ETSIIT
Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Ingeniería Informática + ADE

Universidad de Granada (UGR)

Autor: Ismael Sallami Moreno

Asignatura: Tema 5: Capa de Aplicación (FR)

Índice

1 Introducción a las aplicaciones de red

1.1. Aquitectura cliente - servidor

Las aplicaciones de red permiten la comunicación entre dispositivos conectados a través de Internet. Esta interacción se organiza principalmente en dos roles fundamentales: el servidor y los clientes. A continuación, se describen sus características principales:

1.1.1. Servidor

El servidor es el núcleo de la interacción en red y se caracteriza por las siguientes propiedades:

- **Siempre en funcionamiento:** Los servidores están diseñados para operar de manera continua, asegurando que los servicios estén disponibles en cualquier momento.
- **IP permanente y pública:** Para ser accesibles desde cualquier lugar, los servidores cuentan con una dirección IP fija y visible públicamente.
- **Agrupados en "granjas":** En la mayoría de los casos, los servidores no operan de manera aislada, sino que están organizados en grandes conjuntos llamados granjas de servidores, que proporcionan alta capacidad y redundancia.
- **Enlaces relacionados:**
  - Enlace 1: pincha aquí.
  - Enlace 2: pincha aquí.

1.1.2. Clientes

Los clientes son los dispositivos finales que interactúan con los servidores para obtener datos o servicios. Tienen las siguientes características:

- **Funcionando intermitentemente:** A diferencia de los servidores, los clientes no están activos todo el tiempo; se conectan y desconectan según sea necesario.
- **IP dinámica y privada:** Los clientes suelen usar direcciones IP que pueden cambiar con cada conexión (dinámicas) y que son invisibles fuera de la red local (privadas).
- **Comunicación con el servidor:** Los clientes dependen de los servidores para obtener los datos que solicitan; no pueden funcionar de manera independiente.
- **Sin comunicación entre clientes:** Generalmente, los clientes no se comunican directamente entre sí, lo que asegura que todas las interacciones pasen a través del servidor.

1.2. Procesos cliente y servidor

- Proceso cliente: es el proceso que se encarga de iniciar la comunicación.
- Proceso servidor: es el proceso que espera a ser contactado para iniciar la comunicación, además posee una IP pública y permamente.

Ambos procesos envían o reciben mensajes desde su socket. Para que puedan recibir mensajes un proceso debe de tener un identificador que normalmente se forma con la dirección IP y el puerto.

La mayoría de las transacciones entre aplicaciones se basan en el paradigma cliente-servidor:

- Servidor: programa que ofrece un servicio accesible desde la red.
- Cliente: programa que envía peticiones y espera respuestas a través de la red.
- Diferencias:
  - El servidor empieza antes (apertura pasiva).
  - El servidor se ejecuta de forma permanente.
  - El servidor usa puertos reservados.

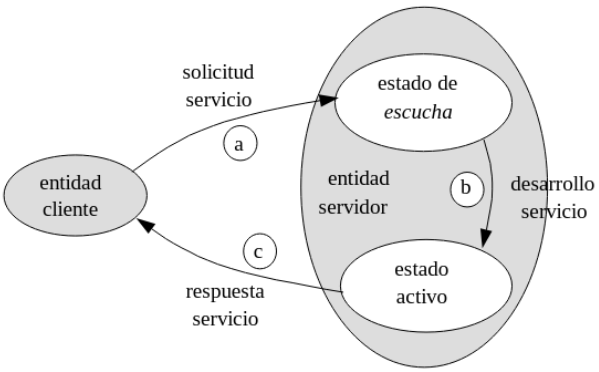

Figura 1: Esquema de la arquitectura cliente-servidor.

1.3. El software TCP/IP como parte del sistema operativo

El software TCP/IP es una pieza fundamental integrada en el sistema operativo. Este software proporciona una interfaz para que las aplicaciones puedan comunicarse a través de redes utilizando los protocolos TCP/IP. Las características principales son:

- **Interacción mediante la interfaz socket:** Las aplicaciones acceden al software TCP/IP a través de una API consistente basada en llamadas al sistema. Esta interfaz es conocida como *la interfaz socket*.

- **Definición independiente de los protocolos:** Aunque está diseñada para operar con TCP/IP, la definición de la interfaz socket no forma parte de ningún protocolo en particular, lo que permite flexibilidad en su implementación.
- **Diversas implementaciones disponibles:** Existen varias implementaciones de la interfaz socket adaptadas a diferentes sistemas operativos y entornos, entre las que se encuentran:
  - **Berkeley Socket Distribution:** Una implementación histórica y ampliamente utilizada en sistemas Unix.
  - **Winsock:** La versión de la interfaz socket diseñada para sistemas Windows.
  - **Transport Layer Interface (TLI):** Otra alternativa desarrollada para proporcionar servicios de transporte en redes.

1.4. La interfaz socket (BSD)

La interfaz *socket* es una extensión de la gestión de entrada/salida (I/O) tradicional del sistema operativo, diseñada para habilitar conexiones en red. Sus conceptos clave son:

- **Gestión de I/O en el sistema operativo:** La entrada/salida en el sistema operativo sigue el modelo básico de:
  - Abrir (`open`): Inicia el acceso a un recurso.
  - Leer/escribir (`read/write`): Opera sobre el recurso.
  - Cerrar (`close`): Finaliza el acceso al recurso.

  Estos pasos se implementan mediante llamadas al sistema (`syscalls`).
- **El descriptor de fichero:** Es un concepto esencial que representa un identificador único para el recurso abierto, como un archivo o una conexión de red.
- **Extensión a conexiones en red:** La interfaz socket amplía el modelo de I/O para incluir la gestión de conexiones en red. Esto requiere identificar los puntos finales de la comunicación, definidos por:
  - Dirección IP local.
  - Dirección IP remota.
  - Puerto local.
  - Puerto remoto.
- **Definición de un socket:** Un socket es un descriptor que representa una transmisión a través de la cual una aplicación puede enviar y/o recibir datos hacia y/o desde otro proceso de aplicación. Funciona como una "puerta" que conecta la aplicación con los servicios de transporte de red.

1.4.1. Definición y estructura de un socket

Un *socket* es un concepto fundamental en la programación de redes y puede definirse como:

- **Un descriptor de transmisión:** Representa una conexión a través de la cual una aplicación puede enviar y/o recibir información hacia y/o desde otro proceso de aplicación.
- **Puerta de acceso a los servicios de transporte:** Metafóricamente, un socket actúa como una "puerta" que conecta la aplicación con los servicios de transporte de red, facilitando la comunicación.
- **Implementación práctica:** En la práctica, un socket es una variable de tipo puntero a una estructura que contiene toda la información necesaria para gestionar la conexión, incluyendo los identificadores de los puntos finales (direcciones IP, puertos, etc.).

1.5. Operaciones básicas con sockets

En esta sección se describen las funciones y estructuras clave utilizadas para configurar y operar con sockets en la programación de redes.

1.5.1. Especificar dirección IP y puerto local

Para asociar un socket con una dirección IP y un puerto local, se utiliza la función `bind`:

```
int bind(int socket, struct sockaddr *myaddr, int addresslen);
```

La dirección se pasa mediante la estructura `sockaddr_in`:

```
struct sockaddr_in { /* Información de socket INET */
    short sin_family;       /* Familia: AF_INET */
    u_short sin_port;       /* Puerto: 16 bits, en orden de red */
    struct in_addr sin_addr; /* Dirección IP de 32 bits */
    char sin_zero[8];       /* No usada */
};
struct in_addr {
    u_long s_addr;          /* Dirección IP de 32 bits, en orden de red */
};
```

Para usar esta estructura, es necesario realizar un *casting*. Los campos incluyen:

- **sin_family:** Especifica la familia del protocolo, típicamente `AF_INET`.
- **sin_port:** Contiene el número de puerto.
- **sin_addr:** Almacena la dirección IP.

1.5.2. Especificar dirección IP y puerto remotos

Para conectarse a un destino remoto, se utiliza la función `connect`:

```
int connect(int socket, struct sockaddr *toaddr, int addresslen);
```

Dependiendo del tipo de socket:

- **SOCK_STREAM:** `connect` envía un SYN para iniciar el *handshake* TCP, lo que requiere la simultaneidad de ambos procesos.
- **SOCK_DGRAM:** No se envía nada a través de la red, pero se especifica la dirección IP y puerto remoto para facilitar el uso posterior.

1.5.3. Poner un socket en modo pasivo

Un servidor puede poner un socket en modo pasivo y definir el número de solicitudes pendientes en la cola con:

```
int listen(int sockfd, int maxwaiting);
```

Esta llamada no es bloqueante. Para aceptar una conexión y detener el flujo del programa hasta que llegue una solicitud, se usa:

```
int accept(int sockfd, struct sockaddr *fromaddrptr, int *addresslen);
```

`accept` devuelve un nuevo socket conectado para comunicarse con el cliente.

1.5.4. Enviar y recibir datos

Enviar datos: Para enviar datos, se pueden utilizar:

```
int sendto(int sockfd, char *buff, int bufflen, int flags,
           struct sockaddr *toaddrptr, int addresslen);
int send(int sockfd, char *buff, int bufflen, int flags);
```

Recibir datos: Para recibir datos, se emplean las funciones:

```
int recvfrom(int sockfd, char *buff, int bufflen, int flags,
             struct sockaddr *fromaddrptr, int *addresslen);
int recv(int sockfd, char *buff, int bufflen, int flags);
```

1.5.5. Cerrar el socket

Para cerrar un socket, se utilizan las funciones:

```
int close(int sockfd);
int shutdown(int sockfd, int how);
```

1.5.6. Otras funciones útiles

Existen otras llamadas relacionadas con sockets y redes:

- `read()`, `readv()`, `recvmsg()`
- `write()`, `writev()`, `sendmsg()`
- `gethostbyname()`, `getservbyname()`, `getprotobyname()`
- `htons()`, `htonl()`, `ntohs()`, `ntohl()`

1.5.7. Ficheros cabecera típicos

Las funciones anteriores requieren incluir los siguientes ficheros cabecera:

- `netinet/in.h`
- `sys/types.h`
- `sys/socket.h`

1.6. Tipos de servidores

Los servidores pueden clasificarse en diferentes categorías según criterios específicos. A continuación, se presentan los tipos principales:

1.6.1. Criterios de clasificación

- **Orientados a conexión vs. No orientados a conexión:**[1]
  - *Orientados a conexión:* Establecen y mantienen una conexión antes de transmitir datos, típicamente usando TCP.
  - *No orientados a conexión:* No requieren una conexión persistente, trabajando con paquetes independientes, típicamente usando UDP.
- **Iterativos vs. Concurrentes:**
  - *Iterativos:* Atienden una solicitud a la vez, procesando las solicitudes de manera secuencial.
  - *Concurrentes:* Atienden múltiples solicitudes simultáneamente, generalmente mediante el uso de hilos o procesos.

1.6.2. Tipos específicos de servidores

Servidor iterativo no orientado a conexión: Este tipo de servidor procesa las solicitudes una por una y no establece una conexión persistente con el cliente. Es eficiente para aplicaciones que usan UDP.

[1] Para imágenes detalladas, accede a las diapositivas 9-11 del tema 5.

Servidor iterativo orientado a conexión: Este servidor también atiende las solicitudes de manera secuencial, pero utiliza TCP para establecer una conexión con el cliente antes de procesar la solicitud.

Servidores concurrentes orientados a conexión: Estos servidores gestionan múltiples conexiones simultáneamente, lo que los hace ideales para aplicaciones que requieren manejar múltiples clientes en paralelo.

Servidores multiprotocolo con I/O asíncrona (select): Un único proceso iterativo ofrece soporte para múltiples protocolos. Utiliza I/O asíncrona y técnicas como `select()` para gestionar múltiples conexiones en una sola ejecución.

Servidores multiservicio: Estos servidores ofrecen múltiples servicios con concurrencia, permitiendo manejar diferentes tipos de solicitudes simultáneamente, a menudo mediante el uso de hilos o procesos separados para cada servicio.

1.7. Estimación de retardos en cola utilizando la teoría de colas

La teoría de colas es una herramienta matemática utilizada para modelar y analizar los tiempos de espera en sistemas donde los recursos son compartidos. En el caso de redes, puede aplicarse para estimar los retardos en cola en servidores y routers.

1.7.1. Modelo M/M/1

Un servidor se modela como un sistema **M/M/1**, que se caracteriza por:

- **Llegadas Poisson:** Las solicitudes llegan al sistema siguiendo una distribución de Poisson con una tasa promedio de llegadas λ.
- **Tiempo de servicio exponencial:** El tiempo de servicio de las solicitudes sigue una distribución exponencial con tiempo promedio T_s.
- **Un único servidor:** El sistema cuenta con un único recurso para procesar las solicitudes.

1.7.2. Fórmula del retardo en cola

El retardo promedio en cola (R) en un sistema M/M/1 puede calcularse con la siguiente fórmula:

$$R = \frac{T_s \cdot \lambda}{1 - \lambda \cdot T_s}$$

- T_s: Tiempo promedio de servicio.
- λ: Tasa promedio de llegadas de solicitudes.

[1] Para profundizar en el tema de implementación de un socket puedes revisar la implementación del mismo en las diapositivas 23-29 del tema 5, pero no es objeto evaluable de examen.

- El denominador $1 - \lambda \cdot T_s$ refleja la relación entre la capacidad del servidor y la carga del sistema.

1.7.3. Aplicación en routers

Esta misma fórmula puede utilizarse para calcular el retardo en cola en un **router**, considerando las llegadas de paquetes y el tiempo necesario para procesarlos.

- **Ejemplo:** Si un router procesa paquetes con un tiempo promedio de servicio T_s y recibe paquetes a una tasa de λ, el retardo promedio en cola puede calcularse utilizando la expresión anterior.

1.8. Definición y características de un protocolo de aplicación

Un protocolo de aplicación define las reglas y mecanismos mediante los cuales dos procesos se comunican en una red. Los aspectos principales que caracterizan un protocolo de aplicación incluyen:

1.8.1. Aspectos clave de un protocolo de aplicación

- **Tipo de servicio:**
  - *Orientado o no orientado a conexión.*
  - *Realimentado o no realimentado.*
- **Tipo de mensaje:**
  - *Request*, *response*, entre otros.
- **Sintaxis:** Define y estructura los "campos" en el mensaje.
  - En aplicaciones generalmente son orientados a texto (ej., HTTP).
  - Excepciones: algunos protocolos como DNS.
  - Tendencia: usar formato *Type-Length-Value (TLV)*.
- **Semántica:** Define el significado de los campos.
- **Reglas:** Especifican cuándo los procesos deben enviar mensajes o responder a mensajes.

1.8.2. Tipos de protocolos

- **Según propiedad:**
  - *Dominio público:* Definidos en RFCs (ej., HTTP, SMTP).
  - *Propietarios:* No estandarizados públicamente (ej., Skype, IGRP).
- **Según la transmisión:**
  - *In-band:* Los datos y el control viajan juntos.

- *Out-of-band:* Los datos y el control viajan por canales separados.

- **Según estado:**
  - *Stateless:* No mantienen información de estado entre transacciones.
  - *Stateful:* Mantienen información de estado.

- **Según persistencia:**
  - *Persistentes:* Mantienen la conexión abierta entre transacciones.
  - *No persistentes:* Cierran la conexión después de cada transacción.

1.8.3. Tendencias en los protocolos de aplicación

Los protocolos modernos tienden a ser flexibles mediante:

- Una **cabecera fija**.
- Una serie de **trozos** que pueden ser:
  - Obligatorios u opcionales.
  - Con cabeceras específicas más datos en forma de parámetros.

Formato TLV (Type-Length-Value):

- Parámetros fijos (en orden).
- Parámetros de longitud variable u opcionales.

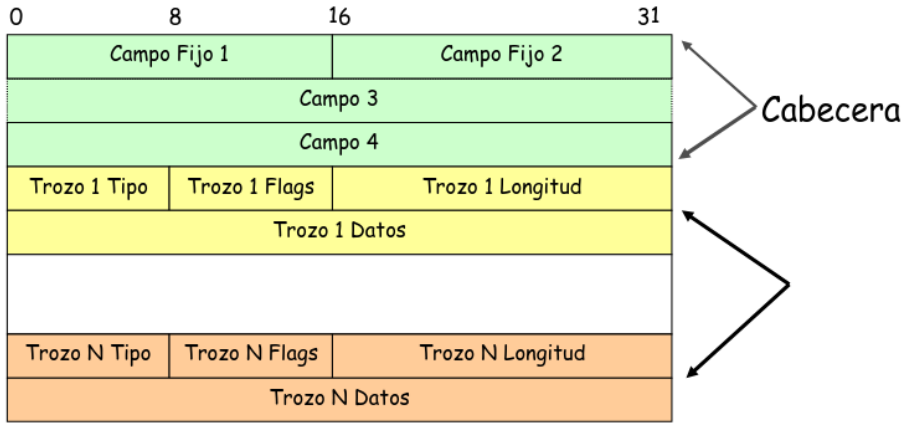

Figura 2: Estructura flexible en protocolos de aplicación.

1.9. Características y requisitos de las aplicaciones

Las aplicaciones de red tienen diferentes características y requisitos según sus necesidades de funcionamiento. A continuación se describen algunos de los aspectos más importantes:

- **Tolerancia a pérdidas de datos (errores):** Algunas aplicaciones pueden tolerar la pérdida de datos sin afectar significativamente su rendimiento, como en el caso de aplicaciones de audio. En cambio, otras aplicaciones, como FTP, Telnet o HTTP, requieren una transferencia de datos completamente fiable.
- **Exigencia de requisitos temporales:** Las aplicaciones inelásticas, como la telefonía por Internet y los juegos interactivos, requieren un retardo (delay) acotado para ser efectivas. Otras aplicaciones, en cambio, no tienen estrictos requisitos de tiempo y pueden tolerar mayores retardos sin afectar la experiencia del usuario.
- **Demanda de ancho de banda (tasa de transmisión o throughput):** Algunas aplicaciones requieren un ancho de banda constante o una tasa de transmisión específica, como los códecs de vídeo. Otras aplicaciones, como la navegación web, pueden ser más flexibles en cuanto a la cantidad de datos transmitidos.
- **Nivel de seguridad:** Los requisitos de seguridad varían entre aplicaciones. Algunas pueden necesitar características avanzadas como encriptación, autenticación, no repudio e integridad de los datos, mientras que otras tienen requisitos de seguridad más bajos.

1.9.1. Conclusión

En resumen, las distintas aplicaciones tienen requisitos heterogéneos, lo que significa que no existe un único estándar que se adapte a todas las aplicaciones. Estos requisitos deben ser gestionados adecuadamente para garantizar un funcionamiento eficiente y seguro de las aplicaciones de red.

1.10. Requerimientos de algunas aplicaciones

Application	Data loss	Throughput	Time Sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video:10kbps-5Mbps	yes, 100's ms
stored audio/video	loss-tolerant	same as above	yes, few s
interactive games	loss-tolerant	few kbps up	yes, 100's ms
instant messaging	no loss	elastic	yes and no

Figura 3: Requerimientos de algunas aplicaciones.

1.11. Protocolos de Transporte

Los protocolos de transporte son fundamentales para la comunicación en redes, ya que gestionan la transferencia de datos entre aplicaciones en diferentes dispositivos. Dos de los protocolos más utilizados en esta capa son **TCP** y **UDP**.

1.11.1. Introducción a las aplicaciones de red

TCP y UDP, como protocolos de la capa de transporte, se apoyan en el protocolo IP de la capa de red. Sin embargo, es importante destacar que no garantizan *Calidad de Servicio* (QoS), lo que significa que:

- El **retardo no está acotado**.
- Las **fluctuaciones en el retardo no están acotadas**.
- No hay una **velocidad de transmisión mínima garantizada**.
- No existe una **probabilidad de pérdidas acotada**.
- Tampoco hay garantías de **seguridad**.

1.11.2. Servicio TCP

El protocolo TCP (Transmission Control Protocol) ofrece un servicio **orientado a conexión** que garantiza un transporte fiable, es decir, asegura que los datos lleguen sin errores y en el orden correcto. Sus características incluyen:

- **Transporte fiable con control de errores.**
- **Control de flujo.** Asegura que el receptor no sea sobrecargado con datos.
- **Control de congestión.** Ajusta la cantidad de datos enviados para evitar la congestión en la red.

1.11.3. Servicio UDP

El protocolo UDP (User Datagram Protocol), en contraste con TCP, es **no orientado a conexión** y no garantiza la fiabilidad en la transmisión de datos. Sus características incluyen:

- **Transporte no fiable.** Los datos pueden perderse sin ser retransmitidos.
- **Sin control de flujo.** No hay mecanismos para regular la cantidad de datos que se envían.
- **Sin control de congestión.** No hay medidas para prevenir la sobrecarga en la red.

1.11.4. ¿Para qué existe UDP?

UDP es útil en situaciones donde la rapidez es más importante que la fiabilidad, como en aplicaciones de streaming de video, voz sobre IP (VoIP) y juegos en línea, donde la transmisión rápida de datos es crucial y se puede tolerar alguna pérdida de información.

Application	Application layer protocol	Underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	HTTP (eg Youtube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	typically UDP

Figura 4: Comparación de los servicios TCP y UDP.

2 Servicio de Nombres de Dominio (DNS)

2.1. La comunicación en Internet y las direcciones IP

La comunicación en Internet depende de las direcciones IP para identificar a los dispositivos y permitir la transmisión de datos. Sin embargo, los usuarios prefieren utilizar **nombres de dominio**, ya que son mucho más fáciles de recordar que una dirección IP numérica. Actualmente, existen más de 300 millones de nombres de dominio registrados.

2.1.1. Sistema de nombres de dominio (DNS)

El **DNS** (Domain Name System) es el sistema que permite traducir los nombres de dominio a direcciones IP, un proceso conocido como *resolución de nombres*. Por ejemplo, el nombre de dominio `dns.ugr.es` se traduce a la dirección IP `150.214.204.10`.

2.1.2. Estructura jerárquica de los dominios

El sistema de nombres de dominio tiene una estructura jerárquica, donde los dominios se organizan de la siguiente manera:

Parte_local.dominio_nivel1. . . . .dominio_nivel2.dominio_nivel1.

El **dominio de nivel 1** es el más cercano al final y generalmente es un dominio genérico, como `.com`, `.es`, `.edu`, etc.

El **dominio raíz** o `.` es el nivel más alto de la jerarquía y es gestionado por **ICANN** (Internet Corporation for Assigned Names and Numbers). ICANN delega la gestión de algunos dominios genéricos a centros regionales de autoridad.

2.1.3. Dominios genéricos

Inicialmente, se definieron los siguientes nueve dominios genéricos en el RFC 1591:

- `.com` ->Organizaciones comerciales.
- `.edu` ->Instituciones educativas, como universidades, en los Estados Unidos.
- `.gov` ->Instituciones gubernamentales de los Estados Unidos.
- `.mil` ->Organizaciones militares de los Estados Unidos.
- `.net` ->Proveedores de Internet.
- `.org` ->Organizaciones diversas que no encajan en las categorías anteriores.
- `.arpa` ->Usado para propósitos exclusivos de infraestructura de Internet.
- `.int` ->Organizaciones establecidas por tratados internacionales entre gobiernos.
- `.xy` ->Indicativos de zona geográfica, como `.es` para España, `.pt` para Portugal, `.jp` para Japón, etc.

2.1.4. Lecturas recomendadas

Para profundizar más sobre el tema de los nombres de dominio y el sistema DNS, se recomienda la siguiente documentación:

- **Tutorial sobre "Los nombres de dominios":** `https://www.icann.org/en/system/files/files/domain-names-beginners-guide-06dec10-es.pdf`
- **Instrucciones para registrar un nombre de dominio en .es:** `https://www.dominios.es/es/registra-un-dominio/como-registrar-dominio`
- **Instalación y ejemplos de ficheros de configuración de `named`:** `https://www.tldp.org/HOWTO/DNS-HOWTO.html`

2.2. DNS: Protocolo de aplicación para la resolución de nombres

DNS (Domain Name System) es un protocolo de aplicación utilizado para acceder a una base de datos distribuida que gestiona la resolución de nombres en Internet. Este protocolo permite traducir los nombres de dominio a direcciones IP, facilitando la comunicación entre los dispositivos de la red.

2.2.1. Niveles de servidores DNS

El sistema DNS se organiza en tres niveles de servidores, que gestionan la resolución de nombres de dominio:

- **Servidores raíz (`.`)**: Son los servidores más altos en la jerarquía de DNS y gestionan la información sobre los servidores de dominio de nivel superior (TLD).
- **Servidores de dominio (`TLD`)**: Estos servidores gestionan los dominios de nivel superior como `.com`, `.org`, `.edu`, etc.
- **Servidores locales**: Son servidores DNS específicos para una red local o proveedor de servicios de Internet, encargados de resolver las consultas locales de los usuarios.

2.2.2. Proceso de resolución de nombres

Cuando un usuario quiere acceder a un sitio web como `jcp.ugr.es` o `www.google.com`, el proceso de resolución de nombres sigue estos pasos:

1. El **resolver local** realiza una consulta para resolver el nombre de dominio.
2. Se establece una conexión con el servidor DNS local, cuyo IP es conocida previamente.
3. El servidor DNS local lleva a cabo la resolución del nombre de dominio.

2.2.3. Tipos de resolución

Existen dos tipos de resolución que se utilizan en el proceso de obtención de la dirección IP asociada a un nombre de dominio:

- **Resolución iterativa**: El servidor DNS responde con la mejor información disponible, indicando a la aplicación que consulte otros servidores si es necesario.
- **Resolución recursiva**: El servidor DNS se encarga de realizar todas las consultas necesarias hasta encontrar la respuesta completa y devolverla al cliente.

2.2.4. Mejoras en el rendimiento: Uso de caches

Para mejorar el rendimiento y reducir el tiempo de respuesta, los servidores DNS utilizan **caches** para almacenar las respuestas a las consultas más frecuentes. De esta manera, si un nombre de dominio ha sido resuelto recientemente, el servidor puede devolver la dirección IP almacenada en lugar de realizar una consulta a los servidores superiores[2].

[2]Para más imágenes consulte la diapositiva 43 del tema 5.

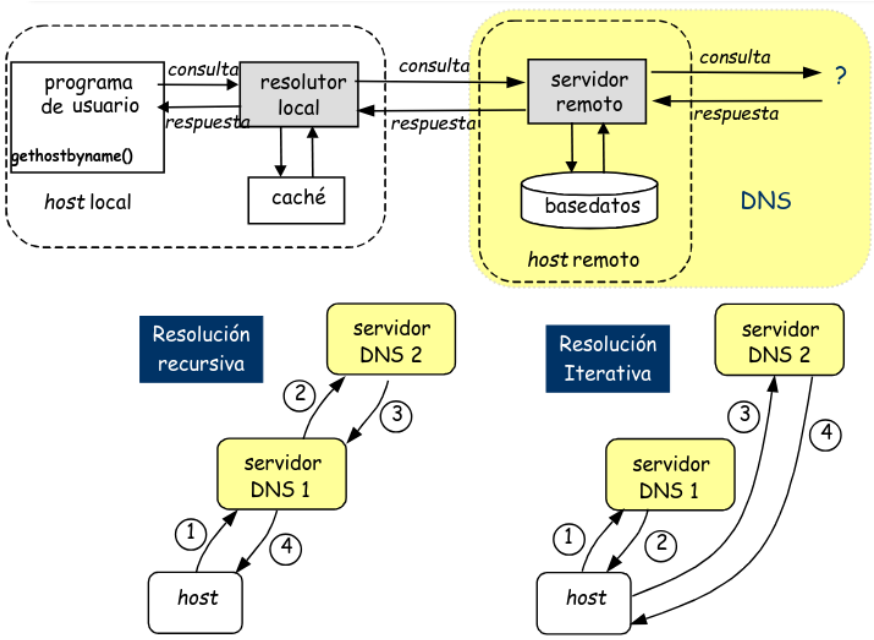

Figura 5: Esquema del funcionamiento del sistema DNS.

2.3. Gestión de la base de datos distribuida y jerárquica en DNS

El sistema DNS está formado por una base de datos distribuida y jerárquica, gestionada por un conjunto de servidores cooperativos que almacenan parcialmente la información. Esta base de datos se denomina **BIND** (Berkeley Internet Name Domain).

2.3.1. Zonas en DNS

Cada servidor DNS es responsable de una **zona**. Una zona es un conjunto de nombres de dominio contiguos dentro de un nodo del árbol DNS. El servidor tiene toda la información sobre esa zona y es considerado su autoridad.

- **Servidor de autoridad (SOA, Start of Authority)**: Los servidores de autoridad deben contener toda la información de su zona, no solo la cacheada.
- **Delegación jerárquica**: La autoridad puede delegarse a otros servidores en niveles más bajos de la jerarquía del DNS.

2.3.2. Gestión de la base de datos DNS

La gestión de la base de datos DNS se organiza de la siguiente manera:

- Cada zona debe tener al menos un **servidor de autoridad**.
- En cada zona existen servidores **primarios**, que almacenan una copia maestra de la base de datos en discos locales, y servidores **secundarios**, que obtienen la base de datos por transferencia.
- Además, existe un servicio de **cache** para mejorar el rendimiento de las consultas.

2.3.3. Topología de servidores DNS

La topología real de los servidores DNS es compleja. Existen 13 servidores raíz, identificados por las letras A a M, que forman el núcleo de la infraestructura DNS global. Los servidores raíz tienen copias distribuidas de los datos, y algunos de ellos están ubicados en diferentes puntos del mundo, como en Madrid, en el punto neutro `Espanix`.

2.3.4. Resolución de nombres por el resolver local

Cuando un cliente realiza una consulta DNS a través de un **resolver local**, pueden ocurrir varios escenarios:

- **Respuesta con autoridad**: El servidor tiene autoridad sobre la zona que contiene el nombre solicitado y devuelve la dirección IP correspondiente.
- **Respuesta sin autoridad**: El servidor no tiene autoridad sobre la zona solicitada, pero tiene la respuesta almacenada en su cache.
- **No conoce la respuesta**: El servidor consulta otros servidores de forma recursiva o iterativa. Normalmente, la petición se eleva a uno de los servidores raíz.

2.3.5. Servidores raíz DNS

Los servidores raíz DNS son gestionados por diferentes organizaciones alrededor del mundo. A continuación, se enumeran algunos de los servidores raíz:

- **Servidor A**: Network Solutions, Herndon, Virginia, USA.
- **Servidor B**: Instituto de Ciencias de la Información de la Universidad del Sur de California, USA.
- **Servidor C**: PSINet, Virginia, USA.
- **Servidor D**: Universidad de Maryland, USA.
- **Servidor E**: NASA, Mountain View, California, USA.
- **Servidor F**: Internet Software Consortium, Palo Alto, California, USA.
- **Servidor G**: Agencia de Sistemas de Información de Defensa, California, USA.
- **Servidor H**: Laboratorio de Investigación del Ejército, Maryland, USA.
- **Servidor I**: NORDUnet, Estocolmo, Suecia.
- **Servidor J**: (TBD), Virginia, USA.
- **Servidor K**: RIPE-NCC, Londres, Inglaterra.
- **Servidor L**: (TBD), California, USA.
- **Servidor M**: Wide Project, Universidad de Tokyo, Japón.

2.4. Base de datos DNS

En el sistema DNS, todo dominio está asociado al menos a un **registro de recurso** (Resource Record, RR). Cada RR es una tupla con cinco campos:

- **Nombre del dominio**: El nombre del dominio al que se refiere el registro.
- **Tiempo de vida**: Tiempo de validez del registro, utilizado para la cache de las consultas.
- **Clase**: En Internet, siempre es `IN` (Internet).
- **Tipo**: El tipo de registro, que define el propósito del RR.
- **Valor**: El contenido que depende del tipo de registro.

Los tipos de registros más comunes son:

- **SOA (Start Of Authority)**: Indica la autoridad de la zona.
- **NS (Name Server)**: Contiene información sobre los servidores de nombres de la zona.
- **A (Address)**: Define una dirección IPv4 asociada a un nombre de dominio.
- **MX (Mail Exchange)**: Define un servidor de correo electrónico para el dominio.
- **CNAME (Canonical Name)**: Define el nombre canónico de un nombre de dominio.
- **HINFO (Host Information)**: Contiene información sobre el tipo de máquina y el sistema operativo.
- **TXT (Text)**: Contiene información textual sobre el dominio.

Además, existe una base de datos asociada de **resolución inversa** para traducir direcciones IP en nombres de dominio. Esta base de datos se encuentra bajo el dominio `in-addr.arpa`[3].

2.5. Formato de mensajes DNS

DNS generalmente utiliza el puerto 53 y se comunica a través de **UDP**, aunque utiliza **TCP** para respuestas grandes (cuando el tamaño de la respuesta es mayor a 512 bytes).

Para más información, se pueden consultar los siguientes documentos:

- RFC 1034 y RFC 1035 (actualizados por los RFC 3597 y 3658)
- `/usr/doc/HOWTO/trans/es/DNS-COMO`

[3]Ejemplos de estos registros se encuentran en la diapositiva 49 del Tema 5.

- Comandos como `man named`, `nslookup`, `resolver`, `host.conf`, `dig`
- DNSSEC: `http://www.dominios.es/dominios/sites/dominios/files/1318333648229_0.pdf`

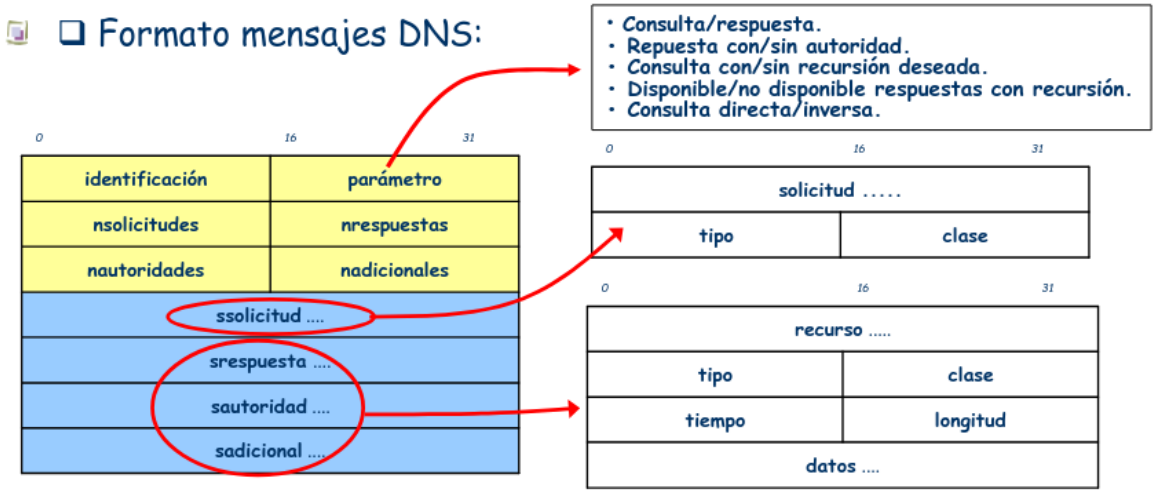

Figura 6: Formato de mensajes DNS

2.5.1. Ejemplo de solicitud DNS

En la siguiente figura, se ilustra un ejemplo de acceso DNS por parte de una máquina (`jcp.ugr.es`) que desea acceder a los servicios de `www.google.com`. Para obtener la dirección IP del servidor, la consulta debe pasar por varios servidores, como se muestra en el gráfico.

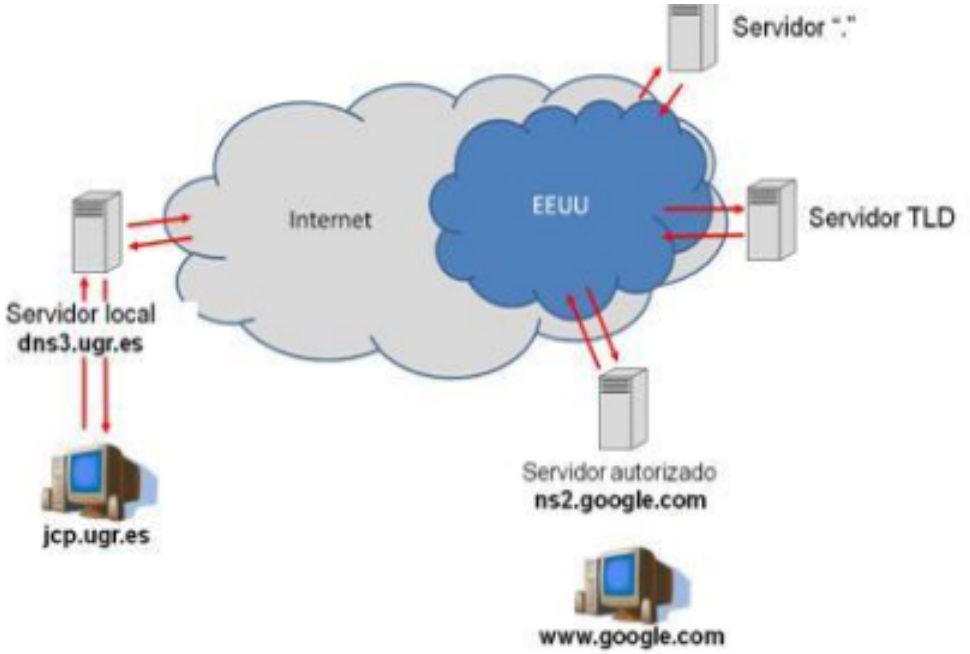

Figura 7: Ejemplo de acceso DNS de jcp.ugr.es a www.google.com

Supongamos que se consideran los siguientes retardos promedio:

- 8 μs dentro de una red LAN.
- 12 ms por cada acceso a través de Internet.
- 1 ms de procesamiento en cada servidor.
- 4 ms si la conexión se restringe a EE. UU.

Para calcular el tiempo que se tarda en procesar la solicitud bajo dos políticas diferentes de consulta, realizamos los siguientes cálculos:

1. Solicitud recursiva al servidor local, pero el servidor realiza solicitudes iterativas: En este caso, la solicitud recursiva al servidor local implica que el servidor local debe realizar consultas a otros servidores de forma iterativa. Considerando los servidores involucrados en la jerarquía de DNS (servidores raíz, servidores TLD, y servidores autoritativos), se suman los retardos asociados con cada acceso.

- Retardo en la LAN (8 μs).
- 1 ms de procesamiento en el servidor local.
- 12 ms por cada acceso a través de Internet para llegar a un servidor de nivel superior.
- 1 ms de procesamiento adicional en cada servidor intermedio.

La suma total para cada nivel de servidor debe repetirse dependiendo de cuántos servidores estén involucrados en la consulta (de acuerdo con el gráfico). Los tiempos de acceso se acumulan a medida que el servidor local hace las consultas iterativas.

2. Solicitud iterativa directamente a los servidores raíz: En este caso, el servidor local no realiza consultas recursivas. El cliente consulta directamente a los servidores raíz, lo que reduce el número de pasos involucrados en la resolución.

En este caso, se puede deducir que el tiempo total se reduce debido a que el servidor local no realiza múltiples consultas.

2.5.2. Comparación y desventajas

La política **recursiva** es más lenta debido a que cada servidor debe realizar consultas adicionales para resolver el nombre. En cambio, la política **iterativa** es más rápida porque el cliente directamente consulta a los servidores raíz y TLD.

Sin embargo, la desventaja de la política iterativa es que depende de la capacidad del cliente para manejar las consultas directamente a los servidores autoritativos. Además, la política iterativa puede resultar menos eficiente si el cliente no tiene configuraciones adecuadas para realizar consultas directas a varios servidores.

3 Navegación Web

3.1. Páginas Web y HTTP

Una página web es un archivo (generalmente en formato `HTML`) que está formado por diversos objetos, tales como:

- **Archivos HTML**
- **Imágenes** (por ejemplo, en formato JPEG)
- **Applets de Java**
- **Archivos de audio**
- **Archivos de vídeo**

Cada objeto dentro de la página web es direccionado a través de una **URL (Uniform Resource Locator)** o **URI (Uniform Resource Identifier)**. El formato de la URL sigue una estructura estándar, que puede desglosarse de la siguiente manera:

- `[usuario[:contraseña]@]dominio[:puerto][/ruta][/recurso][?solicitud][#fragmento]`

3.2. Protocolo HTTP

Las páginas web se sirven a través del protocolo **HTTP (Hypertext Transfer Protocol)**, que es un protocolo **cliente-servidor**. En este modelo:

- **Cliente**: Generalmente un navegador web (como Google Chrome, Firefox, etc.), que realiza una solicitud, recibe y muestra los objetos web.
- **Servidor**: Un servidor web, que recibe las solicitudes del cliente y responde enviando los objetos web correspondientes.

Las páginas web pueden clasificarse en dos tipos:

1. **Estáticas**: El contenido de estas páginas no cambia. Son simples y contienen los mismos datos para cada usuario.
2. **Dinámicas**: Estas páginas pueden cambiar su contenido en función de factores como la interacción del usuario, la hora, los datos extraídos de bases de datos, entre otros.

3.3. Páginas Dinámicas

Las páginas dinámicas generan contenido variable utilizando:

- **Lenguajes de scripting en el cliente**: Como `JavaScript` o `Flash`, que son ejecutados en el navegador del cliente.
- **Lenguajes de scripting en el servidor**: Como `Perl`, `PHP`, `Ruby`, `Python`, que se utilizan para procesar las solicitudes del cliente en el servidor, insertando contenido dinámico, como resultados de consultas a bases de datos. Estos lenguajes permiten incrustar etiquetas dentro del código HTML, y cuando el servidor recibe la solicitud, interpreta estas etiquetas para generar contenido dinámico.

3.4. Características de HTTP

- **Puerto de conexión**: HTTP utiliza el servicio de **TCP (Transmission Control Protocol)** en el puerto `80`. Esto significa que la comunicación entre el cliente y el servidor se realiza a través de conexiones TCP.
- **Modelo de Conexión**:
  1. **Inicio de Conexión TCP**: El cliente establece la conexión con el servidor.
  2. **Envío de solicitud HTTP**: El cliente realiza una solicitud HTTP al servidor.
  3. **Cierre de Conexión TCP**: Después de que el servidor envía la respuesta al cliente, la conexión TCP se cierra.

- **HTTP es "stateless"**: Esto significa que **HTTP no guarda el estado** de la sesión del cliente. Cada solicitud es independiente de las anteriores, lo que ahorra recursos en el servidor. Sin embargo, esto hace que la interacción entre el cliente y el servidor sea más compleja, ya que el servidor no mantiene ninguna información sobre las solicitudes previas. Para manejar el estado, se utilizan tecnologías como las **cookies**, que permiten a los servidores almacenar pequeños fragmentos de información en el navegador del cliente.

3.5. Tipos de Servidores HTTP

Existen dos tipos principales de servidores HTTP:

1. **Servidores No Persistentes**: En este tipo de servidores, solo se envía un objeto por cada conexión TCP. Esto significa que cada vez que el cliente solicita un nuevo objeto (por ejemplo, una imagen o un archivo HTML), se establece una nueva conexión TCP entre el cliente y el servidor.

2. **Servidores Persistentes**: En estos servidores, se pueden enviar múltiples objetos a través de una sola conexión TCP entre el cliente y el servidor. Esto mejora la eficiencia y reduce el tiempo necesario para cargar una página web, ya que no es necesario establecer una nueva conexión TCP para cada objeto solicitado.

3.6. Mensajes HTTP

1. El cliente HTTP (navegador) solicita un objeto identificado por su URL, en el ejemplo `www.ugr.es/pages/Universidad`. Según la configuración del servidor, si no se especifica nada, por defecto se sirve el fichero `index.html`.

2. El cliente consulta al resolver de DNS por la dirección IP de `www.ugr.es`.

3. DNS contesta `150.214.27.71` (IP virtual de un servicio balanceado).

4. El cliente abre una conexión TCP al puerto 80 de `150.214.27.71` (3 bandas).

5. El cliente envía una petición `GET /pages/universidad/ ...` (más otra información adicional: cabeceras, cookies, variables, etc).

6. El servidor responde enviando el fichero `index.html` por la misma conexión TCP.

7. Al usar TCP, el cliente y servidor de HTTP reciben un servicio orientado a conexión, fiable, sin errores, con control de flujo, con control de congestión, etc. Es decir, una comunicación TRANSPARENTE y FIABLE.

8. Si es persistente, se siguen solicitando objetos de la página (`GET ...`) por la conexión.

9. Se cierra la conexión TCP y se liberan recursos en el servidor y cliente.

10. El cliente visualiza el contenido.

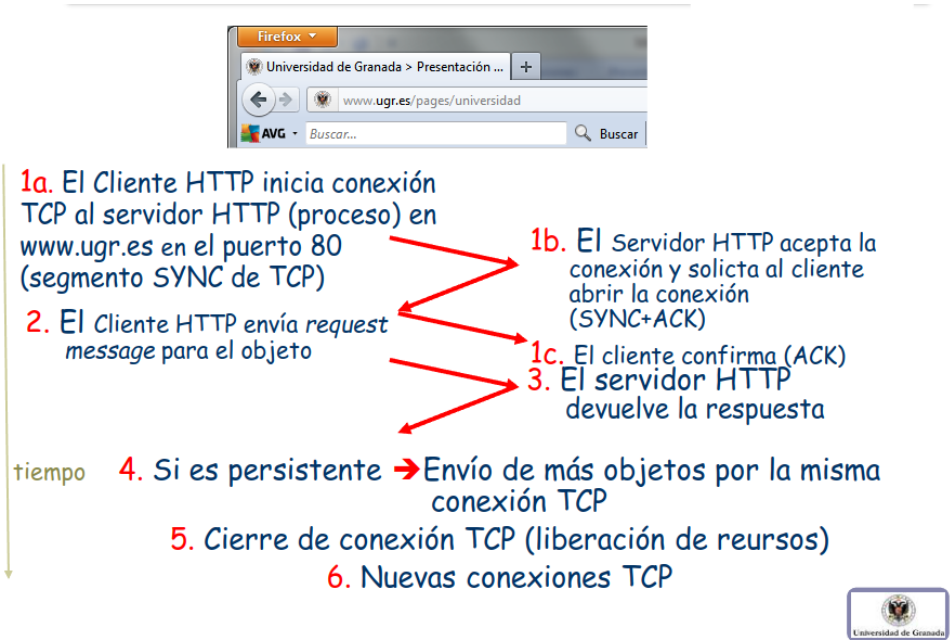

Figura 8: Ejemplo de comunicación HTTP.

HTTP define dos tipos de mensajes principales: las solicitudes del cliente al servidor (request) y las respuestas del servidor al cliente (response).

3.6.1. HTTP Request Message (Solicitud del Cliente al Servidor)

Un mensaje de solicitud HTTP enviado por el cliente al servidor tiene la siguiente estructura:

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language: fr
```

- **Línea de petición:** Esta línea contiene el método HTTP (en este caso, `GET`), el recurso solicitado (`/somedir/page.html`) y la versión del protocolo HTTP (`HTTP/1.1`).
- **Líneas de cabecera:** Estas líneas contienen información adicional sobre la solicitud, como el host (`Host: www.someschool.edu`), el agente de usuario (`User-agent: Mozilla/4.0`), el tipo de conexión (`Connection: close`) y el idioma aceptado (`Accept-language: fr`).
- **Carriage Return + Line Feed (CRLF):** Indican el fin del mensaje.

3.6.2. HTTP Response Message (Respuesta del Servidor al Cliente)

Un mensaje de respuesta HTTP enviado por el servidor al cliente tiene la siguiente estructura:

```
HTTP/1.1 200 OK
Connection: close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998
Content-Length: 6821
Content-Type: text/html
data data data data data ...
```

- **Línea de estado:** Esta línea contiene la versión del protocolo HTTP (`HTTP/1.1`), el código de estado de la respuesta (`200 OK`) que indica que la solicitud fue exitosa, y una breve descripción del estado.
- **Líneas de cabecera:** Estas líneas contienen información adicional sobre la respuesta, como el tipo de conexión (`Connection: close`), la fecha en la que se generó la respuesta (`Date: Thu, 06 Aug 1998 12:00:15 GMT`), el servidor que está procesando la solicitud (`Server: Apache/1.3.0 (Unix)`), la última vez que el recurso fue modificado (`Last-Modified: Mon, 22 Jun 1998`) y el tipo de contenido (`Content-Type: text/html`).
- **Datos:** Finalmente, los datos del mensaje pueden ser el contenido solicitado, como el fichero HTML o cualquier otro recurso.

3.7. Ejercicio: Comparación de rendimiento entre HTTP Persistente y No Persistente

Considerando los siguientes parámetros:

- **Descarga de una página web con 10 objetos incrustados**
- **Tiempo de Establecimiento de conexión TCP**: 5 ms
- **Tiempo de Cierre de conexión TCP**: 5 ms
- **Tiempo de Solicitud HTTP**: 2 ms
- **Tiempo de Respuesta HTTP**: 10 ms

3.7.1. HTTP No Persistente

En HTTP no persistente, cada objeto se solicita en una conexión TCP separada. El tiempo por objeto es el siguiente:

$$\text{Tiempo por objeto} = \text{Establecimiento de conexión} + \text{Solicitud HTTP} + \text{Respuesta HTTP} + \text{Cierre de conexión}$$

$$\text{Tiempo por objeto} = 5\,\text{ms} + 2\,\text{ms} + 10\,\text{ms} + 5\,\text{ms} = 22\,\text{ms}$$

Dado que hay 10 objetos incrustados, el tiempo total de descarga será:

$$\text{Tiempo total} = 10\,\text{objetos} \times 22\,\text{ms} = 220\,\text{ms}$$

3.7.2. HTTP Persistente

En HTTP persistente, todos los objetos se solicitan a través de una única conexión TCP. El tiempo por objeto es:

$$\text{Tiempo por objeto} = 2\,\text{ms} + 10\,\text{ms} = 12\,\text{ms}$$

El tiempo total de descarga será:

$$\text{Tiempo total} = \text{Establecimiento de conexión} + 10 \times \text{Tiempo por objeto} + \text{Cierre de conexión}$$

$$\text{Tiempo total} = 5\,\text{ms} + 10 \times 12\,\text{ms} + 5\,\text{ms} = 5\,\text{ms} + 120\,\text{ms} + 5\,\text{ms} = 130\,\text{ms}$$

3.7.3. Solución

- **HTTP No Persistente**: Tiempo total = 220 ms
- **HTTP Persistente**: Tiempo total = 130 ms

3.8. Métodos HTTP y Códigos de Respuesta

Métodos HTTP (acciones solicitadas por los clientes en los request messages):

- **OPTIONS:** Solicitud de información sobre las opciones disponibles en el servidor.
- **GET:** Solicitud de un recurso. Puede ser condicional.
- **HEAD:** Igual que GET pero el servidor no devuelve el cuerpo, solo las cabeceras.
- **POST:** Solicitud para que el servidor acepte y subordine los datos de la solicitud a la URI especificada.
- **PUT:** Solicitud para reemplazar la URI especificada con los datos incluidos en la solicitud.
- **DELETE:** Solicitud para borrar el recurso identificado por la URI.

Códigos de Respuesta (para los response messages del servidor):

- **1xx:** Mensajes informativos.
- **2xx:** Indican éxito en la operación.
- **3xx:** Redirección, el cliente debe hacer algo más para completar la solicitud.
- **4xx:** Errores del cliente, como una solicitud mal formulada.
- **5xx:** Errores del servidor, el servidor no pudo completar la solicitud.

Cabeceras comunes en HTTP:

- **Content-Type:** Describe el tipo MIME de la información contenida en el mensaje.
- **Content-Length:** Longitud en bytes de los datos enviados.
- **Content-Encoding:** Formato de codificación de los datos enviados (por ejemplo, compresión).
- **Date:** Fecha y hora de la operación.

Cabeceras solo para peticiones del cliente:

- **Accept:** Tipos MIME aceptados por el cliente.
- **Authorization:** Clave de acceso para acceder a recursos protegidos.
- **From:** Dirección de correo electrónico del usuario que realiza la petición.
- **If-Modified-Since:** Permite operaciones GET condicionales según la fecha de modificación del objeto.
- **Referer:** URL del documento que activó el enlace.
- **User-Agent:** Identifica el tipo y versión del cliente que realiza la petición.

Cabeceras solo para respuestas del servidor HTTP:

- **Allow:** Métodos HTTP opcionales que se pueden aplicar sobre el objeto.
- **Expires:** Fecha de expiración del objeto enviado, utilizada para sistemas de cache.
- **Last-Modified:** Fecha de la última modificación del objeto.

3.9. Web Cache

El objetivo de una caché web es satisfacer el requerimiento del cliente sin involucrar al servidor destino. Para ello, el usuario configura el navegador para acceder a la web vía caché:

- El navegador envía todos los requerimientos HTTP al caché.
- Si el objeto está en caché, el caché retorna el objeto directamente al cliente.
- Si el objeto no está en caché, el caché realiza una solicitud al servidor web y retorna el objeto al cliente.

Ejemplo de respuesta típica de un servidor configurado para gestionar cachés:

```
HTTP/1.1 200 OK
Date: Fri, 30 Oct 1998 13:19:41 GMT
Server: Apache/1.3.3 (Unix)
Cache-Control: max-age=3600
Expires: Fri, 30 Oct 1998 14:19:41 GMT
Last-Modified: Mon, 29 Jun 1998 02:28:12 GMT
ETag: "3e86-410-3596fbbc"
Content-Length: 1040
Content-Type: text/html
```

Las cabeceras asociadas al archivo en la caché local permiten definir la validez de la copia almacenada, evitando solicitudes innecesarias al servidor si el objeto no ha cambiado.

Objetivo de la caché: No enviar objetos si la caché tiene la versión actualizada. Para verificar esto, la caché especifica la fecha de la última copia del objeto en el requerimiento HTTP mediante las cabeceras:

- **If-Modified-Since:** Indica la fecha de la última copia de la caché.
- **If-None-Match:** Indica un identificador único para la versión del objeto (por ejemplo, un ETag).

Si el objeto no ha sido modificado desde la fecha indicada en la solicitud, el servidor responderá sin enviar el objeto, utilizando el código de estado **304 Not Modified**, como se muestra a continuación:

```
HTTP/1.0 304 Not Modified
```

En caso contrario, si el objeto ha sido modificado, el servidor responderá con el objeto actualizado:

```
HTTP/1.0 200 OK
<data>
```

Esto permite reducir la carga en el servidor y acelerar la navegación, ya que no es necesario volver a enviar objetos que ya están disponibles y actualizados en la caché.

3.10. Cookies

Las cookies son pequeños ficheros de texto que se intercambian entre los clientes y servidores HTTP para solucionar una de las principales deficiencias del protocolo: la falta de información de estado entre dos transacciones. Fueron introducidas por Netscape y estandarizadas en el RFC 2109.

- La primera vez que un usuario accede a un documento en un servidor, este proporciona una cookie que contiene datos que permitirán relacionar posteriores operaciones.
- El cliente almacena la cookie en su sistema para usarla en accesos futuros. En los siguientes accesos a este servidor, el navegador enviará automáticamente la cookie original, lo que permitirá identificar al usuario.
- Todo este proceso se realiza automáticamente, sin intervención del usuario.

Un uso inmediato de las cookies es en los sistemas de compra electrónica. Los supermercados virtuales utilizan las cookies para asociar un pedido con el cliente que lo ha solicitado. Otro uso interesante de las cookies es en los sistemas personalizados de recepción de información, donde es posible construir una página a medida con información procedente de diversas fuentes. En accesos sucesivos, el cliente enviará la cookie, y el servidor podrá generar una página personalizada con las preferencias del usuario.

Además, algunas compañías emplean las cookies para realizar un seguimiento de los accesos a sus servidores WWW, identificando las páginas más visitadas y cómo los usuarios navegan de una página a otra.

Funcionamiento de las Cookies:

Un servidor HTTP envía los diferentes campos de una cookie con la nueva cabecera HTTP `Set-Cookie`:

```
Set-Cookie: Domain=www.unican.es; Path=/;
Nombre=Luis; Expires=Fri, 15-Jul-97 12:00:00 GMT
```

Cuando se accede a una URL que verifica el par dominio/path registrado, el cliente enviará automáticamente la información de los diferentes campos de la cookie con la cabecera HTTP `Cookie`:

```
Cookie: Domain=www.unican.es; Path=/; Nombre=Luis
```

3.11. Acceso Restringido

HTTP no es un protocolo seguro, pero incluye cabeceras (`WWW-Authenticate` y `Authorization`) para restringir el acceso a recursos.

- Es vulnerable a ataques por repetición.

Las cabeceras para el acceso restringido son las siguientes:

- `WWW-Authenticate`: Se utiliza para indicar al cliente el tipo de autenticación que debe realizar. Su sintaxis es la siguiente:

```
WWW-Authenticate: <type> realm=<realm>[, charset="UTF-8"]
```

- `Authorization`: Se utiliza para enviar las credenciales de autenticación al servidor. Su sintaxis es la siguiente:

```
Authorization: <type> <credentials>
```

- Si el tipo (`type`) es `BASIC`, las credenciales incluyen el `username:password` codificado en `BASE64`.

4 Correo Electrónico

4.1. Elementos y Protocolos Principales

En el funcionamiento del correo electrónico, intervienen varios elementos y protocolos esenciales:

4.1.1. Cliente de Correo (Mail User Agent, MUA)

El **Agente de Usuario de Correo** (MUA) es la aplicación que permite al usuario componer, editar y leer los mensajes de correo electrónico. Ejemplos de MUA son aplicaciones como Outlook y Thunderbird.

4.1.2. Servidor de Correo (Mail Server o Mail Transfer Agent, MTA)

El **Servidor de Correo** (MTA) es el responsable de almacenar los correos electrónicos entrantes en los buzones de los usuarios y de reenviar los correos electrónicos salientes. Este servidor gestiona la transferencia de los mensajes entre el cliente de correo y otros servidores.

4.1.3. Protocolo de Envío: SMTP

El protocolo utilizado para el envío de correos electrónicos es el **Simple Mail Transfer Protocol** (SMTP). Este protocolo se implementa mediante dos programas, que están presentes en cada servidor de correo:

- **Cliente SMTP**: Se ejecuta en el MTA que está enviando el correo.
- **Servidor SMTP**: Se ejecuta en el MTA que está recibiendo el correo.

SMTP utiliza el puerto TCP 25 y es un protocolo orientado a texto. Es un protocolo orientado a conexión, in-band y state-full, lo que implica tres fases:

- **Handshaking** (saludo)
- **Transferencia de mensajes**
- **Cierre**

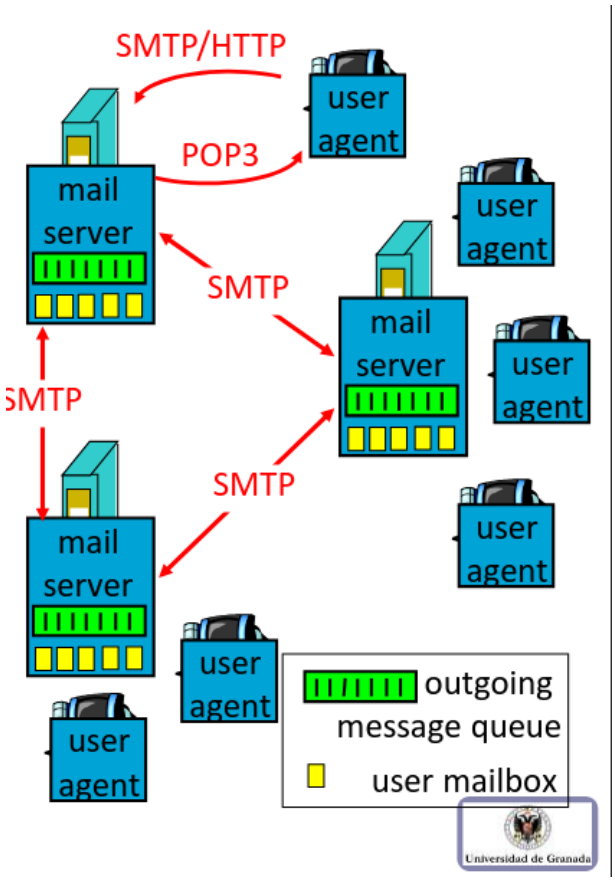

Figura 9: Proceso de envío de correo

La interacción entre el cliente SMTP y el servidor SMTP se realiza mediante comandos y respuestas, siendo los comandos texto ASCII y las respuestas códigos de estado con frases explicativas. Los mensajes deben estar codificados en ASCII de 7 bits, pero con la definición posterior de las extensiones MIME, también es posible enviar mensajes en ASCII de 8 bits y formatos enriquecidos.

4.1.4. Protocolos de Descarga (o Lectura): POP3, IMAP, HTTP

Para leer o descargar los correos electrónicos del servidor de correo, se utilizan diferentes protocolos, entre los cuales los más comunes son POP3, IMAP y HTTP.

4.1.5. Pasos en el Envío/Recepción de Correo

A continuación se describen los pasos básicos en el proceso de envío y recepción de un mensaje de correo electrónico:

1. El usuario origen compone un mensaje mediante su Agente de Usuario (MUA) dirigido al usuario destino.
2. El mensaje se envía utilizando SMTP (o HTTP) al servidor de correo (MTA) del usuario origen, donde se coloca en la cola de mensajes salientes.
3. El cliente SMTP abre una conexión TCP con el servidor de correo (MTA) del usuario destino, obtenido a través del sistema DNS.
4. El cliente SMTP envía el mensaje a través de la conexión TCP.
5. El servidor de correo del usuario destino coloca el mensaje en el buzón (mailbox) del usuario destino.
6. El usuario destino invoca su Agente de Usuario (MUA) para leer el mensaje, utilizando POP3, IMAP o HTTP[4].

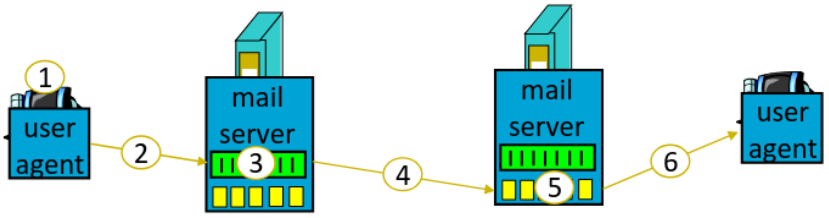

Figura 10: Pasos en el envío y recepción de correo

4.2. Multipurpose Internet Mail Extensions (MIME)

Las extensiones MIME (Multipurpose Internet Mail Extensions) no cambian la arquitectura de correo electrónico, pero permiten soportar diversas funcionalidades adicionales. Los objetivos de MIME son:

- Soportar texto en conjuntos de caracteres distintos de US-ASCII.
- Permitir adjuntos que no sean de tipo texto.
- Manejar cuerpos de mensajes con múltiples partes (multi-part).
- Incluir información de encabezados con conjuntos de caracteres distintos de ASCII.

MIME está especificado en seis RFCs: RFC 2045, RFC 2046, RFC 2047, RFC 4288, RFC 4289 y RFC 2077.

[4]Para ver los códigos de respuesta y de petición, accede a las diapositivas 78 y 79 del tema 5

4.2.1. No confundir los mensajes del protocolo con el formato de almacenamiento

Es importante destacar que los mensajes MIME no deben confundirse con el formato de almacenamiento. Los mensajes MIME definen cómo los datos son representados para su transmisión a través de la red, mientras que el formato de almacenamiento puede diferir.

4.2.2. Cabeceras de mensajes MIME

Las cabeceras MIME permiten definir y gestionar diferentes aspectos de los mensajes. Algunas de las cabeceras más importantes son:

- **MIME-Version**: Identifica la versión de MIME. Si no existe, se considera que el mensaje es texto normal en inglés.
- **Content-Description**: Cadena de texto que describe el contenido del mensaje, necesaria para que el destinatario sepa si desea descodificar y leer el mensaje.
- **Content-Id**: Identificador único, utiliza el mismo formato que la cabecera estándar `Message-Id`.
- **Content-Transfer-Encoding**: Indica la manera en que el cuerpo del mensaje está envuelto para su transmisión.
- **Content-Type**: Especifica la naturaleza del cuerpo del mensaje.

4.2.3. Content-Transfer-Encoding

Este campo indica cómo está envuelto el cuerpo del mensaje para su transmisión, ya que caracteres distintos de letras, números y signos de puntuación pueden generar problemas. Existen 5 tipos de codificación, especificados en la RFC 1521:

- ASCII 7
- ASCII 8
- Codificación binaria
- Base64
- Codificación entrecomillada-imprimible

4.2.4. Content-Type: Tipos y Subtipos

El campo `Content-Type` define el tipo y subtipo del contenido del mensaje. La lista inicial de tipos y subtipos especificada por el RFC 1521 es la siguiente:

- **Text/Plain**: Texto sin formato.
- **Text/Richtext**: Texto con comandos de formato sencillos.
- **Image/Gif**: Imagen fija en formato GIF.

- **Image/Jpeg**: Imagen fija en formato JPEG.
- **Audio/Basic**: Sonido.
- **Video/Mpeg**: Película en formato MPEG.
- **Application/Octet-stream**: Secuencia de bytes no interpretada.
- **Postscript**: Documento imprimible en formato PostScript.
- **Message/Rfc822**: Mensaje MIME en formato RFC 822.
- **Multipart/Mixed**: Partes independientes en el orden especificado.
- **Multipart/Alternative**: El mismo mensaje en diferentes formatos.
- **Multipart/Parallel**: Las partes deben verse simultáneamente.
- **Multipart/Digest**: Cada parte es un mensaje RFC 822 completo.

4.2.5. Content-Type: Tipo `application`

El tipo `application` es un tipo general para los formatos que requieren procesamiento externo no cubierto por otros tipos. El subtipo `octet-stream` es una secuencia de bytes no interpretada, y su recepción sugiere al usuario que lo copie en un archivo.

4.2.6. Content-Type: Tipo `message`

El tipo `message` permite que un mensaje esté completamente encapsulado dentro de otro. Los subtipos más comunes son:

- **rfc822**: Se utiliza cuando un mensaje RFC 822 completo está encapsulado en un mensaje exterior.
- **partial**: Divide un mensaje en partes y las envía por separado. Los parámetros permiten ensamblar correctamente todas las partes.
- **external-body**: Usado para mensajes grandes (por ejemplo, archivos de vídeo), donde se proporciona una dirección FTP en lugar de incluir el archivo en el mensaje.

4.2.7. Content-Type: Tipo `multipart`

El tipo `multipart` permite que un mensaje contenga más de una parte. Los subtipos más comunes son:

- **mixed**: Permite que cada parte sea diferente.
- **alternative**: Contiene el mismo mensaje en diferentes formatos.
- **parallel**: Las partes deben ser vistas simultáneamente, como en el caso de los canales de audio y vídeo de las películas.

- **digest**: Agrupa varios mensajes en un único mensaje compuesto[5].

4.3. Protocolo POP3 e IMAP4

4.3.1. Protocolo POP3

El **Post Office Protocol 3** (POP3) es un protocolo utilizado para la descarga de correos electrónicos desde un servidor a un cliente. El puerto TCP utilizado por POP3 es el `110`. Este protocolo se organiza en tres fases:

- **Fase de Autorización**: En esta fase el cliente debe autenticarse en el servidor utilizando los comandos `USER` y `PASS`.
- **Fase de Transacción**: Durante esta fase el cliente puede interactuar con el servidor para listar mensajes (`LIST`), obtener mensajes específicos (`RETR`) o eliminar mensajes (`DELE`).
- **Fase de Actualización del Servidor**: Al finalizar la sesión, el servidor elimina los mensajes marcados para su eliminación y cierra la conexión.

4.3.2. Comandos POP3

- **USER**: Permite la identificación del usuario, seguido del nombre de usuario que identifica al cliente en el servidor.
- **PASS**: Especifica la contraseña para el usuario previamente indicado con el comando `USER`.
- **STAT**: Proporciona información acerca de los mensajes en el servidor.
- **RETR**: Solicita la descarga de un mensaje especificado por su número.
- **DELE**: Elimina el mensaje especificado por su número.
- **LIST**: Lista los mensajes por su número.
- **NOOP**: Mantiene la conexión abierta en caso de inactividad.
- **TOP**: Muestra las primeras `n` líneas del mensaje, proporcionando los encabezados y una parte del cuerpo.
- **UIDL**: Solicita al servidor que envíe un identificador único para el mensaje especificado.
- **QUIT**: Finaliza la sesión, eliminando los mensajes marcados como eliminados y enviando el estado de la acción.

[5] Ejemplo en la diapositiva 87 del tema 5

4.3.3. Protocolo IMAP4

El **Internet Message Access Protocol** versión 4 (IMAP4)[6] permite trabajar con el correo electrónico de manera similar a si fuera local, manteniendo la organización en el servidor (MTA). Los estados de IMAP4 incluyen:

- **No Autenticado (NA)**: El cliente aún no se ha autenticado.
- **Autenticado (A)**: El cliente ha iniciado sesión con éxito.
- **Seleccionado (S)**: El cliente ha seleccionado una carpeta o un mensaje específico.
- **Desconexión (D)**: El cliente ha cerrado la sesión.

4.3.4. Comandos IMAP4

IMAP4 utiliza una serie de comandos según el estado del cliente:

- **CAPABILITY**: Obtiene la lista de capacidades del servidor IMAP.
- **LOGIN / AUTHENTICATE**: Permite la autenticación del usuario.
- **SELECT**: Selecciona una carpeta para acceder a los mensajes.
- **CREATE / DELETE**: Crea o elimina una carpeta.
- **LIST**: Lista las carpetas disponibles en el servidor.
- **APPEND**: Añade un mensaje a una carpeta.
- **FETCH**: Recupera un mensaje.
- **STORE**: Modifica los flags o atributos de un mensaje.
- **SEARCH**: Permite buscar mensajes con ciertos criterios.

4.3.5. Ventajas de IMAP4

IMAP4 ofrece varias ventajas sobre otros protocolos, como POP3:

- **Organización en carpetas en el servidor**: IMAP permite que los correos se gestionen y organicen directamente en el servidor, lo que facilita su acceso desde distintos dispositivos.
- **Mantiene información entre sesiones**: Los flags y la organización de los mensajes se mantienen entre sesiones, lo que permite acceder a la misma estructura desde cualquier cliente.
- **Descarga de partes del mensaje**: IMAP permite descargar solo una parte del mensaje (como los encabezados) para optimizar el ancho de banda.
- **Acceso desde varios clientes**: IMAP permite acceder al mismo correo desde diferentes dispositivos, sin necesidad de descargarlo primero, a diferencia de POP3.

[6] Más explicaciones y ejemplos de IMAP4 en las diapositivas 91, 92, 93 del tema 5.

4.3.6. Ventajas de Webmail

El acceso al correo a través de Webmail, utilizando un navegador web y protocolos como HTTP o HTTPS, tiene sus propias ventajas:

- **Acceso desde cualquier dispositivo**: Webmail permite acceder al correo desde cualquier lugar con conexión a Internet.
- **Organización en el servidor**: La organización de los correos se mantiene en el servidor, accesible desde cualquier cliente con HTTP.
- **Seguridad**: El uso de HTTPS mejora la seguridad, cifrando las comunicaciones entre el cliente y el servidor.

4.4. Listado de Puertos Relacionados con el Correo Electrónico

Los protocolos utilizados para el envío y recepción de correos electrónicos operan en puertos específicos. A continuación se presenta un listado de los puertos más comunes asociados a estos protocolos:

- **POP3**: Puerto `110` - Protocolo utilizado para la descarga de correos electrónicos desde el servidor.
- **IMAP**: Puerto `143` - Protocolo utilizado para acceder y gestionar correos electrónicos en el servidor.
- **SMTP**: Puerto `25` - Protocolo utilizado para el envío de correos electrónicos.
- **HTTP**: Puerto `80` - Protocolo utilizado para la navegación web, también utilizado en el acceso a Webmail.
- **Secure SMTP (SSMTP)**: Puerto `465` - Versión segura de SMTP utilizando SSL/TLS para cifrar las comunicaciones.
- **Secure IMAP (IMAP4-SSL)**: Puerto `585` - Versión segura de IMAP utilizando SSL/TLS.
- **IMAP4 over SSL (IMAPS)**: Puerto `993` - IMAP sobre SSL, utilizado para una comunicación segura entre cliente y servidor.
- **Secure POP3 (SSL-POP)**: Puerto `995` - Versión segura de POP3 utilizando SSL/TLS para cifrar las comunicaciones.

5 Aplicaciones Multimedia

5.1. Conceptos de Calidad de Servicio (QoS) y Aplicaciones Multimedia

5.1.1. Tecnología de Convergencia

La **IP** (Internet Protocol) se considera una "tecnología de convergencia"porque permite integrar diferentes tipos de servicios de comunicación, como voz, datos, y vídeo,

sobre una misma infraestructura. Esto facilita la integración de redes y servicios, ofreciendo una solución eficiente y flexible para la transmisión de información.

5.1.2. Calidad de Servicio (QoS)

La **Calidad de Servicio** (QoS, por sus siglas en inglés) se refiere a la capacidad de una red para ofrecer el rendimiento requerido por una aplicación, garantizando ciertos parámetros de rendimiento, como el ancho de banda, la latencia, el jitter, y la pérdida de paquetes.

En las redes IP tradicionales, se ofrece un servicio de **mejor esfuerzo** (*Best Effort*), lo que significa que no se garantizan niveles específicos de QoS, ya que todos los datos compiten por los mismos recursos de red sin priorización. Sin embargo, para aplicaciones multimedia, que requieren una calidad constante y predecible, es fundamental implementar técnicas de QoS para garantizar un rendimiento adecuado.

5.1.3. Aplicaciones Multimedia

Las aplicaciones multimedia, como audio, vídeo, juegos y comunicaciones en tiempo real, tienen necesidades de rendimiento específicas. A continuación, se describen algunos tipos de aplicaciones multimedia y sus características:

- **Flujo de audio y vídeo (streaming) almacenado**: Este tipo de flujo se refiere a la transmisión de contenido previamente grabado, como en plataformas de *streaming* como *YouTube*.
- **Flujo de audio y vídeo en vivo**: En este caso, el contenido se transmite en tiempo real, como en emisoras de radio o servicios de IPTV (Televisión por protocolo de Internet).
- **Audio y vídeo interactivo**: Las aplicaciones que permiten una comunicación interactiva, como *Skype* para videollamadas, requieren un alto rendimiento en términos de latencia y jitter.

5.1.4. Características Fundamentales de las Aplicaciones Multimedia

Las aplicaciones multimedia tienen varias características fundamentales que requieren un manejo especial por parte de la red:

- **Elevado ancho de banda**: Las aplicaciones multimedia, especialmente el vídeo de alta calidad, requieren un ancho de banda significativo para garantizar una transmisión fluida y sin interrupciones.
- **Tolerancia relativamente alta a la pérdida de datos**: Estas aplicaciones pueden tolerar pequeñas pérdidas de datos sin que la calidad se vea gravemente afectada, pero no pueden soportar pérdidas significativas.
- **Exigen delay (retardo) acotado**: La latencia debe ser baja, ya que los retrasos en la transmisión de audio o vídeo pueden causar problemas de sincronización y afectar la experiencia del usuario.

- **Exigen jitter (fluctuación del retardo) acotado**: Las fluctuaciones en la latencia (jitter) deben mantenerse dentro de límites aceptables para evitar problemas en la calidad de la experiencia multimedia.
- **Beneficio del multicast**: Las aplicaciones multimedia pueden beneficiarse del uso de multicast, donde se envía el contenido a múltiples destinos a la vez, optimizando el uso de la red y mejorando la eficiencia.

5.1.5. ¿Cómo es un Router con QoS?

Un **router con QoS** es un dispositivo de red capaz de gestionar el tráfico de forma que las aplicaciones con requisitos de rendimiento específicos reciban un tratamiento preferencial. Los routers con QoS implementan diversas técnicas de control, como la priorización de paquetes, la asignación de ancho de banda y la gestión de la congestión, para asegurar que los flujos de datos más importantes, como los de las aplicaciones multimedia, no sufran interrupciones o deterioro de calidad.

Algunas de las funcionalidades que un router con QoS puede implementar incluyen:

- **Priorización de tráfico**: Los paquetes de datos de aplicaciones multimedia pueden ser priorizados sobre otros tipos de tráfico menos sensibles a la latencia o pérdida de paquetes, como el correo electrónico o la navegación web.
- **Asignación de ancho de banda**: El router puede reservar un cierto ancho de banda para las aplicaciones que lo necesiten, asegurando que el rendimiento no se vea afectado durante picos de tráfico.
- **Control de la congestión**: El router puede usar algoritmos de gestión de tráfico, como el control de la cola, para evitar la sobrecarga de la red y asegurar una distribución equitativa de los recursos.
- **Garantía de calidad**: Para aplicaciones que requieren un rendimiento garantizado, el router puede establecer mecanismos para asegurar que se mantengan los niveles de QoS especificados.

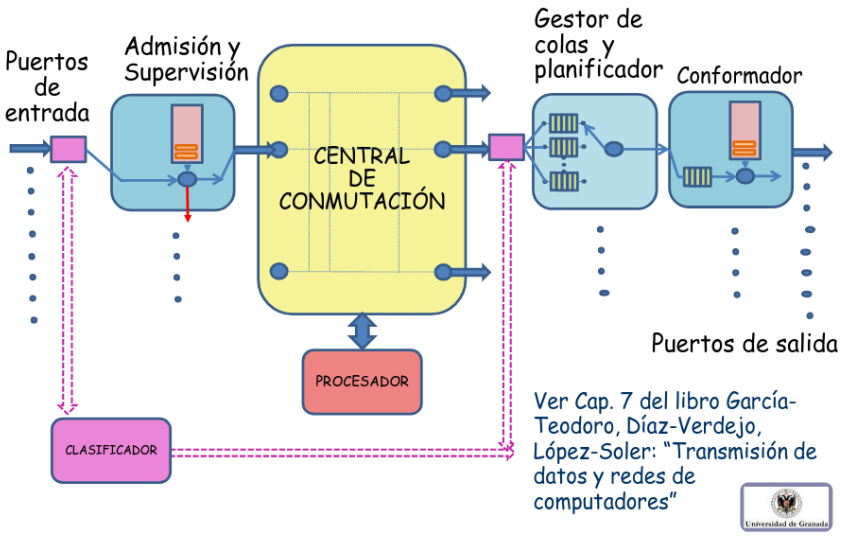

Figura 11: Router con QoS