa.

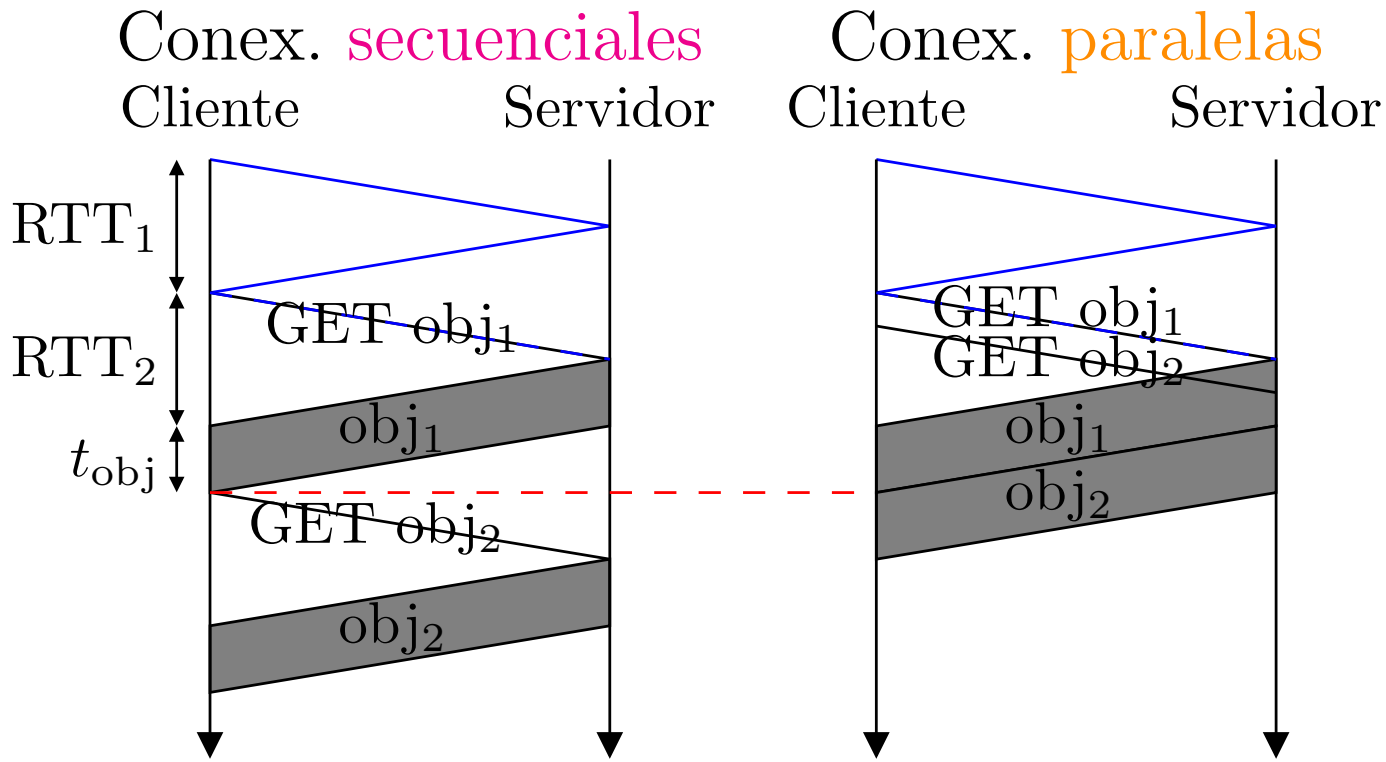
1.3.1. Conexiones no persistentes



RTT (Round-Trip Time) es el tiempo de ida y vuelta de un mensaje de longitud despreciable y t_{obj} es el tiempo de transmisión del objeto Web. En cada conexión TCP, el tiempo de establecimiento de conexión TCP necesita un RTT (RTT_1). El segundo RTT (RTT_2) se emplea en

solicitar el objeto.

1.3.2. Conexiones persistentes



El ahorro del establecimiento de una conexión TCP para cada conexión Web reduce el tiempo en el caso de las conexiones secuenciales. El tiempo de respuesta de las conexiones paralelas persistentes es

el mismo que el de las conexiones no persistentes, aunque la carga para el servidor suele ser menor.

1.4. Los mensajes HTTP

En una comunicación HTTP sólo existen dos tipos de mensajes, los de petición (request) y los de respuesta (reply).

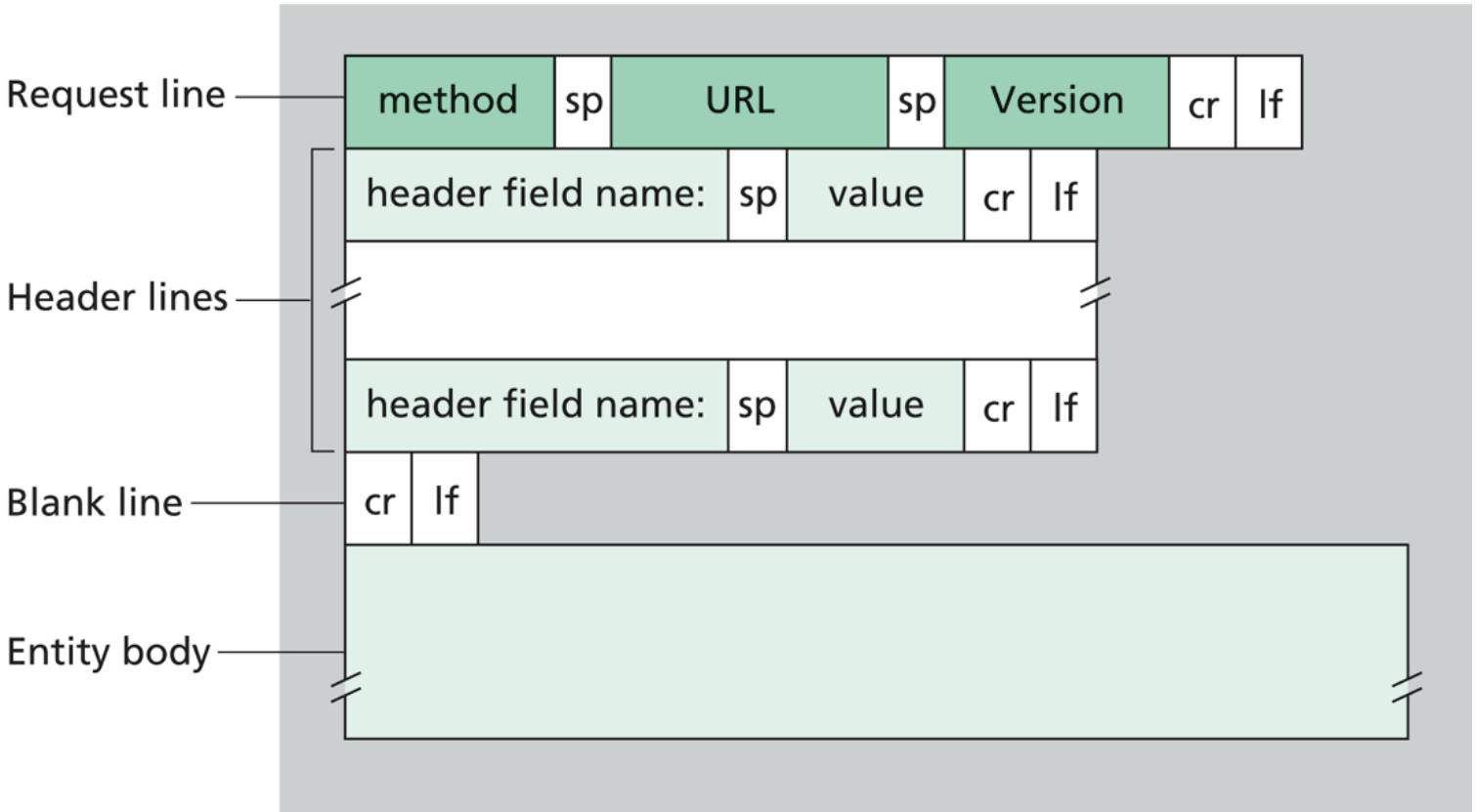


Figure 2.8 ♦ General format of a request message

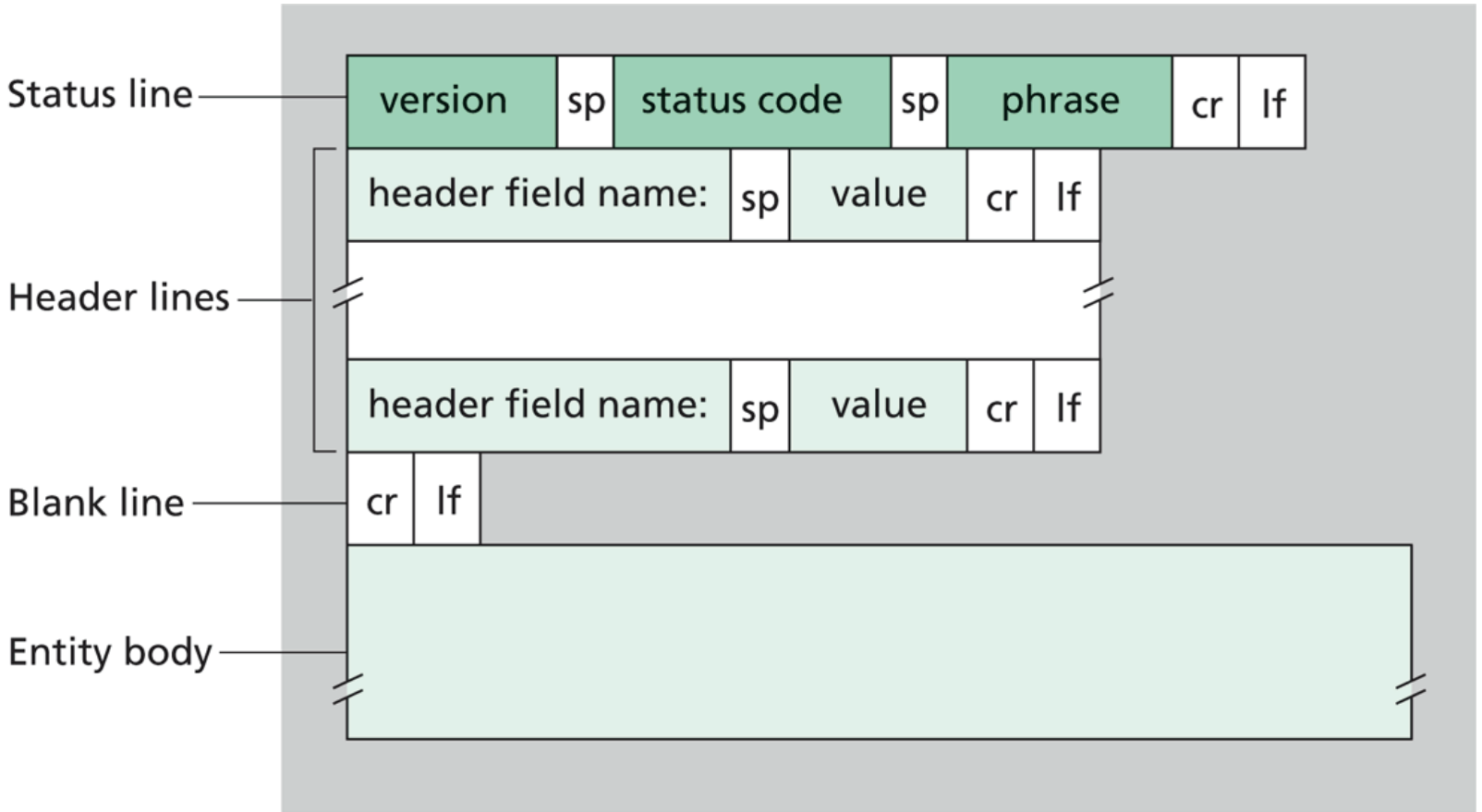


Figure 2.9 ♦ General format of a response message

1.4.1. Un mensaje de petición

Capturado usando `ethereal` conectándose a `www.google.es` mediante `mozilla`.

```
GET / HTTP/1.1
Host: www.google.es
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.2.1) G
Accept: text/xml,application/xml,application/xhtml+xml,text/htm
Accept-Language: en-us, en;q=0.50
Accept-Encoding: gzip, deflate, compress;q=0.9
Accept-Charset: ISO-8859-1, utf-8;q=0.66, *;q=0.66
Keep-Alive: 300
Connection: keep-alive
Cookie: PREF=ID=1e9556644260c793:LD=es:TM=1076751818:LM=107675

<Cuerpo de entidad>
```

Significado de algunas de las líneas:

- Los mensajes de petición están codificados en ASCII, comienzan siempre por la cadena `GET`, la cadena `POST` o la cadena `HEAD` y acaban siempre en una línea en blanco (retorno de carro y nueva línea).
- El número de líneas es variable, pero como mínimo siempre existe la primera (línea de petición) donde se indica el tipo de petición (`GET`), la páqina HTML reclamada (`/directorio/pagina.html`, aunque en el ejemplo se trata de la página `index.html` que es la página por defecto) y la versión del protocolo HTTP utilizado (`HTTP/1.1`).
- El resto de las líneas de cabecera.
  - La primera línea, que comienza en el ejemplo por `Host:` especifica el host en que reside el objeto (necesario para los proxies).
  - La línea `Connection:` indica el tipo de conexión (`keep-alive` significa conexión persistente y `close` conexión no persistente.

  - La línea `User-Agent:` indica el navegador Web utilizado. Esto es importante para el servidor porque dependiendo del navegador se pueden enviar páginas HTML diferentes (con idéntica URL).
  - La línea `Accept-Language:` indica que el usuario prefiere recibir una versión en inglés del objeto.

- `<Cuerpo de entidad>` es un campo de los mensajes de petición que está vacío cuando se utiliza el método `GET`, pero que sí se utiliza con el método `POST`. Este método se emplea, por ejemplo, para rellenar formularios (donde el cliente debe enviar información al servidor). Finalmente, el método `HEAD` es similar al método GET, pero el servidor en su respuesta no va a incluir el objeto pedido. Este método es normalmente utilizado por los desarrolladores de aplicaciones.

- En la versión HTTP/1.1, además de `GET`, `POST` y `HEAD`, existen otros dos métodos: `PUT` que permite a un usuario cargar un objeto en el servidor Web y `DELETE` que permite borrarlo.

1.4.2. Un mensaje de respuesta

```
HTTP/1.1 200 OK
Cache-control: private
Content-Type: text/html
Content-Encoding: gzip
Server: GWS/2.1
Content-length: 1484
Date: Sat, 14 Feb 2004 10:52:44 GMT

<Cuerpo de entidad>
```

Significado de algunas de las líneas:

- Los mensajes de respuesta siempre tienen 3 secciones: la línea inicial de estados, las lineas de cabecera y el cuerpo de la entidad.
- La línea inicial de estados tiene 3 campos: la versión del protocolo, el código de estado y el estado (estas dos cosas significan lo mismo). En el ejemplo, `200 OK` significa que el servidor ha encontrado el objeto y que lo ha servido (en el ejemplo no se muestra tal cual, se

trata de un objeto comprimido). En la siguiente tabla se presentan algunos de los códigos de control más comunes:

Código	Significado
`200 OK`	Petición exitosa
`301 Moved Permanently`	El objeto demandado ha sido movido a la URL especificada en `Location:`
`400 Bad Request`	Petición no entendida por el servidor
`404 Not Found`	Objeto no encontrado en el servidor
`505 HTTP Version Not Supported`	Obvio

- `Server:` indica el software del servidor Web.
- `Date:` indica la fecha y hora en la que se creó y envió la respuesta HTTP. Este campo es importante porque ayuda a los proxies a mantener sus cachés.
- `Content-length:` indica el tamaño en bytes del objeto enviado.

- `Content-Type:` indica que el objeto enviado en el cuerpo de la entidad es texto HTML.

1.5. Paso de parámetros en las URL's

- En la sección anterior hemos visto que uno de los métodos usados en los mensajes de petición (`POST`) era utilizado para enviar datos de tipo "formulario" al servidor. Esto también ser puede realizar mediante `GET`.
- Para ello, en la URL usada se especifican dichos datos mediante la forma:

  URL?primer parámetro&segundo parámetro&...

  Por ejemplo:

  ```
  http://www.google.es/language_tools?hl=es
  ```

1.6. Identificación de usuarios

- El HTTP no tiene estado lo que significa que un servidor no guarda información acerca de los datos enviados a un determinado cliente. Esto simplifica el diseño de los servidores.
- A menudo, el servidor necesita identificar a los usuarios, bien porque el acceso al servidor esté restringido, o porque quisiera servir cierto contenido en función de la identidad del usuario.
- El HTTP proporciona dos mecanismos:
  1. La autorización "login/password".
  2. Las cookies (galletas).

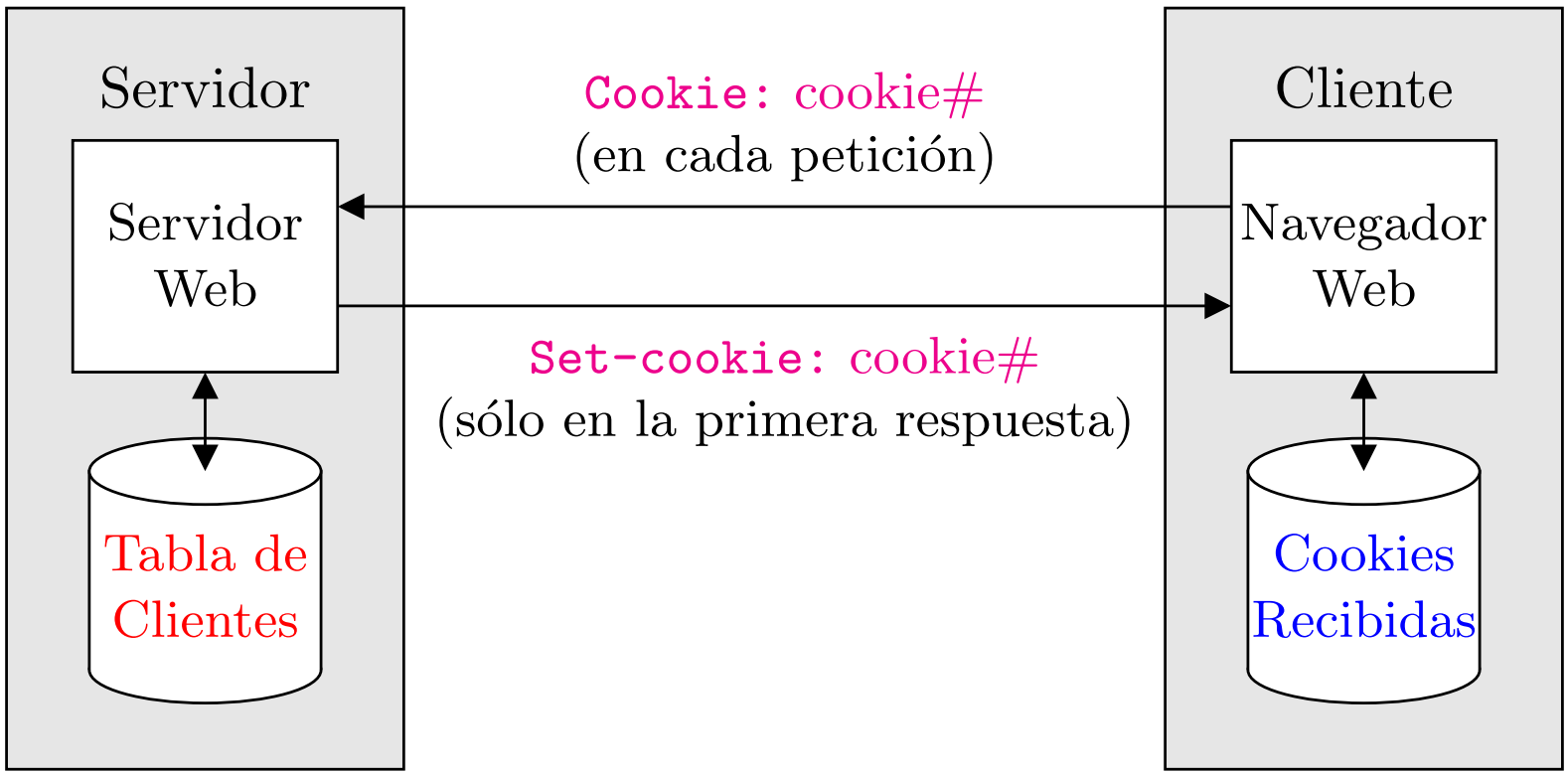
1.6.1. Autorización "login/password"

- El usuario se identifica mediante un nombre de usuario (login) y una palabra clave (password). Pasos:

  1. El servidor avisa al navegador que debe identificarse mediante una línea inicial de estado `HTTP/1.1 401 AuthorizationRequired` en su mensaje HTTP de respuesta.
  2. El navegador solicita al usuario el login/password e incluye esta información en una línea `Authorization:` del próximo mensaje de petición. Dicha línea se incluye en cualquier nueva petición de cualquier nuevo objeto al servidor.

1.6.2. Cookies

- Las cookies sirven para que los navegadores identifiquen a los usuarios que anteriormente se han conectado.



- En la tabla de clientes existen entradas de la forma (cookie#,usuario), que son creadas la primera vez que el usuario se conecta.

- En el fichero de cookies recibidas el navegador almacena una entrada (servidor Web, cookie#) cada vez que se conecta (por primera vez) a un servidor que utiliza cookies.

1.7. El GET condicional

- Los navegadores Web poseen una caché donde almacenan los objetos más recientes.
- Cuando un navegador va a reclamar un objeto, primero mira si está en su caché. Si está, entonces su petición es condicional. Si no está, su petición no es condicional.
- Cuando se realiza una petición condicional, el servidor Web envía una nueva versión sólo si la copia local es obsoleta.

1.7.1. GET (normal)

Petición

```
GET /fruit/kiwi.gif HTTP/1.0
User-agent: Mozilla/4.0
```

Respuesta

```
HTTP/1.0 200 OK
Date: Web, 12 Aug 1998 15:39:29
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 09:23:24
Content-Type: image/gif

(cuerpo de entidad)
```

1.7.2. GET condicional

En la caché del cliente existe el objeto reclamado.

Petición

```
GET /fruit/kiwi.gif HTTP/1.0
User-agent: Mozilla/4.0
If-modified-since: Mon, 22 Jun 1998 09:23:24
```

Respuesta

Suponiendo que el objeto no ha sido modificado desde `22 Jun 1998 09:23:24`.

```
HTTP/1.0 304 Not Modified
Date: Web, 19 Aug 1998 15:39:29
Server: Apache/1.3.0 (Unix)
```

1.8. Las cachés Web (proxies Web)

Funcionamiento

- La Web es un entramado de servidores de contenidos y de clientes (navegadores Web y cualquier otra aplicación distribuida que utilice el HTTP para comunicarse) (véase http://www.rediris.es/si/cache).
- Para minimizar los tiempos de respuesta y el tráfico en la red, la Web utiliza un conjunto de servidores especiales (llamdos proxies Web[2]) que funcionan como una caché, almacenando todo lo que sirven a los clientes para su posterior reenvío según una determinada política (por ejemplo, almacenando siempre los objetos más frecuentes).
- De esta forma, cuando un cliente (configurado para utilizar un

[2]Existen otros tipos de proxies que no son Web y por lo tanto, cuando exista confusión debería especificarse. En este documento esto no es así por lo que prescindiremos de especificar el tipo de proxy.

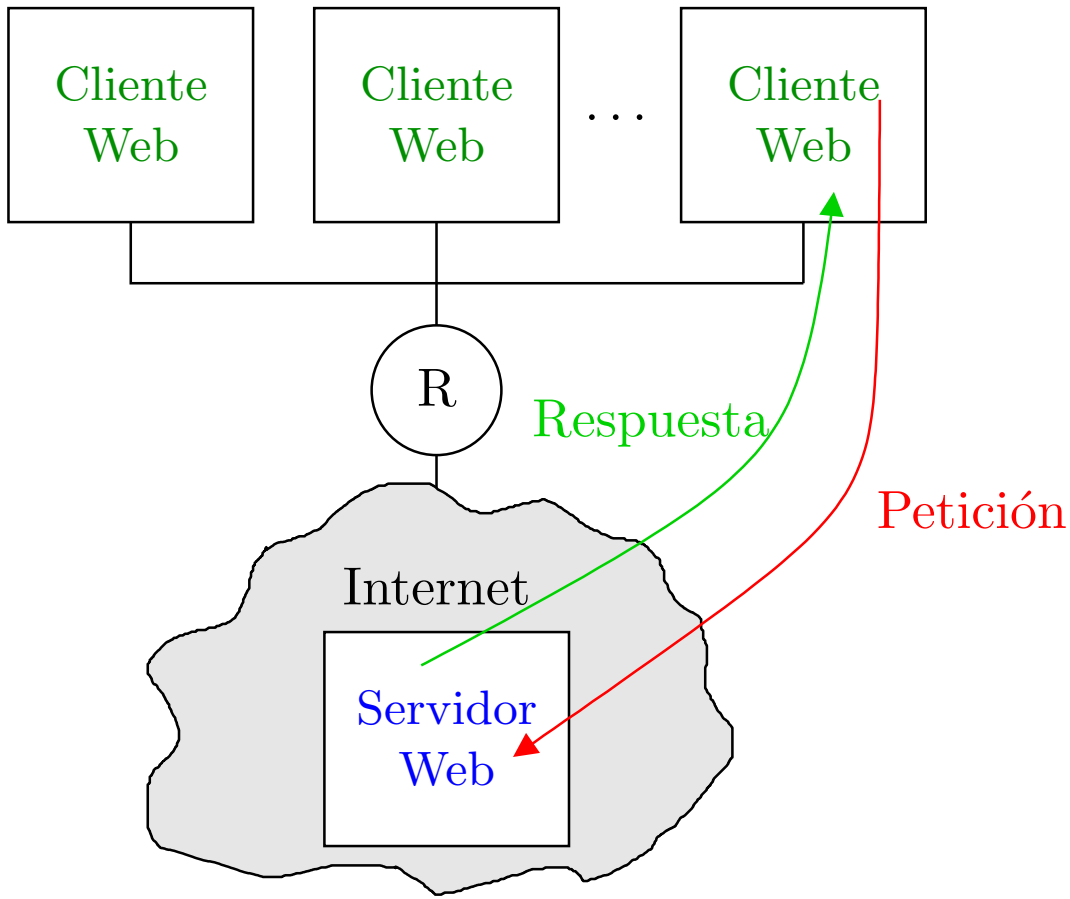
proxy) solicita un objeto a un servidor Web, en lugar de hacerlo directamente al servidor lo realiza al proxy. El proxy entonces mira si posee el objeto y si es así, lo sirve. En caso contrario lo solicita al servidor, lo almacena y lo sirve.

- Para mantener actualizada la caché de un proxy, este utiliza el GET condicional con el servidor (o el proxy de nivel superior).

1.9. Arquitecturas Web

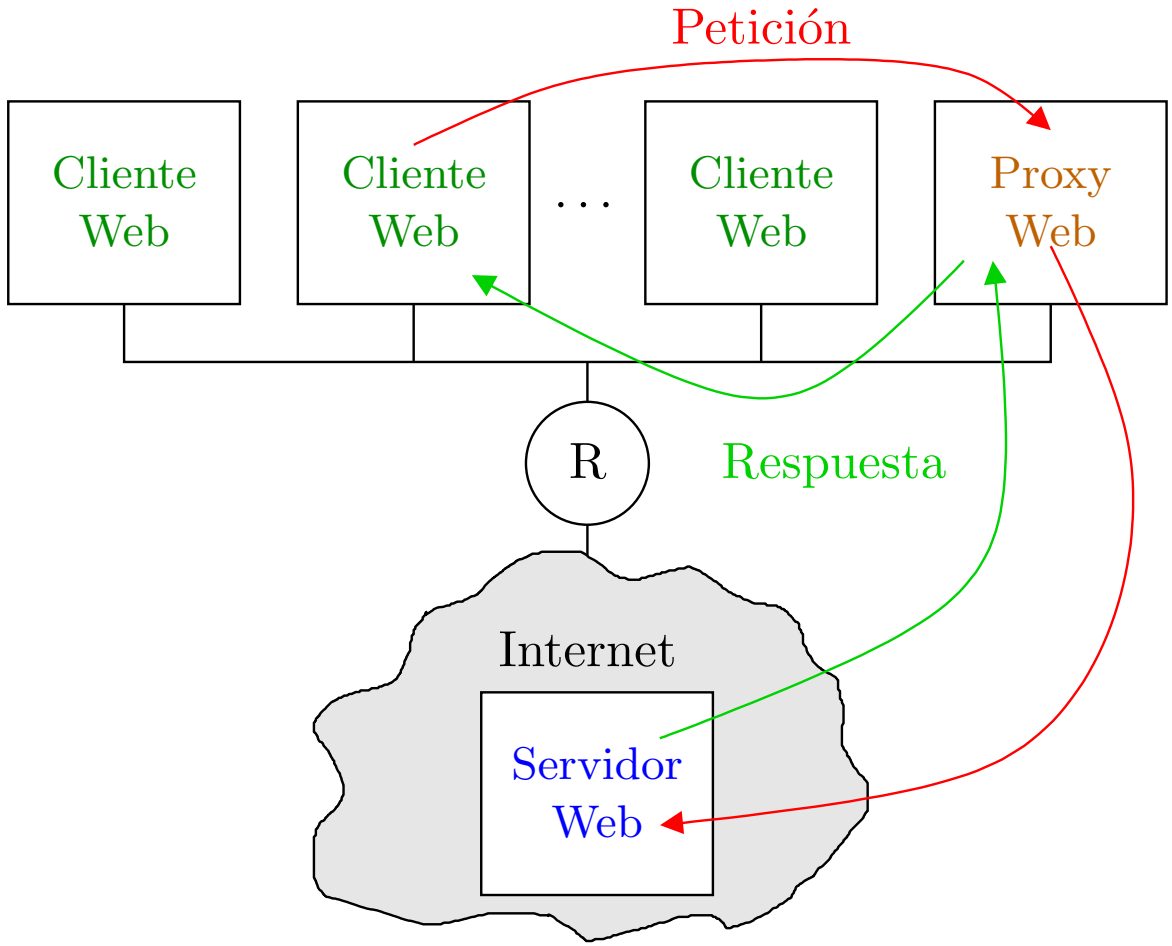
1.9.1. La configuración más sencilla

- En su configuración más simple, los clientes "atacan" directamente a/los servidores Web.



1.9.2. Sistemas proxy de 1 nivel

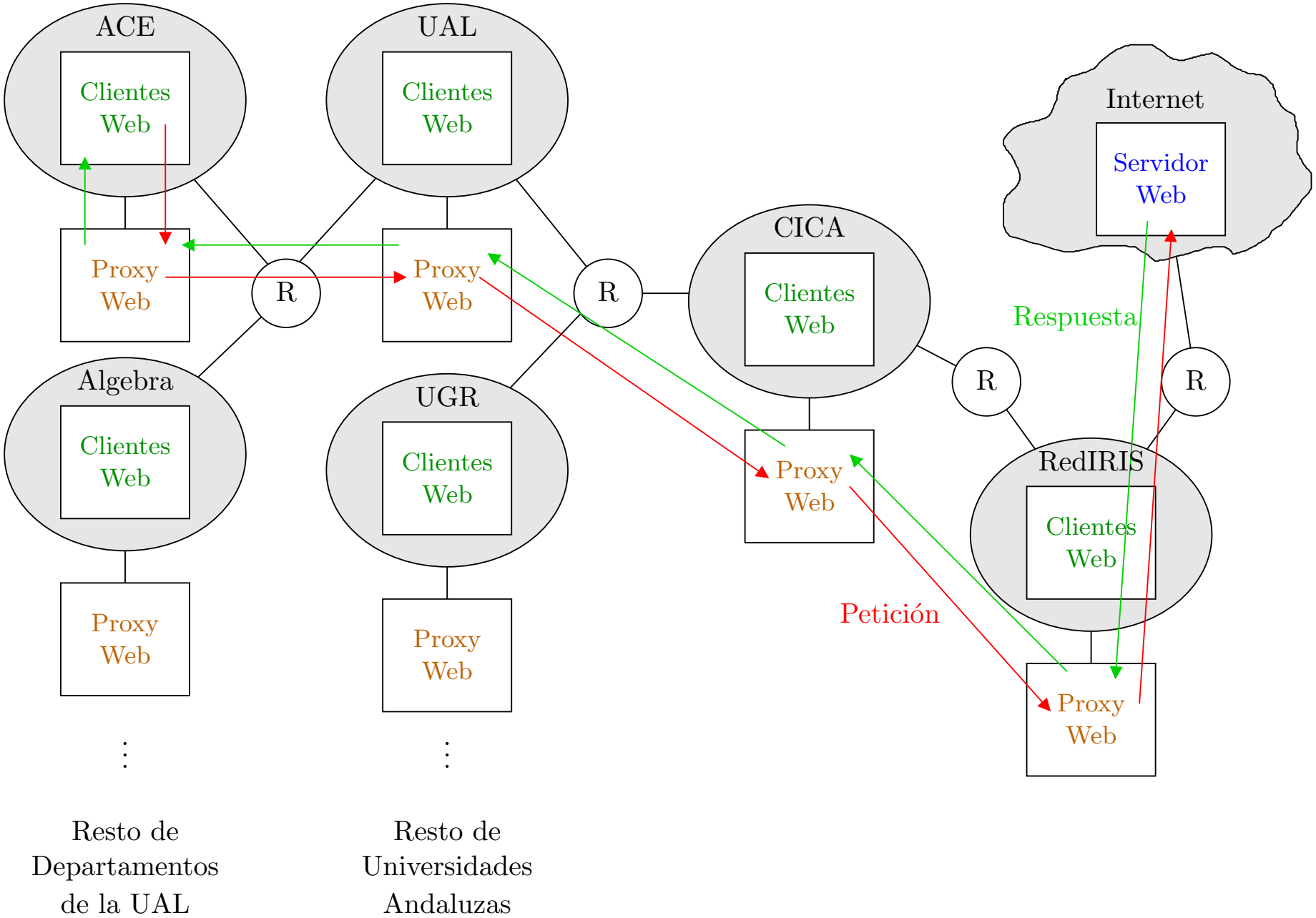
- El cliente solicita un objeto a su proxy local y si este no lo encuentra, lo solicita al servidor Web pertinente.



1.9.3. Sistemas proxy multinivel

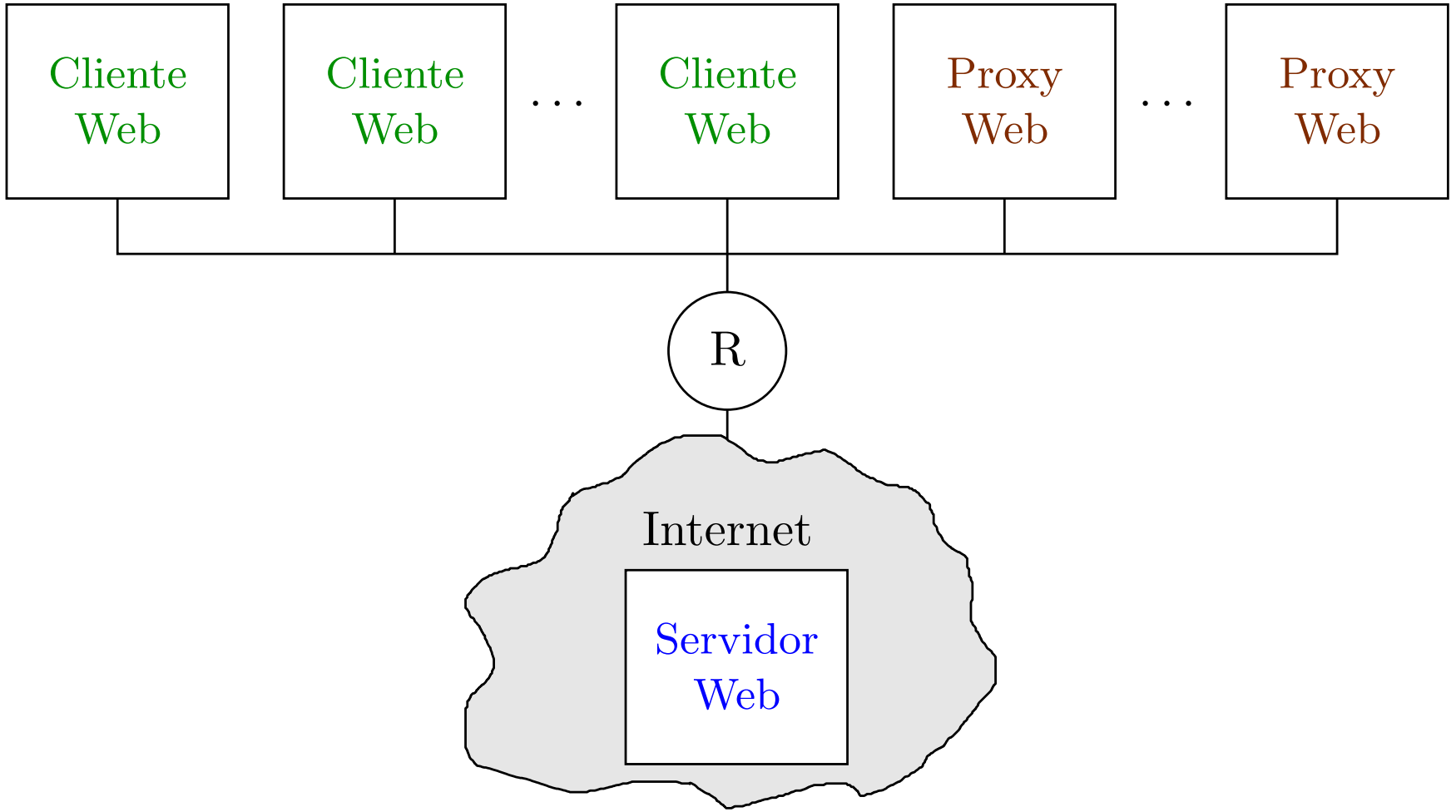
- Los servidores proxy pueden configurarse para utilizar de forma recursiva otros servidores proxy de mayor ambito. A esto se le llama caché Web cooperativa.

- Los proxies de una jerarquía pueden utilizar el Internet Caching Protocol (ICP) para hablar todos con todos a la hora de localizar un objeto dentro de la jerarquía. De esta manera, si el objeto se localiza en un proxy cercano el proceso de descarga del objeto (que finalmente se realiza mediante HTTP) se aceleraría.


ACE
Clientes Web
Proxy Web
Algebra
Clientes Web
Proxy Web
UAL
Clientes Web
Proxy Web
UGR
Clientes Web
Proxy Web
R
R
CICA
Clientes Web
Proxy Web
R
R
Internet
Servidor Web
Respuesta
RedIRIS
Clientes Web
Petición
Proxy Web
Resto de Departamentos de la UAL
Resto de Universidades Andaluzas

1.9.4. Sistemas proxy distribuidos

- Los proxies pueden soportar cargas muy altas cuando "representan" a una gran cantidad de servidores Web.
- Cuando un proxy soporta demasiada carga se puede proceder a la distribución del contenido de su caché sobre un conjuto de proxies.
- El contenido de cada proxy se controla mediante el Cache Array Routing Protocol (CARP) que utiliza rutado por dispersión (hashing). Esta forma de rutado consiste en usar un proxy distinto en función de la URL del objeto Web solicitado.
- Cuando todos los clientes utilizan la misma función de dispersión, un objeto sólo reside en uno de los proxies (en ese nivel).

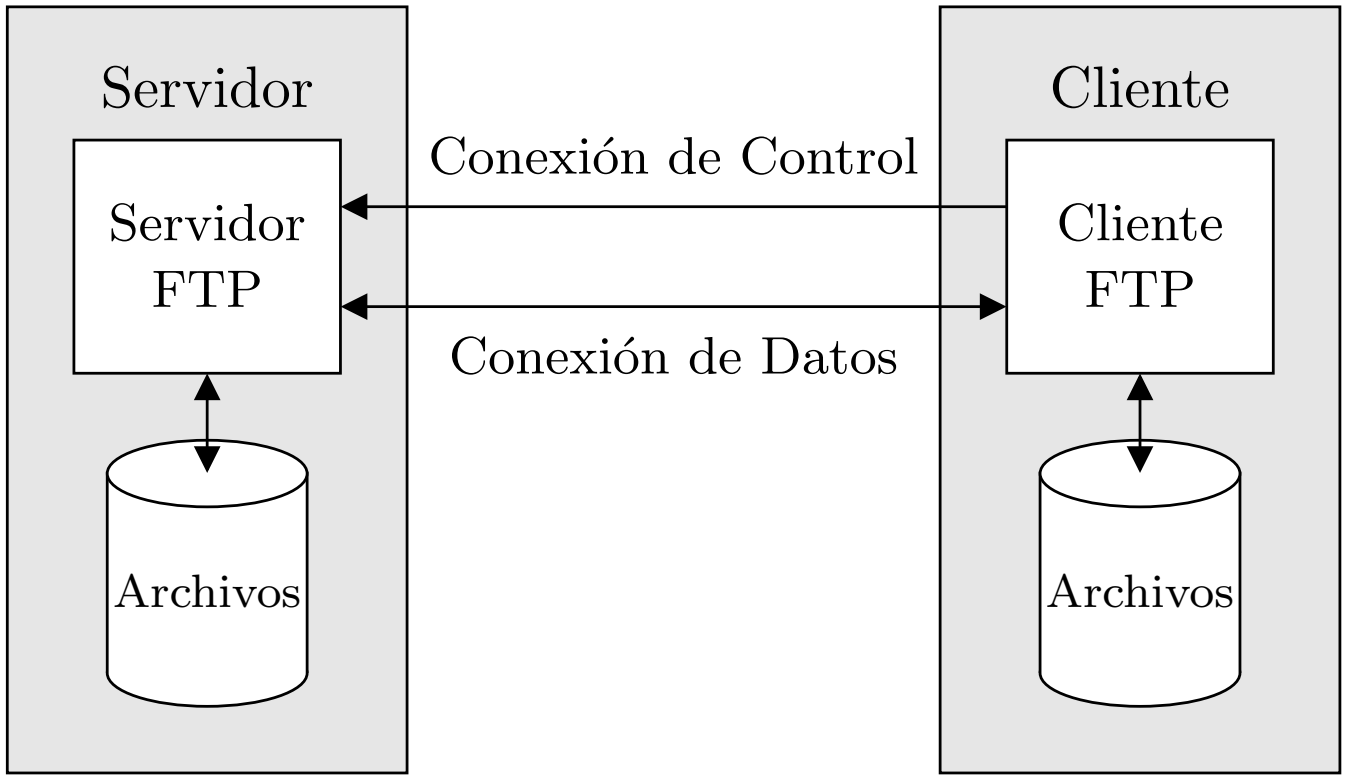

Cliente Web
Cliente Web
...
Cliente Web
Proxy Web
...
Proxy Web
R
Internet
Servidor Web

Capítulo 2

Compartición de Ficheros (File Sharing)

2.1. El FTP (File Transfer Program)

- La aplicación FTP permite la transmisión de archivos entre hosts remotos.



- Utiliza TCP en ambas conexiones.

- La conexión de control transporta la información (en formato ASCII) necesaria para controlar la transmisión de archivos (login/password, comandos para movernos por el sistemas de ficheros remoto y mover archivos, listar directorios, ...). El servidor recibe la información de control a través del puerto 21 y utiliza el File Transfer Protocol (FTP, descrito en el RFC 959).
- La conexión de datos se utiliza para transferir 1 archivo (se establece una conexión TCP para cada archivo). El servidor utiliza para esto el puerto 20.

2.2. Aplicaciones P2P (Peer-to-peer)

- Muy alta escalabilidad (los clientes son también servidores).
- Se utilizan para compartir archivos que residen en los hosts de los usuarios. Por esto a las aplicaciones P2P también se les llama aplicaciones de compartición de archivos entre iguales. Ejemplos: Napster, Gnutella, KaZaA, eMule y BitTorrent.
- Contexto complejo: típicamente los hosts no permanecen todo el tiempo encendidos (ni conectados) y rara vez disponen de una dirección IP fija.
- Cuando un usuario busca un archivo obtiene una lista de hosts que en este momento están conectados y almacenan (al menos parcialmente) dicho archivo. Con esto se calcula un índice de accesibilidad al fichero.

- La transmisión (generalmente mediante TCP) de un archivo se produce directamente (sin servidores intermedios) entre uno o varios de esos hosts y el del usuario (que ha realizado la búsqueda). Esto permite transmitir simultáneamente distintas parte del mismo archivo mediante múltiples conexiones.

- En cuanto el usuario dispone de una sección del archivo, automáticamente entra a formar parte de la lista de iguales que disponen de él. Así, mientras lo descarga, otros usuarios pueden comenzar a descargarlo de él.

- Por tanto, las aplicaciones P2P de compartición de archivos son altamente escalables en lo que se refiere a la cantidad de datos que pueden gestionar, porque aunque un nuevo usuario consume ancho de banda de salida de los iguales que contienen los archivos que se está descargando, también proporciona el suyo para que otros hagan lo mismo.

- Sin embargo, en general los sistemas P2P presentan un problema de cuello de botella en los buscadores de contenidos. A continuación discutiremos las 3 estrategias existentes.

2.2.1. Búsqueda usando un directorio centralizado

- Ejemplos: Napster, eMule y BitTorrent.
- Existe un gran servidor (Napster) o un conjunto de servidores (parcialmente) replicados (eMule) para que proporcionan servicio de búsqueda. En el caso de BitTorrent, cada fichero `.torrent` especifica un servidor que no es de búsqueda de contenidos, sino de determinación de peers para ese contenido.
- Cada igual (cuando se conecta) informa al servidor de su dirección IP y de los contenidos que desea compartir. Así, el servidor crea y mantiene una base de datos dinámica que relaciona cada nombre de fichero con el conjunto de IP's de los hosts que lo contienen.

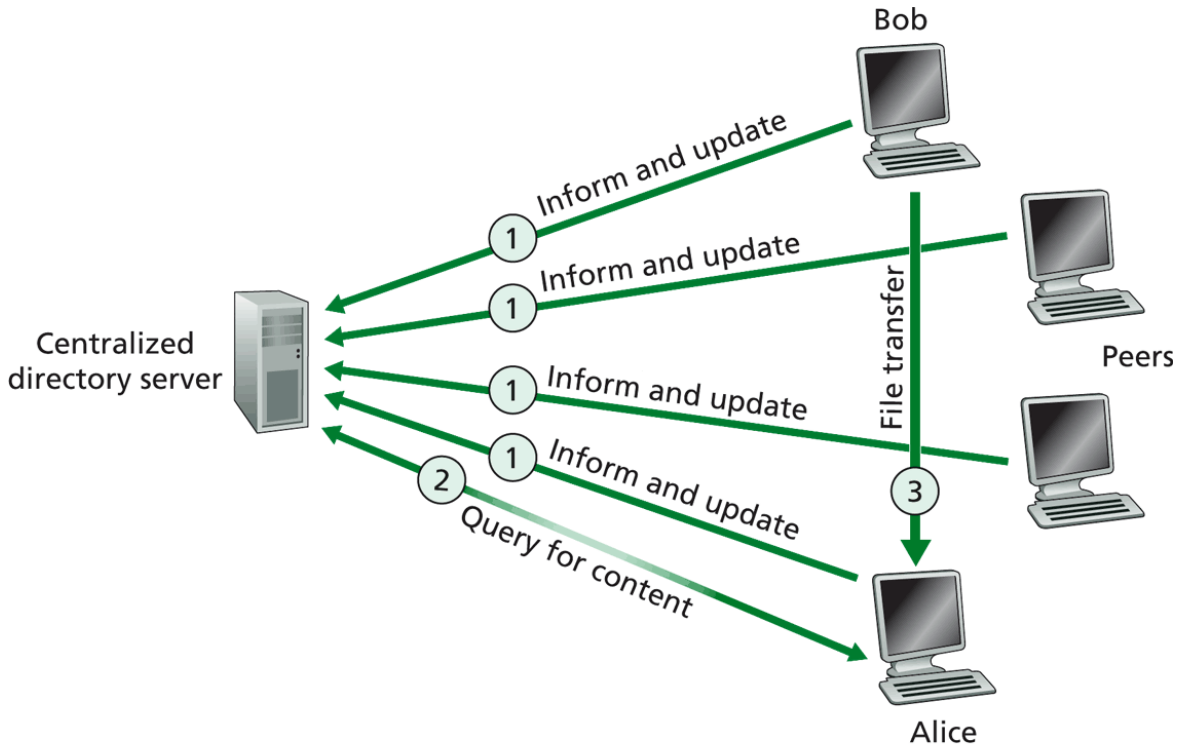


Figure 2.23 ♦ The P2P paradigm with a centralized directory

- Ventajas: la búsqueda de contenidos (por parte de los iguales) es sencilla porque la base de datos está centralizada.
- Desventajas: el servidor central se convierte en el cuello de botella del sistema y si no existe ninguno funcionando, ningún igual puede buscar.

2.2.2. Búsqueda usando un directorio descentralizado

- Ejemplos: KaZaA, Overnet, eDonkey y Kademlia (eMule).
- Un subgrupo de iguales son designados como líderes de grupo (supernodos) y cuando un nuevo igual se conecta, se le asigna uno de ellos.

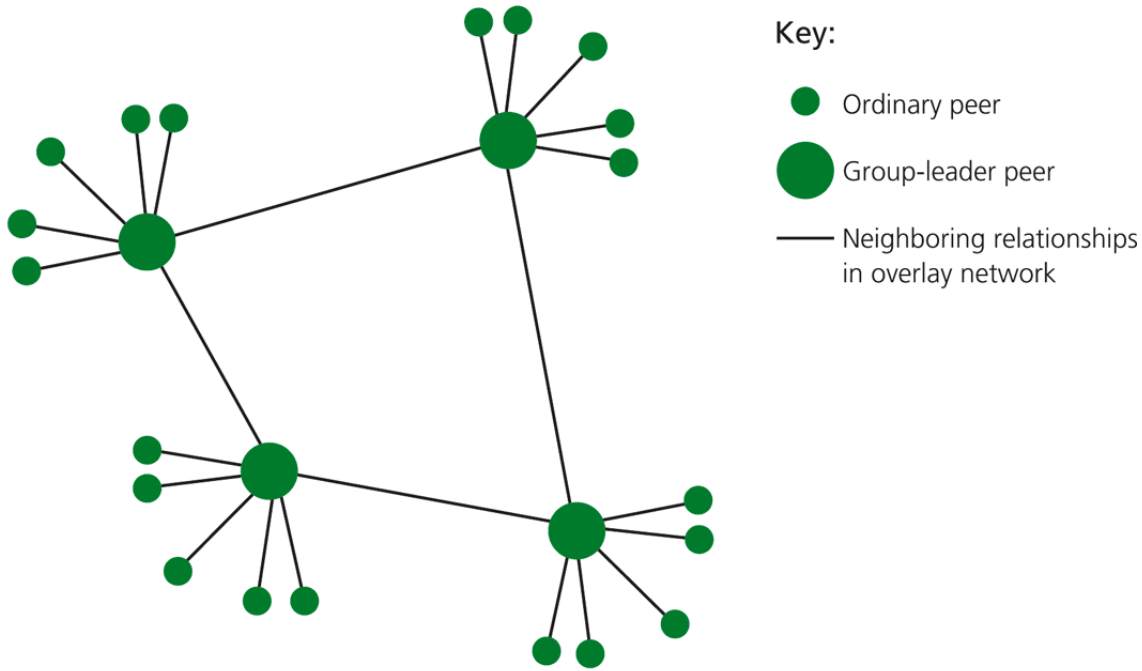


Figure 2.25 ♦ Hierarchical overlay network for P2P file sharing

- Cada líder de grupo gestiona una base de datos con la información de los iguales de su grupo.
- Cuando un usuario busca, lo hace primero dentro de su grupo y obtiene una contestación rápida. Si el usuario lo solicita, puede ampliar la búsqueda al resto de grupos ya que el líder de su grupo conoce a los demás líderes.
- Cuando un usuario se conecta lo hace a partir de un nodo de arranque que conoce al menos uno de los líderes de grupo. No es necesario conocer, de entrada, a un supernodo.
- Un nodo (un igual) se convierte en un líder normalmente porque el usuario lo solicita[1].

[1] Probablemente a cambio de gastar ancho de banda extra en ser servidor de consultas, obtenga mayores prioridades a la hora de acceder a los ficheros compartidos.

- Ventajas de la búsqueda descentralizada:
  - Las bases de datos son localmente más pequeñas (un líder sólo se encarga de un subconjunto de los iguales, generalmente los más próximos (menos hops)).
  - El sistema es más escalable y resistente a fallos porque ahora la base de datos global está distribuida.
- Desventajas:
  - Cuando un líder se apaga, los iguales tienen que buscar otro líder vecino y regenerar la base de datos local.
  - Los nodos no son todos iguales (los líderes, que también funcionan como igual, consumen más recursos que los iguales).
  - Sigue existiendo la necesidad de un nodo inicial de arranque que conozca al menos un líder y que no puede apagarse.

2.2.3. Búsqueda mediante inundación

- Ejemplo: Gnutella.
- No existe una jerarquía de servidores (supernodos) e iguales. Todos los participantes son iguales.
- Para que un nodo se conecte debe conocer, al menos, la dirección IP de un nodo que ya forme parte de la red de compartición.
- Las búsquedas se realizan mediante *inundación de consultas*:

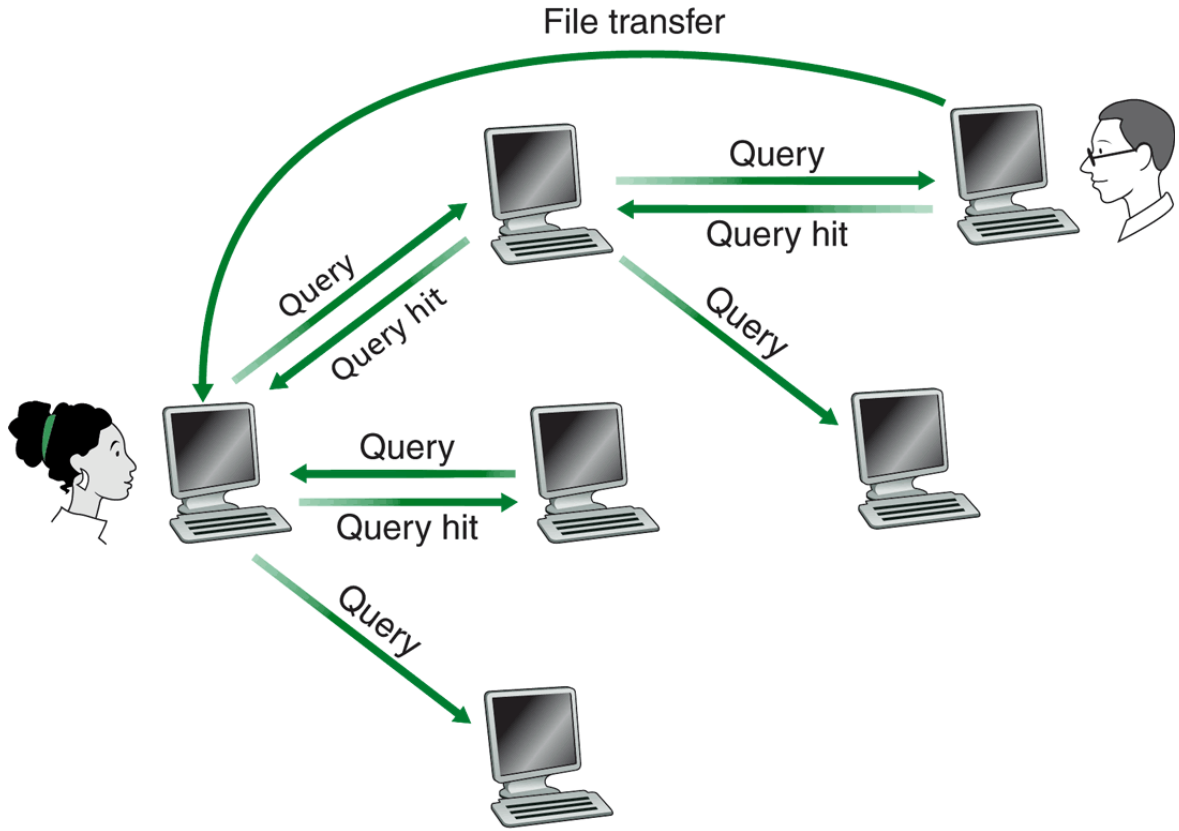


Figure 2.24 ♦ Search and file transfer in Gnutella

1. El nodo que inicia la búsqueda pregunta primero a sus vecinos (inmediatos).
2. Los vecinos repiten el proceso con sus vecinos inmediatos (excepto con los que le han preguntado ya por lo mismo).

3. Los nodos que contienen el archivo buscado se lo comunican al nodo vecino que le ha preguntado la búsqueda. Este proceso acaba cuando las contestaciones llegan hasta el nodo que inició la búsqueda.

- Ventajas:
  - Todos los nodos son iguales.
  - No existen bases de datos con información de directorio.
- Desventajas:
  - El volumen de tráfico generado por las consultas suele ser muy alto. Por este motivo el área de búsqueda (número de hops) suele estar limitado. Lo malo de esto es que puede ocurrir que sólo se obtengan búsquedas parciales.
  - Es necesario conocer un nodo de la red de compartición para poder formar parte de ella (`http://www.gnutella.com`).

2.3. Acerca de la tasa de descarga

- La tasa de descarga de las aplicaciones P2P depende normalmente de la tasa de subida a la red. Si ésta es alta, la de descarga también.
- Existen dos formas distintas de premiar al que más envía:
  1. Incrementando la prioridad del peer en las colas asociadas a los contenidos: "Si subes más, tardas menos tiempo de obtener conexiones". Esto es lo que ocurre en eMule, por ejemplo.
  2. Conectándote a los mejores peers. Hay protocolos, como BitTorrent, que el número de peers a los que te conectas está limitado (5 generalmente). Si los peers tratan de enviar datos a aquellos otros peers que mejor le sirven, aquellos peers que más ancho de banda dedican se comunicarán entre sí. De esta manera, "Si subes más, te conectarás a los mejores peers y por tanto, obtendrás más datos de la red". En BitTorrent cada 30 segundos la peor de las 5 conexiones se cierra y se establece una nueva (tras consultar al tracker) con la idea de mejorar algunas de las otras 4 conexiones existentes.

- En la práctica no debe dedicarse el 100 % del ancho de banda de subida porque ahogaríamos las conexiones TCP de descarga. Técnicamente lo que ocurre es que los segmentos de control de flujo que se envían a los peers se retrasan, provocándose un sobre-control del flujo.

2.4. Network File System

2.4.1. Características

- Desarrollado en 1984 por Sun Microsystems para SunOS. Más tarde donado al *public domain* [**?**].
- Permite montar un sistema de ficheros remoto dentro del sistema de ficheros local.[2] Las aplicaciones no ven la diferencia.
- El sistema de ficheros es "exportado" por el host remoto.
- El sistema de ficheros es "montado" por el host local.
- Muchos hosts remotos pueden montar el mismo sistema de ficheros y compartir así datos de forma sencilla.

[2] Un sistema de ficheros es cualquier directorio que cuelgue del directorio raíz en el host remoto (un directorio, una partición de un disco duro, un CD-ROM, un disco RAM, etc.).

- Las aplicaciones locales acceden al sistema de ficheros remoto como si fuera local. En otras palabras, el NFS es completamente transparente al usuario.
- Arquitectura cliente/servidor. El servidor se monta en el host que exporta el sistema de ficheros y el cliente, en los hosts que lo montan.
- El servidor asegura la integridad de los datos en el sistema de ficheros exportado.
- El NFS utiliza un protocolo a nivel de la capa de aplicación y se monta encima del sistema RPC, otro paquete de la capa de aplicación [**?**]. Dicho protocolo es el NFSP (NFS Protocol). Actualmente cohexisten tres versiones: la 2 [**?**], la 3 [**?**] y la 4 [**?**].
- Los datos son transmitidos a través de la red usando el sistema de representación XDR (eXternal Data Representation) [**?**]. Esto permite que que host de diferente clase (con distintos SO's, endian's y/o longitudes de palabra) puedan intercambiar datos.

-

2.4.2. El NFSP

- Es un protocolo sin estado lo que significa que ante un fallo del sistema (por ejemplo, que el servidor NFS se cuelgue), es el cliente el que sabe cómo recuperarse de los erroes (por ejemplo, re-ejecutando la orden de escribir un fichero que no pudo ser escrito por el cuelgue del servidor).
- El protocolo se describe mediante un conjunto de procedimientos que se ejecutan sobre el sistema RPC. A continuación se muestran algunos de los más usados:
  1. `null() returns()`: Hace un ping al server. Sirve para medir latencias en la red.
  2. `lookup(dir_fh, name) return (fh, attr)`: Busca el fichero `name` en el directorio `dir_fh` y si lo encuentra, devuelve el descriptor de ese fichero `fh` más la información `attr` sobre los atributos del fichero.

3. `create(dir_fh, name, attr) return(new_fh, new_attr)`:] Crea en e directorio `dir_fh` el fichero `name` con atributos `attr`. Si la creación tiene éxito, devuelve un nuevo descriptor de fichero `new_fh` que tiene los atributos `new_attr`.
4. `remove(dir_fh, name) returns(status)`: Borra el fichero `name` en el directorio `dir_fh` y devuelve verdadero si la operación ha tenido éxito.
5. `getattr(fh) returns(attr)`: Devuelve los atributos del fichero.
6. `setattr(fh, attr) returns(new_attr)`: Establece los atributos del fichero.
7. `read(fh, offset, count) returns(attr, data)`: Escribe en el puntero `data` los `count` bytes a partir del byte de índice `offset` del fichero `fh`. Se devuelve además los atributos del fichero.
8. `write(fh, offset, count, data) returns(attr)`: Semejante al anterior método, excepto que escribimos en el fichero

remoto.

9. `rename(dir_fh, name, to_fh, to_name) returns(status)`: Renombra el fichero `name` en el directorio `dir_fh` al fichero `to_name` en el directorio `to_fh`. Se devuelve información sobre el éxito de la operación.
10. `link(dir_fh, name, to_fh, to_name) returns(status)`: Crea el fichero `to_name` en el directorio `to_fh`, que es un enlace al fichero `name` en el directorio `dir_fh`.[3]
11. `symlink(dir_fh, name, string) returns(status)`: Crea un enlace simbólico llamado `name` en el directorio `dir_fh` con el valor `string`.[4]
12. `readlink(fh) return(string)`: Devuelve el nombre del enlace simbólico.

[3] Un enlace a un fichero es una nueva entrada en el sistema de ficheros que apunta a un fichero ya existente. Es como un seudónimo.

[4] La única diferencia entre un enlace (a secas, también llamado enlace duro) y un enlace simbólico es que éste último puede hacerse entre sistema de ficheros diferentes.

13. `mkdir(dir_fh, name, attr) returns(fh, new_attr)`: Crea un nuevo directorio llamado `name` dentro del directorio `dir_fh`, retorna el descriptor de ese directorio `fh` y los atributos `new_attr` con los que finalmente ha sido creado.
14. `mkdir(dir_fh, name) returns(status)`: Elimina un directorio vacío llamado `name` que está dentro del directorio `dir_fh` y devuelve el resultado de dicha operación.
15. `readdir(dir_fh, cookie, count) return(entries)`: En `entries` retorna hasta `count` bytes de las entradas en el directorio `dir_fh`. Cada entrada contiene un nombre de fichero, un identitificador y un puntero `cookie` a la siguiente entrada dentro del directorio. Este fichero permite en sucesivas llamadas ir recorriendo todas las entradas del directorio. Si `cookie == NULL` se devuelve la primera entrada.
16. `statfs(fh) return(fs_stats)`: Devuelve información sobre el sistema de ficheros remoto como el tamaño de bloque, el número de bloques libres, etc.

Capítulo 3

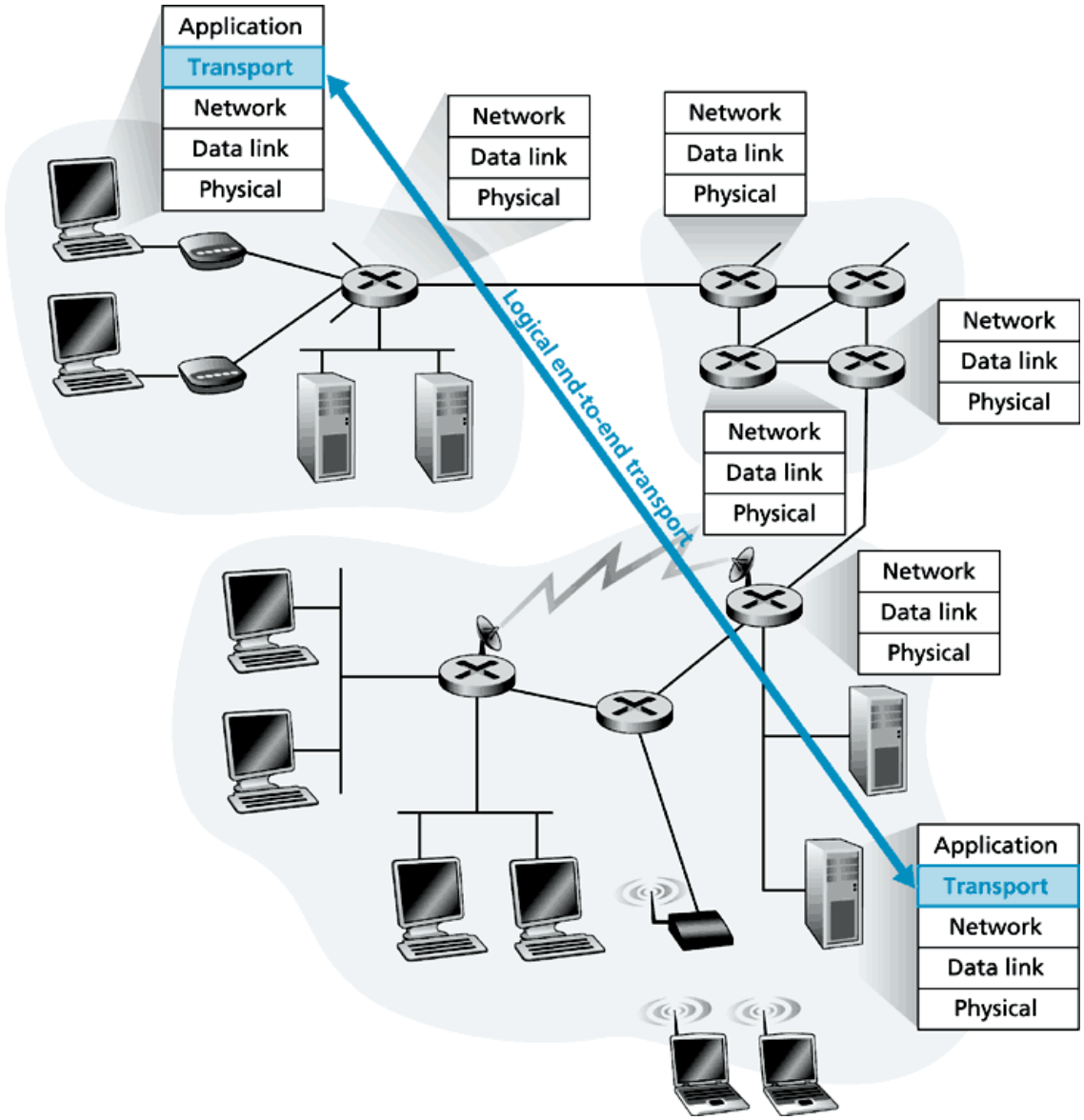
Servicios de la capa de transporte de datos

3.1. ¿Dónde corre la capa de transporte?

- La capa de transporte se ejecuta sólo en los sistemas finales (hosts). ¿Por qué?

  Actualmente la tasa de errores de las tecnologías de transmisión de datos son muy bajas. Esto posibilita que el control de flujo y de errores sea posible de extremo a extremo.

- Se sitúa a nivel 4, entre la capa de aplicación (nivel 5) y la capa de red (nivel 3).



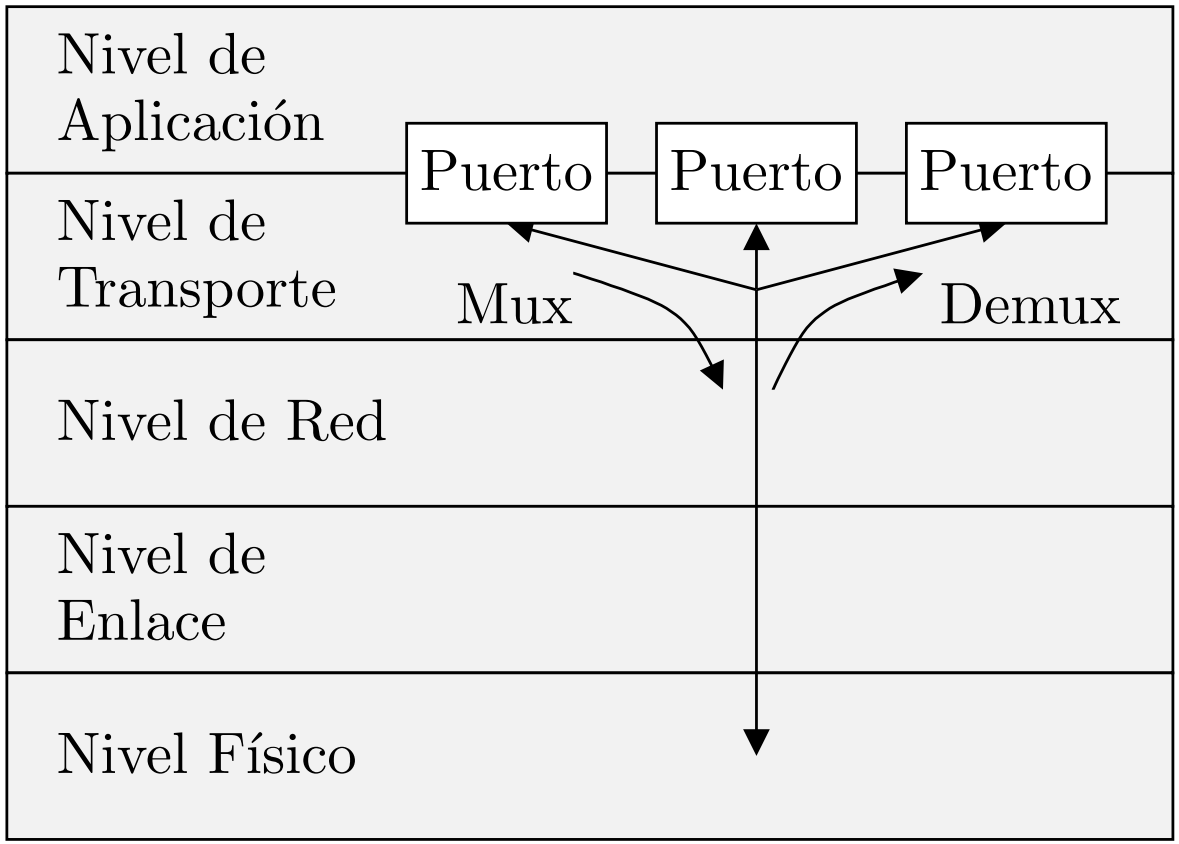
3.2. Servicios proporcionados por la capa de transporte

- Permite la **comunicación a nivel de procesos**[1] [?]. A este servicio también se le llama multiplexación. Proporcionado por el TCP y el UDP.
- Opcionalmente, corrige los errores de transmisión (modificación, pérdida, duplicación y reordenación). Sólo TCP.
- Opcionalmente, controla el flujo y la congestión. Sólo TCP.
- Opcionalmente, proporciona una conexión del tipo "byte-stream". Sólo TCP.
- No garantiza la transmisión de datos entre procesos en un tiempo limitado (principalmente porque la capa de red no garantiza la comunicación entre hosts en un tiempo limitado). TCP y UDP.

[1] Que pueden ejecutarse en una misma máquina o sobre máquinas diferentes.

3.3. El servicio de multiplexación

- En el host emisor se realiza una multiplexación (muchos procesos, una única capa de transporte/host) y en el proceso receptor una desmultiplexación (una única capa de transporte/host, muchos procesos).



- Permite aprovechar el servicio de entrega de host-a-host proporcionado por la capa de red para ofrecer un servicio de entrega de proceso-a-proceso.
- Los procesos que se ejecutan dentro de un host son diferenciados por la capa de transporte a partir del puerto en el que están escuchando.
- En los paquetes de datos figurará el puerto destino.

3.4. Sobre los puertos

- Permiten que más de una aplicación utilice la red en un determinado intervalo de tiempo.
- Cada puerto tiene asignado un número entre 0 y 65.535.
- Los puertos que comprenden desde el 0 al 1.023 están reservados para aplicaciones estándares (como la Web que utiliza el puerto 80) [**?**]. Dicha lista es mantenida por el IANA (Internet Assigned Numbers Authority, `http://www.iana.org`) y se puede consultar en el RFC 1700.[2]

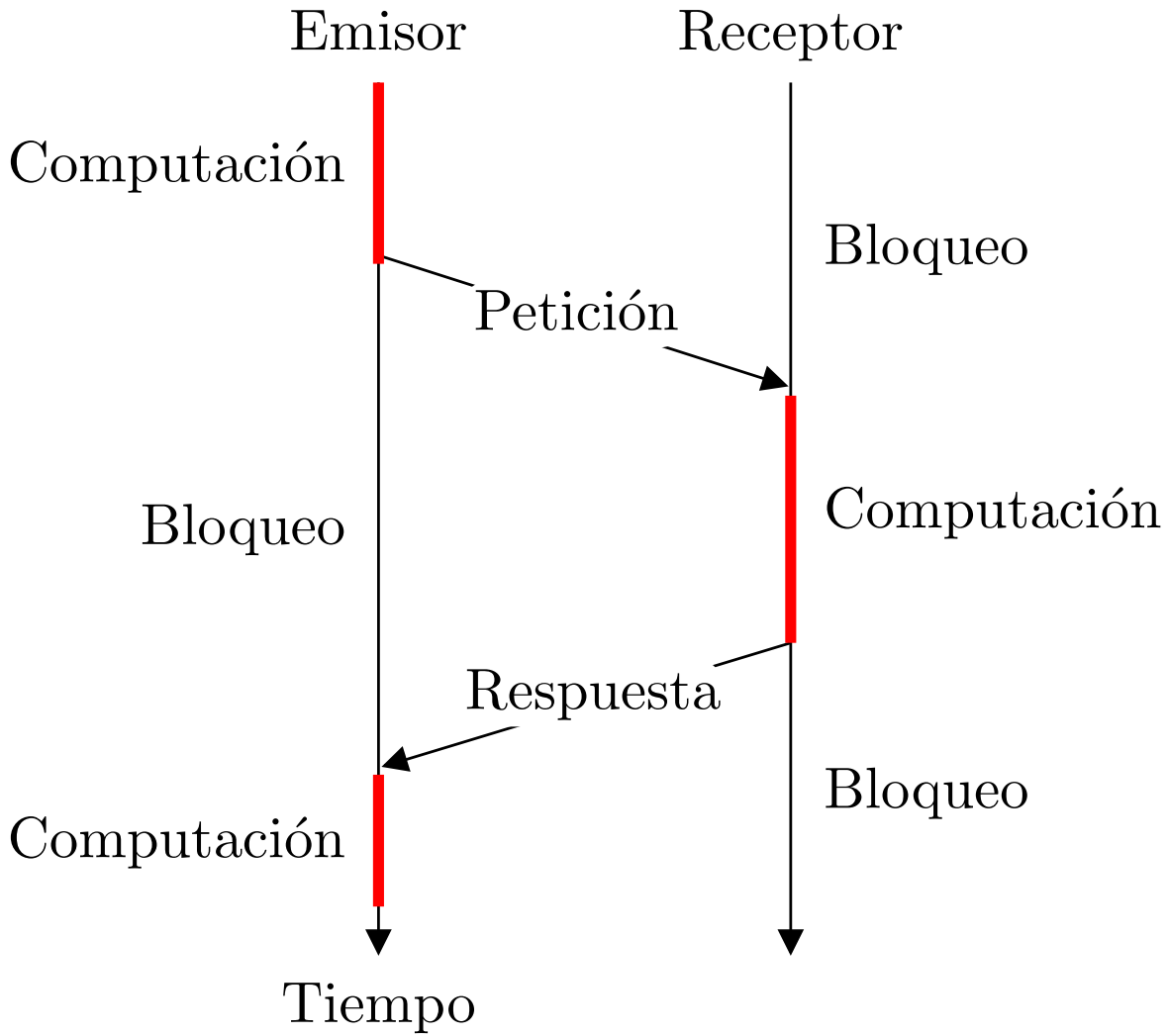
[2] En Unix, esta información el S.O. la almacena en el fichero `/etc/services`.

Capítulo 4

El mecanismo RPC (Remote Procedure Call)

4.1. Características

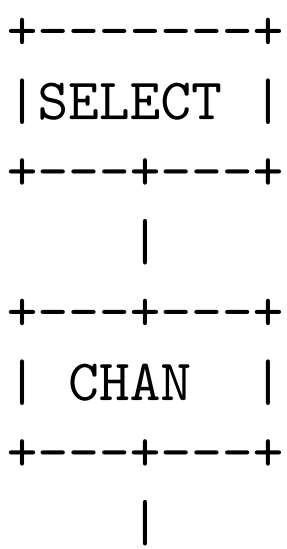
- Basado en el paradigma de comunicación cliente/servidor.
- Permite que el cliente invoque procedimientos (código ejecutable perteneciente a un determinado proceso) en el servidor [**?**].
- La sintasis para el cliente es la misma que si invocara a un procedimiento local (de ahí su nombre).
- Al igual que ocurre con la ejecución de un procedimiento local, el proceso llamador espera (bloqueado) a que el procedimiento remoto termine de ejecutarse.

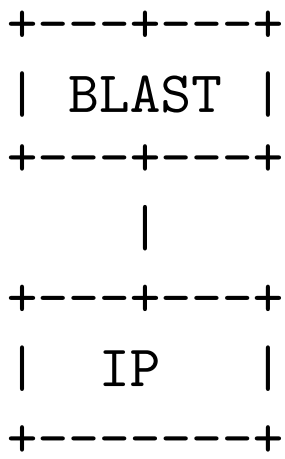


- Para que las aplicaciones usen el RPC, éste debe estar soportado por el compilador.
- Utilizado por aplicaciones como el NFS.

4.2. Microprotocolos del mecanismo RPC

- RPC utiliza tres microprotocolos:
  1. BLAST: fragmenta y ensambla mensajes grandes.
  2. CHAN: sincroniza los mensajes de petición y respuesta.
  3. SELECT: encamina los mensajes de petición al proceso correcto.
- Con todo esto, la pila de protocolos más simple que puede usar RPC sería:

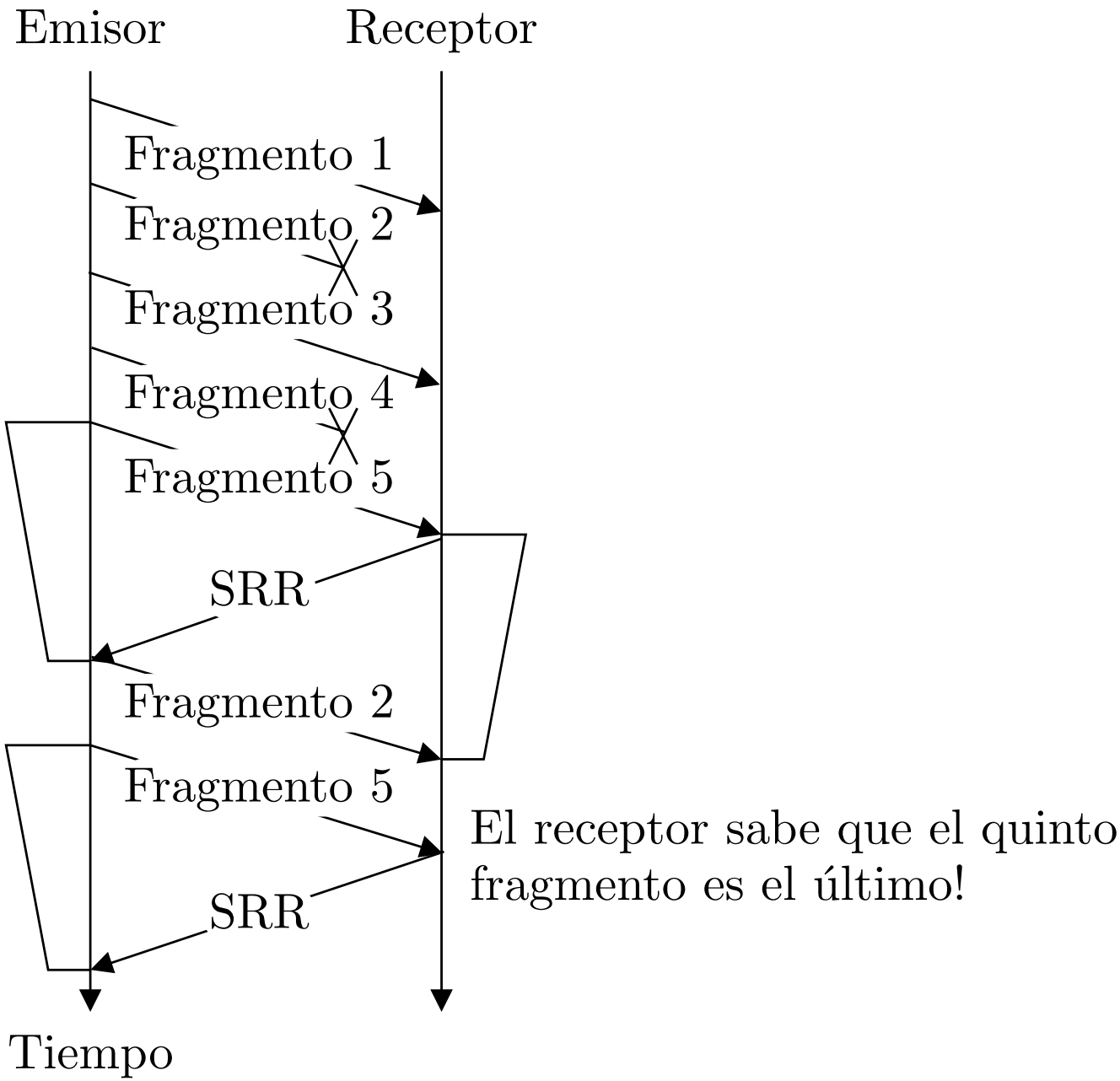



BLAST
IP

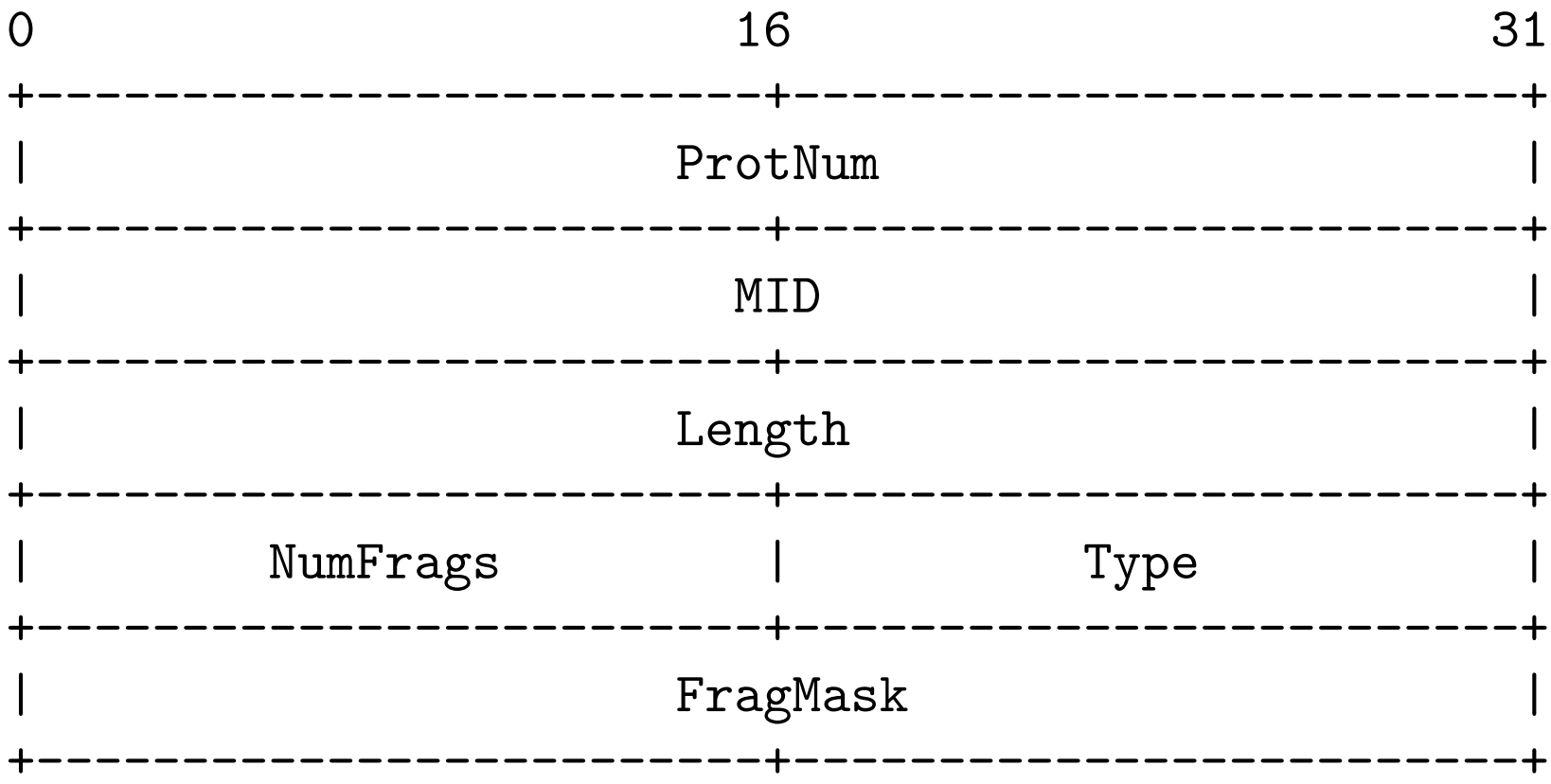
4.2.1. BLAST

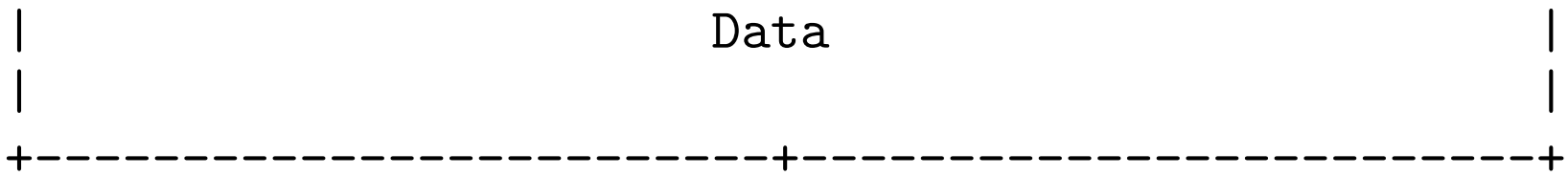
- Acomoda la longitud de los mensajes al MTU (Maximun Transfer Unit) de la red.[1]
- Semejante al TCP, aunque no está orientado a la transmisión de un flujo de datos.
- Soluciona la pérdida de paquetes mediante retransmisión selectiva (SRR = Selective Retransmission Request). Ejemplo:

[1] El IP también tiene esta funcionalidad, pero si un fragmento se pierde BLAST lo retransmite seletivamente. El IP retransmitiría todo el mensaje.


Emisor
Receptor
Fragmento 1
Fragmento 2
Fragmento 3
Fragmento 4
Fragmento 5
SRR
Fragmento 2
Fragmento 5
El receptor sabe que el quinto fragmento es el último!
SRR
Tiempo

- BLAST no garantiza la transmisión sin errores. Por ejemplo, si se pierden todos los paquetes de datos, no se enviará ningún SRR (que sólo se envían cuando hay errores). Por tanto, el mensaje se perderá.
- Cabecera de un paquete BLAST:





`ProtNum:` protocolo que utiliza BLAST.

`MID:` número de secuencia que identifica de forma única el mensaje.

`Length:` longitud del fragmento.

`NumFrags:` número de fragmentos del mensaje.

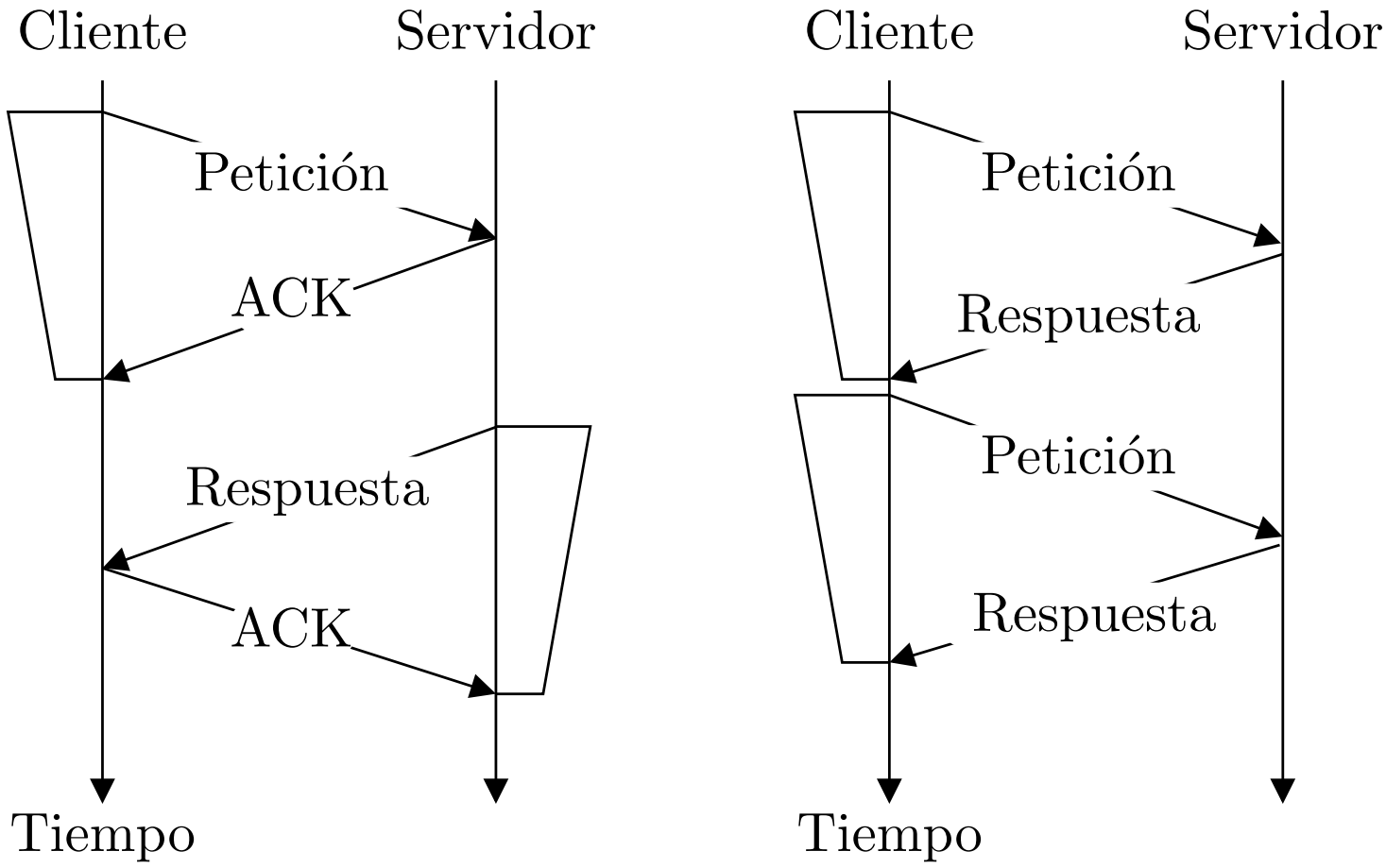
`Type:` DATA | SRR.

`FragMask:` Cuando se trata de un paquete con datos, sólo un bit puede estar a 1 e el índice del paquete dentro de la secuencia de fragmentación. Cuando se trata de un paquete SRR, un bit a 1 indica qué paquete de datos debe ser retransmitido. Nótese que no se pueden transmitir mensajes que al fragmentarse generan más de 32 paquetes.

4.2.2. CHAN(nel)

- Implementa el mecanismo petición/respuesta del RPC.
- Convierte al RPC en fiable.
- Sincroniza a los procesos que se comunican.[2] usando un protocolo en el que cada mensaje se reconoce positivamente (ACK):

[2]Sincronización = la operación `send()` termina cuando el receptor ha recibido los datos correctamente y además, el emsior lo sabe y recibe alguna contestación. Se dice en este caso que el `send()` es bloqueante.



Funcionamiento básico

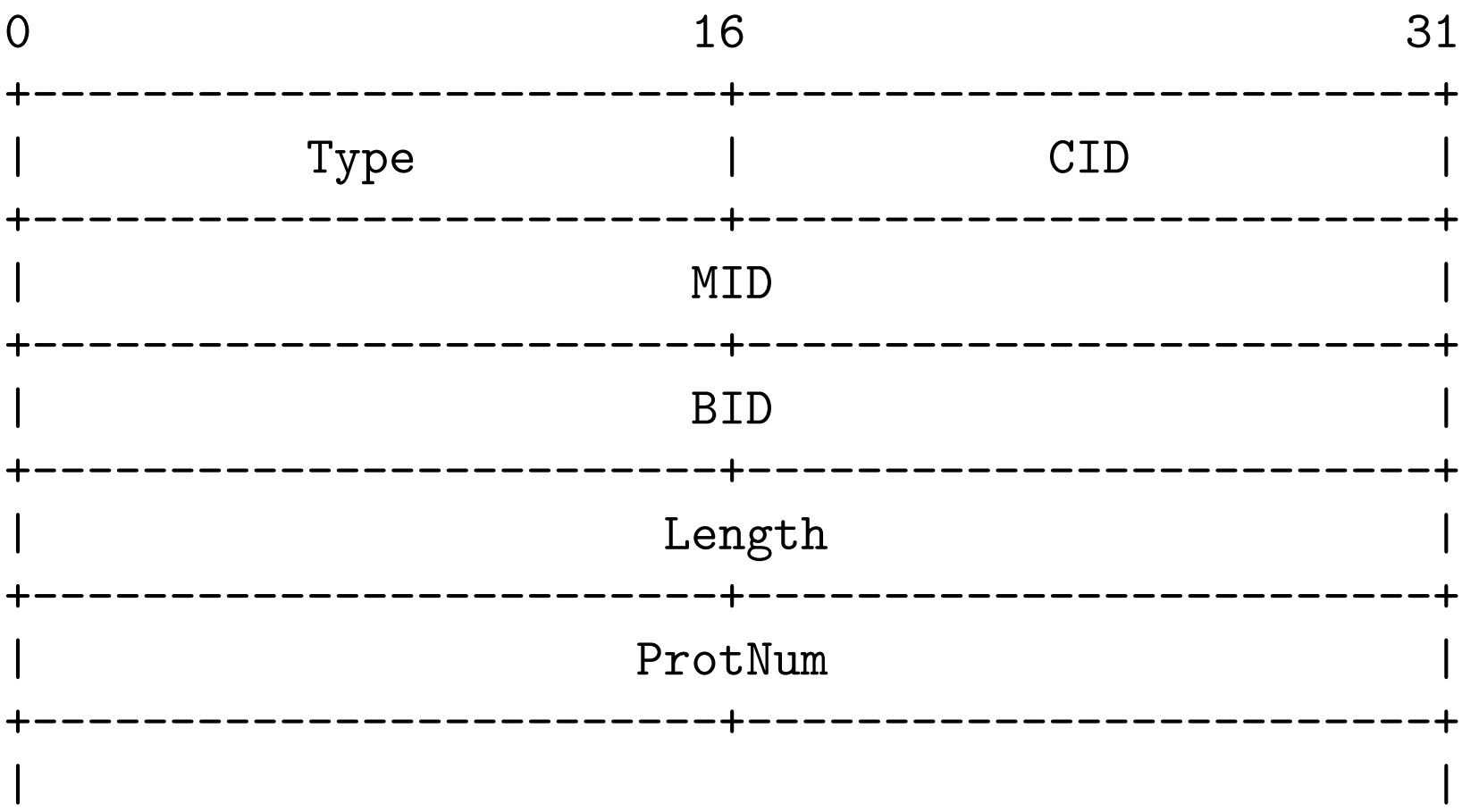
Uso de ACKs implícitos

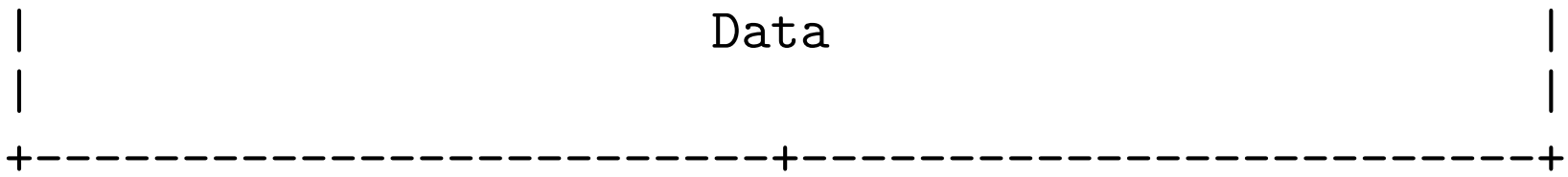
aunque existe un modo donde se suprimen los reconocimientos para acelerar la comunicación.

- Se crea un canal por cada interacción petición/respuesta (llamada

a un procedimiento remoto) que se produce.

- Más de un canal puede usarse al mismo tiempo. Esto es muy útil si las peticiones tardan mucho tiempo en responderse (pipelining).
- Formato de la cabecera CHAN:





Type: REQ(uest) | REP(ly) | ACK(nowlegment) | PROBE (are you alive?).

CID (Channel ID): número de secuencia que identifica de forma única el canal (2^{16} canales simultaneos).

MID (Message ID): número de secuencia que identifica de forma única el mensaje. Sirve para descartar duplicados.

BID (Boot ID): número de secuencia que identifica de forma única el número de rebotes de la máquina. Sirve, junto con el MID, para descartar duplicados.

Length: longitud del mensaje.

ProtNum: protocolo que usa CHAN.

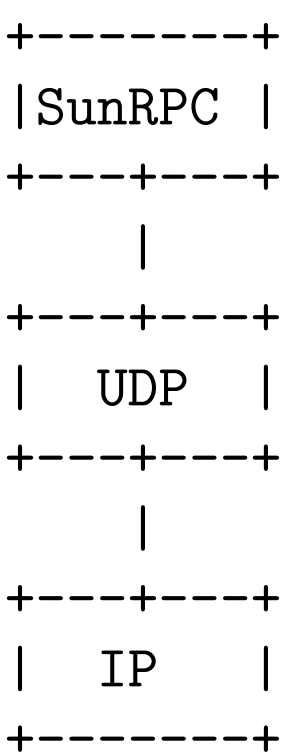
4.2.3. SELECT

- Implementa la funcionalidad del desmultiplexado dentro del RPC: En el servidor hay un número determinado de procedimientos que pueden ser llamados por el cliente. Dichos procedimientos se enumeran y SELECT permite que el cliente pueda invocar a uno en concreto.

- La forma de enumeración más frecuente es la jerárquica: Cada proceso en el servidor tiene un número de proceso, y cada proceso enumera internamente sus procedimientos. El número de bits dedicados a la selección depende de la versión del RPC y del SO.

4.3. El caso particular de SunRPC

- SunRPC es una versión del sistema RPC desarrollado por Sun Microsystems para su SunOS. Es la implementación del RPC más extendida.
- Implementa una versión del RPC distinta de la planteada anteriormente, cuyo grafo de protocolos consiste en:



- El IP sustituye a BLAST (aunque con cierta pérdida de eficiencia) en la tarea de fragmentar los mensajes demasiado largos.
- El UDP sustituye en parte a SELECT ya que permite desmultiplexar al proceso correcto usando un puerto.
- Finalmente, SunRPC implementa la funcionalidad de CHAN y de SELECT, aunque de una forma diferente:
  - SunRPC corre como un demonio llamado *Port Mapper* que escucha, por defecto, en el puerto 111.
  - Cuando un cliente remoto solicita la ejecución de un procedimiento local, primero debe conocer en qué puerto está escuchando el proceso local que posee el procedimiento. Esta información la obtiene preguntando al *Port Mapper*.
  - Seguidamente, el cliente realiza la petición RPC al correspondiente proceso utilizando el puerto retornado en la anterior consulta.

- Los cliente normalmente cachean el resultado de la consulta realizada al *Port Mapper*, lo que no degrada el rendimiento en sucesiva llamadas a procedimientos remotos.

4.4. Otras implementaciones

- Las versiones actuales de RPC tienden a usar el TCP en lugar del UDP por diversas razones:

  1. En muchos cortafuegos el tráfico UDP está restringido.
  2. El mecanismo de control de la congestión del TCP es necesario en las transmisiones a larga distancia.
  3. El TCP es fiable mientras que el UDP no lo es.
  4. La pérdida de un segmento no implica la retransmisión del mensaje completo.