

Tema 5: Servicios Multimedia

Ingeniería de Servicios

2023-2024

T5.1: Introducción

Introducción

Dos tipos de servicio multimedia:

- No interactivos.
 - El usuario "consume" el servicio sin interactuar con él (ej. reproducción de vídeo o audio)
 - Desde el punto de vista de la ingeniería de servicios, es un problema de *transmisión archivos*.
 - Problemas nuevos: *streaming*, sincronización, formatos, codecs, ...
- Interactivos.
 - El usuario interactúa con otro usuario, mediante contenido multimedia, generalmente en tiempo real
 - Ejemplo: Voz sobre IP, videoconferencia (Skype, Google Hangouts, ...)
 - Son necesarios nuevos protocolos (presencia, inicio de sesión, transporte,...)

Aspectos a estudiar

- Representación de la información multimedia
 - Imagen estática
 - Audio
 - Vídeo
- Compresión (códecs)
 - La guerra de los códecs
 - Estándares
- Contenedores
- Protocolos de transmisión

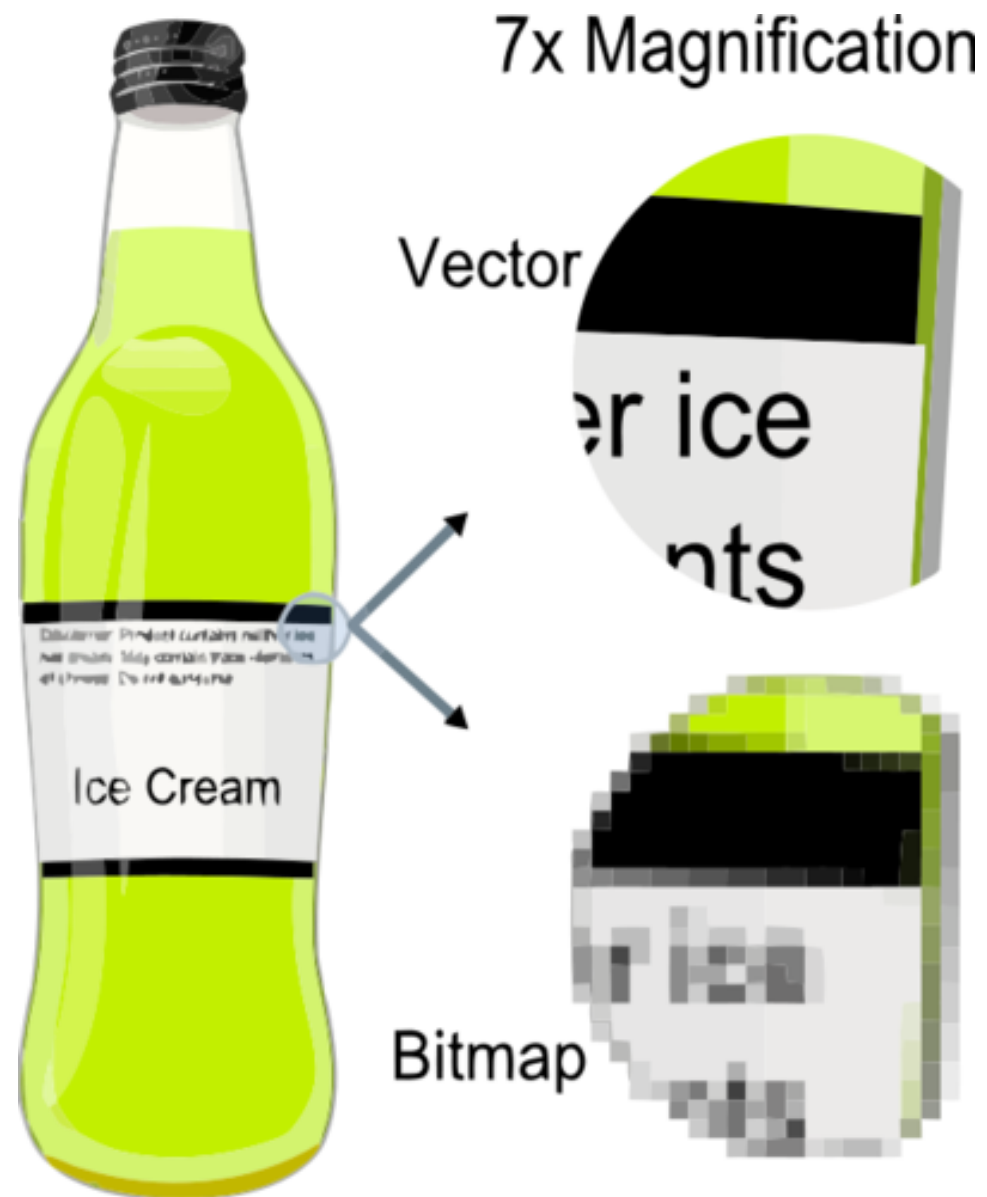
La información multimedia

Imagen

Dos formas de representación:

- **Mapa de bits** (*bitmap* o *raster*):
 - La imagen es un array de puntos llamados *pixels*
 - Cada pixel almacena un color usando n bits
 - El color se expresa por componentes RGB
 - Ej: BMP, PNG, JPG, GIF, TIFF, WEBP
- **Vectorial**
 - La imagen es una lista de objetos
 - Cada objeto es una primitiva: círculo, recta, arco
 - Con unos atributos (tamaño, origen, color de relleno, color de línea, grosor)
 - Es independiente de la resolución del dispositivo en que se muestre
 - Ej: SVG y otros no estándar (WMF, PDF, EPS, AI)

Ejemplo



Mapas de bits

Una imagen es un array de *pixels*, en su tamaño influye:

- Cuántos pixels hay (según la resolución)
- Cuántos bits ocupa cada pixel ("profundidad de color")

Nombre	Resolución	Pixeles	8bit	16bit	24bit
QVGA	320×200	64000	64kB	128kB	192kB
VGA	640×480	307200	307kB	614kB	921kB
XGA	1024×768	786432	787kB	1,58MB	2,36MB
HD*	1280×720	921600	922kB	1,85MB	2,77MB
SXGA	1280×1024	1310720	1,32MB	2,62MB	3,94MB
FullHD**	1920×1080	2073600	2,08MB	4,15MB	6,22MB
QXGA	2048×1536	3145728	3,15MB	6,30MB	9,44MB
UltraHD*	3840×2160	8294400	8,29MB	16,59MB	24,88MB

*HD=720p, **FullHD=1080p, ***UHD≈4K

Compresión de los mapas de bits

Las imágenes suelen contener *redundancia espacial* (zonas con un mismo valor de pixel repetido), y esto se puede aprovechar para comprimirlas.

- Compresión **sin pérdida**. A partir de la imagen comprimida es posible recuperar la imagen original.
 - Algoritmos de compresión estándar válidos para comprimir cualquier secuencia de bytes, sean o no una imagen.
 - Para capturas de pantalla, documentos escaneados, etc.
 - Ejemplo: PNG, GIF, BMP, TIFF.
 - PNG admite transparencia (*canal alpha*)



Compresión de los mapas de bits (2)

- Compresión **con pérdida**. A partir de la imagen comprimida no es posible recuperar la original, sino una *aproximación*.
 - Se basan en deficiencias en la percepción humana, de forma que la *aproximación* sea casi indistinguible del original.
 - Dependiendo de la *calidad* deseada, la aproximación puede alejarse más o menos del original (y así lograr mayores o menores ratios de compresión)
 - Apropiaada para fotografías.
 - Ejemplo: **JPEG**

Compresión de los mapas de bits (y 3)

Ejemplo 1

Calidad: Máxima



Compresión: 2,6 : 1

Ejemplo 2

Calidad: Media



Compresión: 2,6 : 1

Ejemplo 3

Calidad: Baja



Compresión: 46 : 1

Video

- Un vídeo es una secuencia de *frames*
- Cada *frame* es un mapa de bits
- El tamaño del vídeo por tanto depende de:
 - El tamaño de cada *frame*
 - El número de *frames* por segundo (*frame rate*)
 - La duración del vídeo

Importante el concepto de *bitrate* = bits por segundo. Depende de:

- El tamaño de cada *frame*
- El *frame rate*

Ejemplo

Una película (muda) en Blu-Ray de 90 minutos

- Resolución: Full-HD (1920×1080), 24 bits por pixel
- Frame rate: 24 fps (*frames per second*)
- Duración: 90 minutos (5400 segundos)



Tamaño total: $1920 \times 1080 \times 3 \times 24 \times 5400 = 806\text{GB}$ (750GiB)
Bitrate: $1920 \times 1080 \times 24 \times 24 = 1194 \text{ Mbit/s}$

Obviamente ¡hay que comprimir!

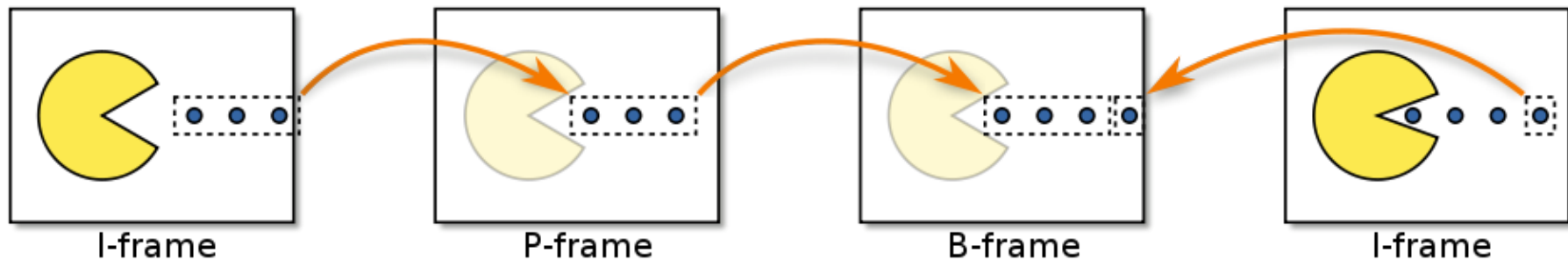
Compresión del vídeo

- Compresión *intra-frame*
 - Cada *frame* se comprime por separado
 - Usando por ejemplo JPEG (este formato se llama MJPEG)
 - Explota la redundancia espacial en cada frame
- Compresión *inter-frame*
 - Existe también redundancia *temporal*
 - Es decir, cada *frame* se parece mucho al anterior y al siguiente
 - Se pueden codificar sólo las diferencias entre frames

Compresión del vídeo

Tres tipos de *frame*:

- I-frame (**I**ntra coded). Usa el mismo algoritmo de compresión que una imagen fija. No necesita de otros *frames* para decodificarse.
- P-frame (**P**redicted). Codifica sólo los cambios respecto al *frame* anterior.
- B-frame (**B**i-predictive). Codifica los cambios con respecto al *frame* anterior y el siguiente.



Algoritmos de compresión

También llamados *codecs* (encoder/decoder)

■ Existen muchos, la mayoría patentados.

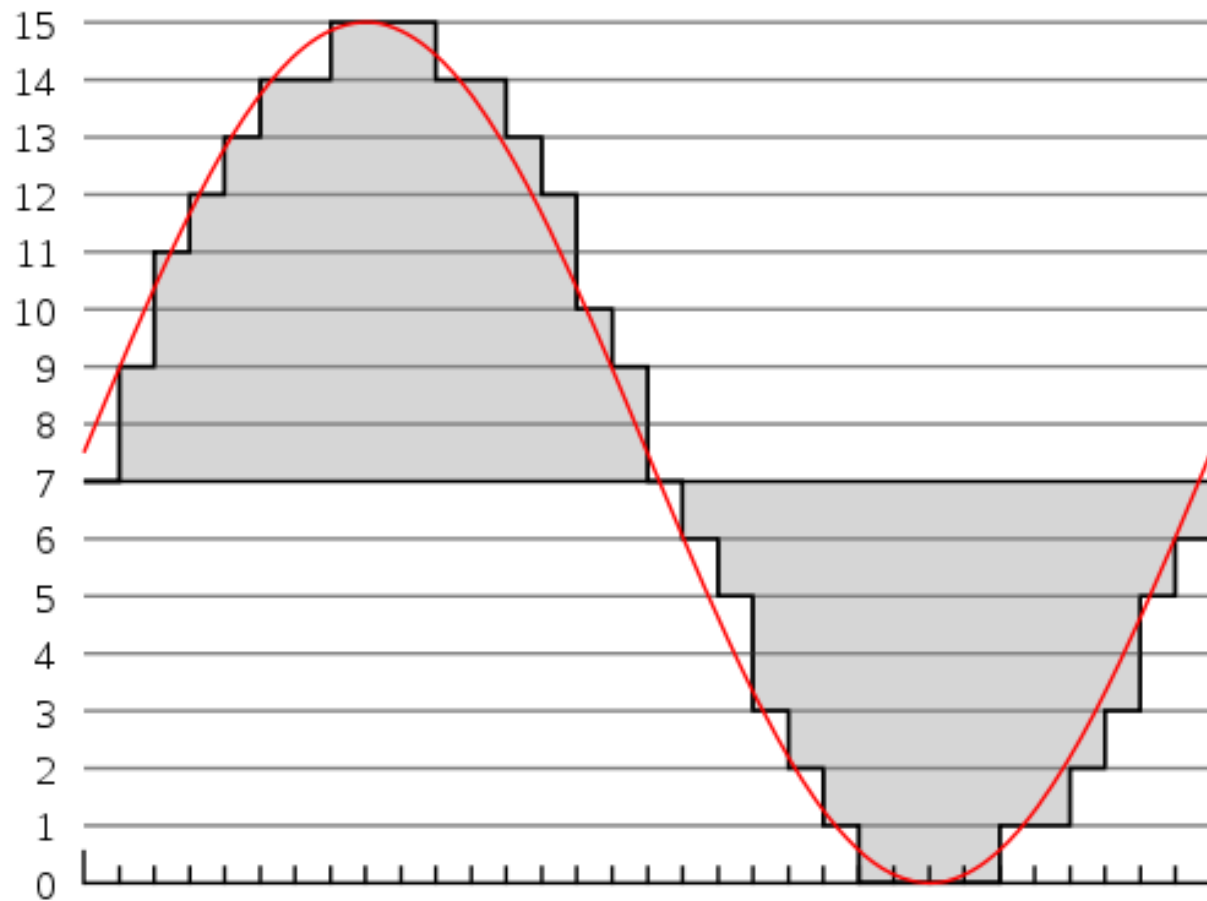
- **MPEG-1 Part 2** (~1993). Patente ya expirada. Baja calidad (VHS)
- **MPEG-2 Part 2** (~1996). Depende de muchas patentes, algunas ya expiradas. Usado en los DVD y Blu-ray (puede comprimir el ejemplo anterior a 80Mbit/s)
- **MPEG-4 Part 2** (~1999). Estandarizado como ISO 14496-2. Muy popular por incorporarse en codecs como DivX, Xvid o Nero.
- **Theora** (~2003). Open source. Basado en el formato VP3 (2001). Implementado en muchos reproductores open source.
- **VP6** (~2003). Desarrollado por On2, usado por Adobe Flash en su formato de vídeo (>2005) y por JavaFX (>2008). Formato propietario.

Algoritmos de compresión (sigue)

- **MPEG-4 Part 10** (~2003). También llamado **AVC**. Estandarizado como H.264 por ITU. Cubierto por muchas patentes. Soportado por Adobe Flash (>2007), Apple iOS, y otros. Decodificadores en *hardware* de alto rendimiento.
- **VP8** (~2008). Desarrollado por On2, comprado por Google (2010) y publicado gratuitamente bajo licencia BSD. Usado en el formato de vídeo WebM (promovido por Google).
- **HEVC** (~2013). Sucesor de MPEG-4, estandarizado como H.265 por ITU. Puede comprimir el ejemplo anterior a 30MBit/s.
- **VP9** (~2013). Sucesor de VP8, para mejorar su compresión. Propiedad de Google, pero de código abierto y libre de cargas. Desde 2014 soportado por Chrome, Firefox y Opera.
- **AV1** (~2017). Incorpora VP10. Estado del arte royalty-free. Apenas empieza a ser soportado, pero será el futuro (Google, Mozilla, Microsoft, Netflix, están detrás).

Audio

- Una vez captado por un micrófono, el sonido es una señal eléctrica.
- Mediante un proceso de *muestreo y cuantización* se convierte a una secuencia de números.



Características

- Periodo de muestreo:
 - Ha de ser el doble de la frecuencia más alta que se quiera retener.
 - El oído humano capta hasta unos 20KHz
 - La típica frecuencia de muestreo es 44.1KHz
- Bits de resolución:
 - Cuantos más bits, mayor fidelidad
 - Pero mayor tamaño de la información digital.
 - Valor típico 16 bits

44.1KHz y 16 bits ⇒ Calidad Compact-Disc

Bitrate y tamaño

Al igual que en vídeo, existe el concepto de *bitrate* = bits por segundo. Depende de:

- Frecuencia de muestreo
- Bits en cada muestra.
- Número de canales (mono/estéreo)



En un Compact-Disc será: $44100 \times 16 \times 2 = 1411,2 \text{ Kbit/s}$

Un Compact-Disc dura 74 minutos ¿cuánto ocupa el audio digitalizado?

Compresión del audio

Al igual que en vídeo, existe una gran diversidad de *codecs* para codificar y decodificar el audio:

- Sin compresión
 - **PCM** (1982) Usado por formatos WAV, AIFF y Compact-Disc
- Con compresión sin pérdida
 - **FLAC** (2001), código abierto, usado por Xiph.org
 - **Monkey's Audio** (APE). Gratuito pero protegido por licencias
 - **Windows Media Audio 9 Lossless**. Propietario.
 - **Apple Lossless** (2004) También llamado ALAC, soportado por QuickTime, propietario. Código abierto en 2011 con Licencia Apache
- Con compresión con pérdida (Transparencia siguiente)

Codecs de audio con pérdida

- **MPEG-1 Audio Layer III** (1993, 1995) Más conocido por **MP3**. Protegido por patentes y licencias.
- **MPEG-2 Part 7** (1997) Más conocido por *Advanced Audio Coding* o **AAC**, soporta muchos más de 2 canales. Hi-Fi Stereo a 128Kbit/s. No hay cargas por usar o distribuir audio en AAC, pero sí para fabricar o programar codecs.
- **Windows Media Audio** o WMA (>1999)
- **Vorbis** (2002) Libre de costes y de patentes. Calidad similar a AAC.
- **MPEG-4 Part 3** (2003) Conocido también como HE-AAC
- **Opus** (2013-actualidad) Sucesor de Vorbis.

Contenedores digitales

Un contenedor es un formato de archivo que contiene uno o varios *streams* de audio o vídeo y otros datos, junto con información sobre cómo sincronizarlos.

Por ejemplo, un contenedor puede contener:

- Un stream de vídeo (sin sonido)
- Un stream de audio con la versión original del mismo
- Otro stream de audio con el doblaje a otro idioma
- Un stream de subtítulos

Cada *stream* de audio o vídeo requerirá el *codec* apropiado para su decodificación. El contenedor es independiente de los codec.

Contenedores habituales

- **Audio Video Interleave** (extensión `avi`). Puede usar cualquier codec soportado por Video For Windows.
- **Flash Video** (extensión `flv`). Suele usar VP6 o H.264 para el vídeo, MP3 o AAC para el audio.
- **Matroska** (extensión `mkv`). Genérico, cualquier tipo de contenido. Abierto.
- **MP4** (extensión `mp4`). Suele contener MPEG-2 part 2 o H.264 para el vídeo y MP3 o (HE-)AAC para el audio. Protegido por patentes. Puede contener solo audio (a veces se usa `m4a` para ese caso)
- **Ogg** (extensión `ogg`). Suele contener Theora para el video y Vorbis para el audio. Abierto. Puede contener solo audio.
- **QuickTime** (extensión `mov`). Puede usar cualquier codec soportado por QuickTime. Propietario de Apple.
- **WebM** (extensión `webm`). Basado en Matroska. Suele contener VP8 para el vídeo y Vorbis para el audio (o AV1 y Opus). Propiedad de Google, pero con licencia gratis perpetua.

Protocolos

Protocolos de transmisión

- Si tratamos el contenedor digital como cualquier otro archivo:
 - Se puede usar un protocolo de transferencia de archivos (FTP, sftp, HTTP)
 - Se descarga el archivo. Después se reproduce
 - Se requiere un reproductor que entienda el formato contenedor
 - Se requieren los *codecs* particulares usados para codificar audio y vídeo
 - No se puede iniciar la reproducción hasta tener el archivo completo
 - El usuario puede pausar, rebobinar, avanzar, etc. pues estas operaciones son locales sobre el archivo descargado.



HTTP es un caso un poco particular, como veremos

Protocolos de *streaming*

- Protocolos específicos para transmitir audio/video
 - Transmiten los *frames* (comprimidos con algún codec)
 - Y los canales de audio
 - Incluyen datos para la sincronización de los diferentes *streams*.
 - Controlan la calidad de la conexión y se ajustan a ella
 - La reproducción puede hacerse simultánea a la descarga
 - La fuente de audio o video no necesita ser un fichero en el servidor. Puede ser contenido generado en tiempo real.
 - El cliente no necesita almacenar el fichero.
 - Se usan otros protocolos para controlar la reproducción (play, pause, seek), a modo de "control remoto"

El caso particular de HTTP

- HTML originalmente no tenía previsto el contenido multimedia.
- Por tanto un audio/vídeo no podía reproducirse en el navegador.
 - Aunque podía descargarse por HTTP y abrirse un reproductor externo

Enseguida se cubrió esa carencia:

- A través del tag `<object>` se pueden "incrustar" otros procesos (plugins) en la página mostrada por el navegador.
- Esto permitía mostrar en la página audio/video
- Pero el proceso que lo reproducía era de "terceras partes"
 - RealPlayer
 - Windows Media Player
 - Macromedia Flash (hoy Adobe Flash)
 - Java (applet)

El caso particular de HTTP (sigue)

- El protocolo usado por ese *plugin* era específico del mismo
 - En muchos casos era un protocolo propietario
 - Distinto de HTTP (ej: Flash usa RTMP)
- El formato de vídeo/audio soportado era específico del *plugin*
 - En muchos casos era un formato propietario
 - Aunque con el tiempo todos empezaron a soportar estándares como H.264
- El aspecto del reproductor era controlado por el plugin
 - Y no por el HTML o el CSS de la página
 - Puesto que no era parte de la misma

HTML5






- El nuevo estándar HTML5 prevé los tags `<audio>` y `<video>`
- Los navegadores que lo soportan, son capaces de reproducir ellos mismos el audio y video sin necesidad de *plugins*.
- Esto elimina la necesidad de Flash, RealPlayer, etc.
- Apple ha sido un gran impulsor de HTML5 y de la eliminación del Flash



¿Qué formatos y codecs soporta HTML5?

■ NO LO ESPECIFICA. Depende de cada navegador

Soporte de multimedia en los navegadores

Navegador	Audio			Video				
	Vorb	MP3	AAC	Theo	H.264	VP8	VP9 (webm)	AV1
Chrome >29	Si	Si	Si	Si	Si	Si	Si	>70
IE >10	--	Si	Si	--	Si	--	--	--
Edge >16	--	Si	Si	--			 	>18 
Safari >4	--	Si	Si	--	Si	--	--	--
Firefox >34	Si	★	★	Si	Si	Si	Si	>67
Opera >24	Si	★	★	Si	Si	Si	Si	>57
iOS >8	--	Si	Si	--	Si	--	--	--
Android >4.4	Si	Si	Si	Si	Si	Si	Si	>10

 A través de Media Source Extensions

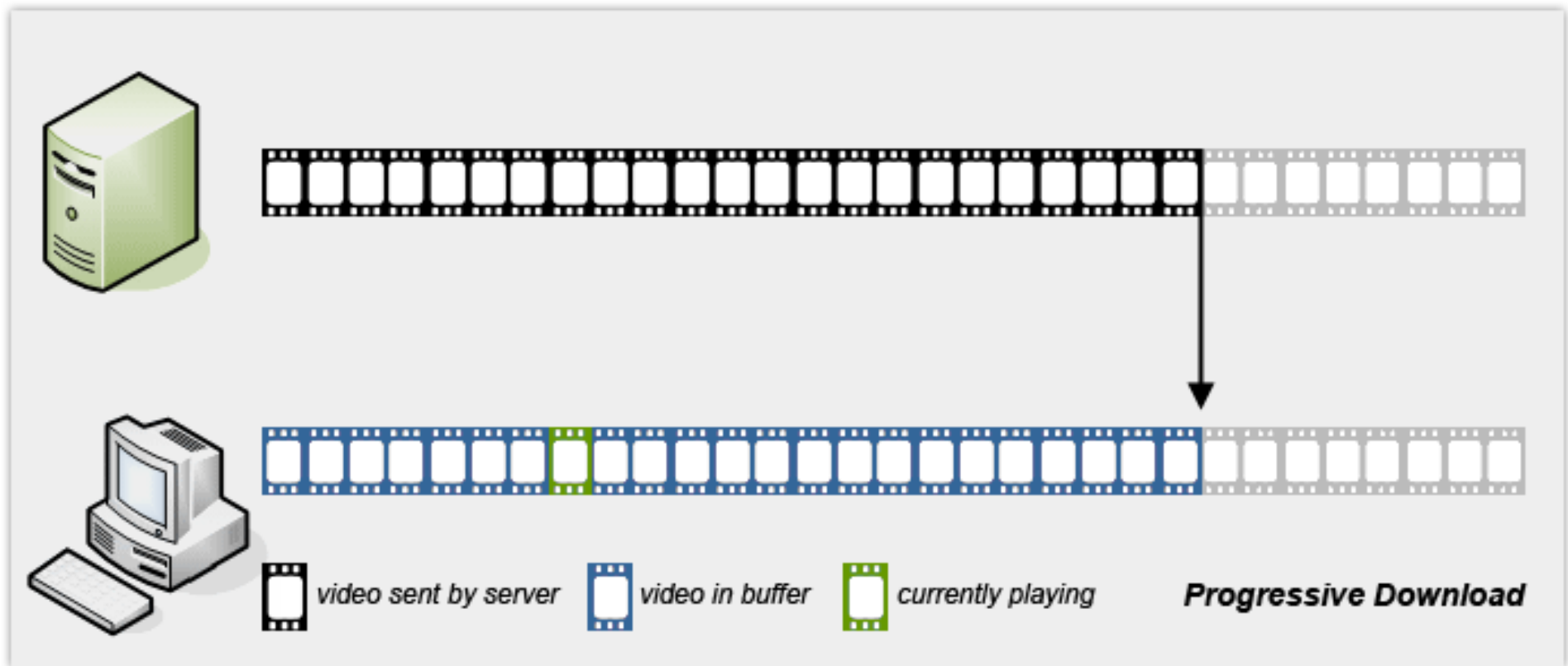
 Necesita decodificador hardware

★ Si lo soporta el operativo

Descarga progresiva

- Usa HTTP para descargar el archivo (un servidor HTTP estándar puede servirlo)
- El cliente comienza la reproducción tan pronto como es posible (sin esperar fichero completo)
 - Requiere un formato de contenedor con información apropiada en la cabecera
 - Requiere velocidad de descarga > velocidad de reproducción (si no, habrá "parones")
- El cliente puede usar la cabecera **Range:** para especificar el byte en que comenzar la descarga
 - Esto permite "avanzar" rápidamente a diferentes puntos del vídeo.
 - El servidor debe indicar **Accept-Ranges:** en su cabecera
- Es usado por todos los navegadores para el tag **<video>** de HTML de HTML5

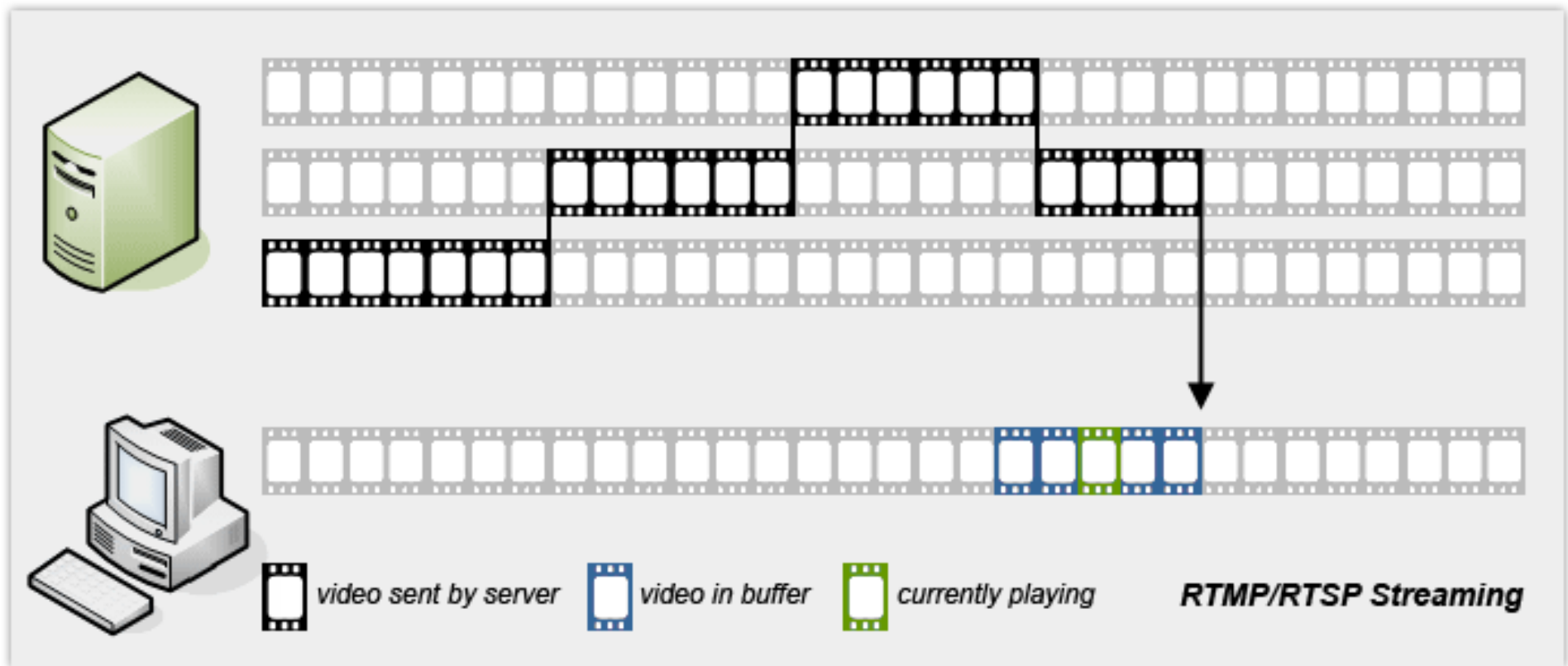
Descarga progresiva: HTTP



Inconvenientes:

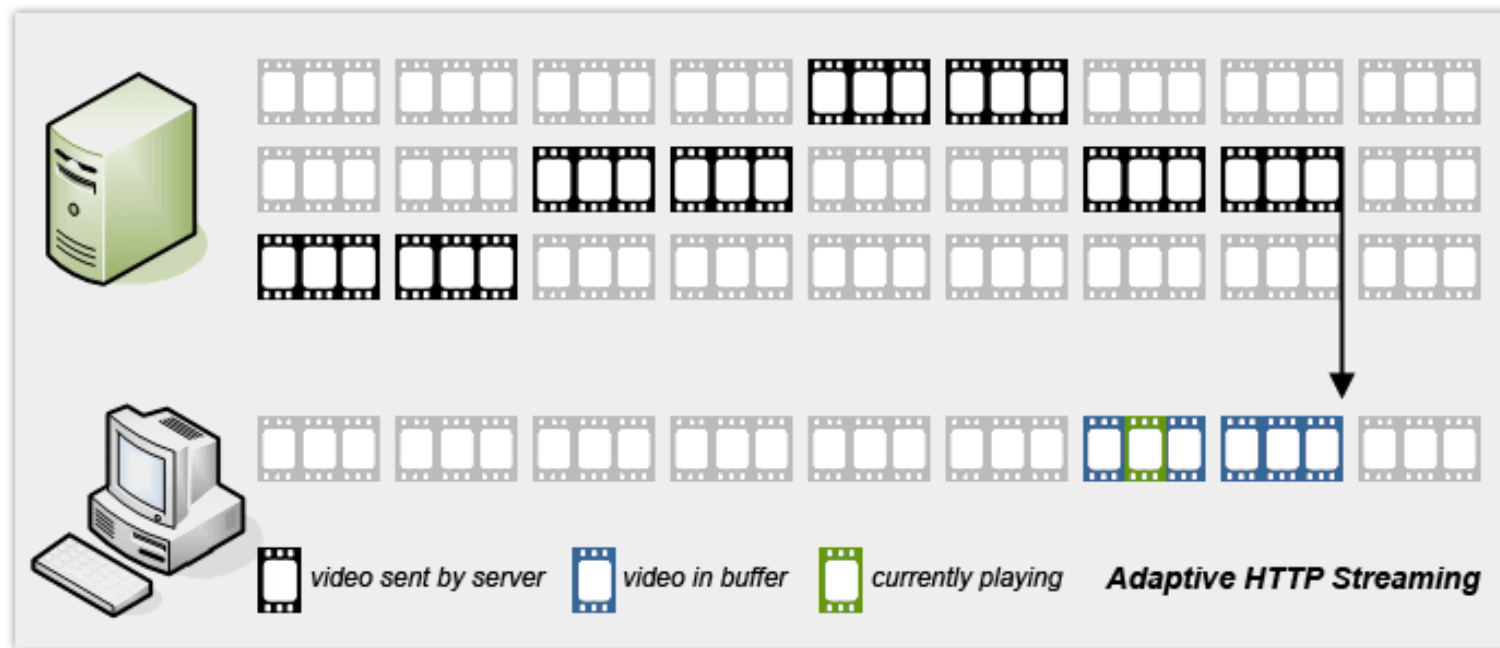
- Derroche de ancho de banda (se descarga incluso si no se ve)
- Incapacidad de adaptar la calidad "al vuelo"
- No válido para contenido "en vivo"

Streaming: RTP, RTCP, RTSP



- Requiere protocolo específico, y por tanto servidor específico (no vale un servidor web)
- Requiere un visualizador específico o *plug-in* (obsoletos)
- Estos protocolos específicos contemplan el control de la calidad

Solución intermedia: HTTP Adaptativo



- El servidor usa HTTP en modo *Chunked* para transmitir el vídeo "por trozos"
- El cliente puede pasar a pedir trozos de otro fichero, para controlar la calidad
- Estándar: DASH (*Dynamic Adaptive Streaming over HTTP*)
- No estándar (pero ampliamente implementado): HLS (de Apple)

5.2: Streaming (RTP)

Streaming

Para implementar un servicio de *streaming* son necesarios varios protocolos:

- El que transporta los *frames*. Se ocupa de enviar un solo *flujo* (audio o vídeo), dividiéndolo en paquetes de tamaño apropiado.
- El que transporta información de sincronización de los flujos.
- El que transporta estadísticas de errores, paquetes perdidos, etc. para ajustar el grado de compresión deseable.
- El que controla la reproducción de los flujos según las indicaciones del usuario (pausa, reproduce, rebobina...)

Protocolos habituales para cada caso

- **RTP** (*Real-time Transport Protocol*). Transporte de los *frames*. Suele implementarse sobre UDP.
- **RTCP** (*Real-time Control Protocol*) Sincronización e información de estadísticas y calidad (dos en uno)
- **RTSP** (*Real Time Streaming Protocol*) Establecimiento de la sesión, control de la reproducción.

Además, existen soluciones propietarias:

- **RTMP** (*Real-Time Message Protocol*). Va sobre TCP, y incorpora la funcionalidad de los tres anteriores. Utilizado por Adobe Flash.

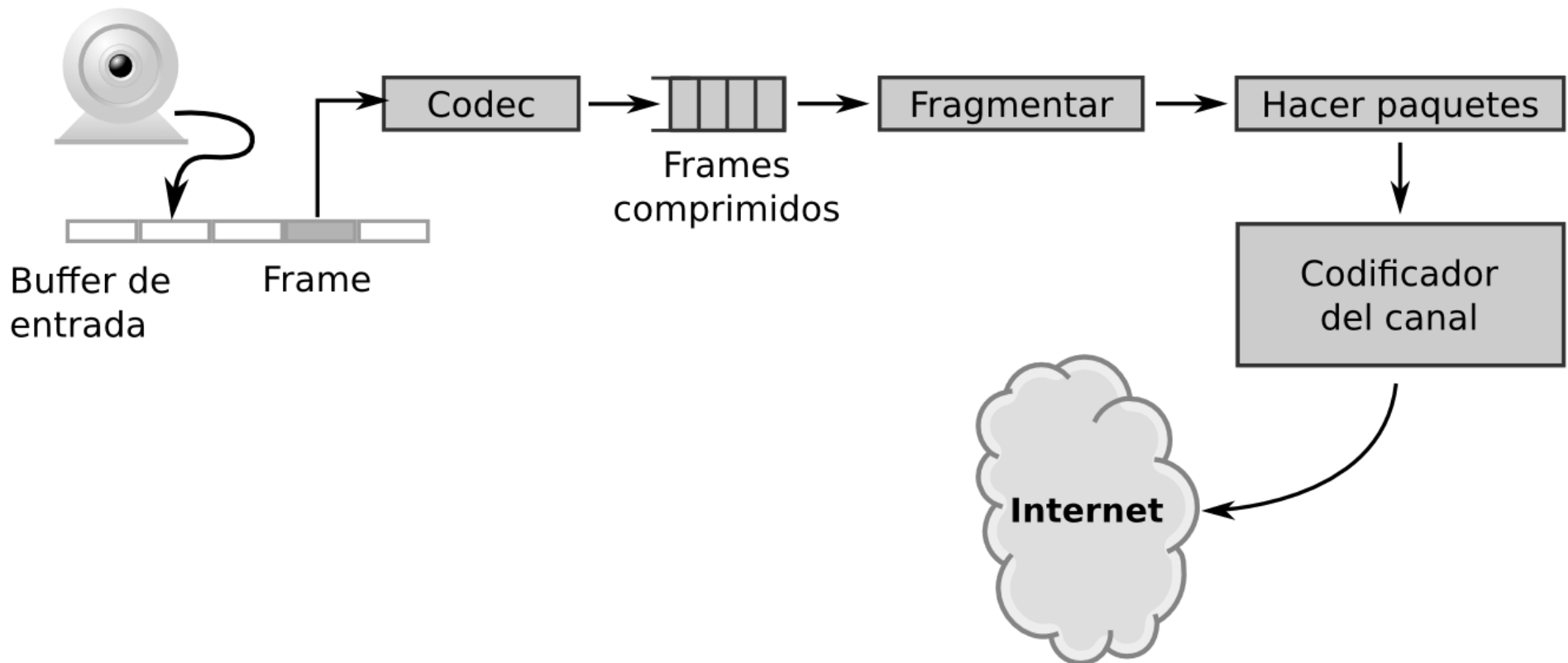
RTP

- Define un formato de paquete estandarizado para transmitir audio y vídeo.
 - No depende del protocolo de transporte por debajo
 - Aunque suele usar UDP por eficiencia
 - Cada flujo es un contenido (vídeo, audio) transmitido separadamente
 - Permite recibir y enviar múltiples flujos de diferentes orígenes y a diferentes destinos (admite *multicast*).

No sólo es para reproducción de video en Internet, se usa también para Voz sobre IP y video conferencia

RTP: visión muy general

- Un flujo de audio o video es capturado por un dispositivo
 - Se descompone en *frames*
 - que son comprimidos por un *codec*
 - y enviados al componente RTP que lo transmitirá

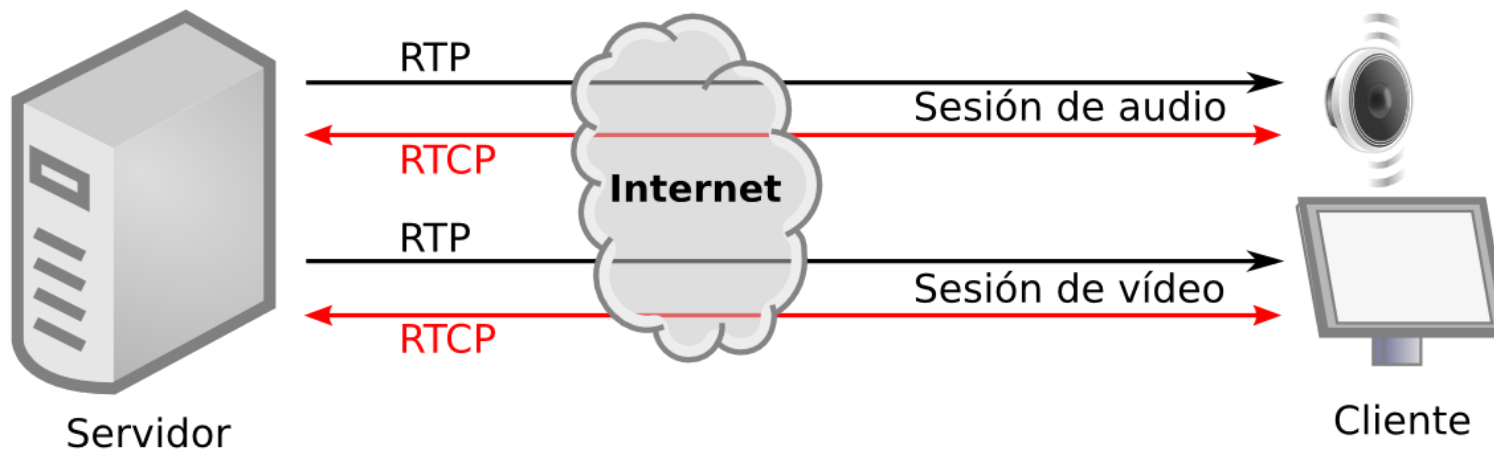


RTP: visión muy general

- Los *frames* a transmitir se empaquetan:
 - Si el *frame* es muy grande se divide en varios paquetes
 - Si es muy pequeño, un paquete puede contener varios *frames*
- Cada paquete lleva información de:
 - Número de secuencia (en que fueron enviados, para detectar pérdidas)
 - *Timestamp* (orden en que deben reproducirse) en unidades que dependen de cada flujo
 - Formato (si es audio o vídeo, qué *codec* usa, etc.) del *payload*
 - Identificador de la fuente (SSRC) ya que puede recibir flujos de varias fuentes. Es un identificador único numérico (abstracto).
- El *payload* es específico de cada *codec*

RTP: sesiones

- Una sesión es un "canal" que transmite audio o video (en realidad RTP es más genérico, pero este es su uso más frecuente)
- Cada sesión se identifica por IP:puerto de sus dos extremos
- El número de puerto es siempre par (el siguiente puerto impar es usado por RTCP)



Qué números de puerto se usarán, qué *codecs*, quién es el usuario detrás de esa IP y puerto, etc... no forma parte de RTP, sino que se negocian previamente mediante otros protocolos que ya veremos

RTP: Control (RTCP)

- RTCP es un protocolo asociado a RTP, siempre van juntos (similar a las dos conexiones usadas por FTP)
- Van siempre en puertos consecutivos (RTP va sobre pares, RTCP sobre impares)
- Su tráfico es mucho menor que el de RTP

Tipos de paquete transportados por RTCP:

- **RR** (Receiver Report). Con estadísticas sobre la sesión (paquetes perdidos, ratio de pérdidas, etc.)
- **SR** (Sender Report). Con la hora "universal" de su envío, y el *timestamp* equivalente del flujo RTP. Permiten sincronizar diferentes sesiones (audio y vídeo, *lip syncing*)
- **SDES** (Source Description) Asocia un "nombre" a los SSRC usados por RTP (*nick*, email, etc.)

RTP: Calidad de servicio

- El emisor recibe periódicamente paquetes **RR** de los receptores.
- Puede detectar si se producen muchas pérdidas de paquetes
- Lo cual es atribuible a insuficiente ancho de banda de los receptores.
- Puede decidir entonces cambiar a otro *codec* o formato con menor *bitrate*



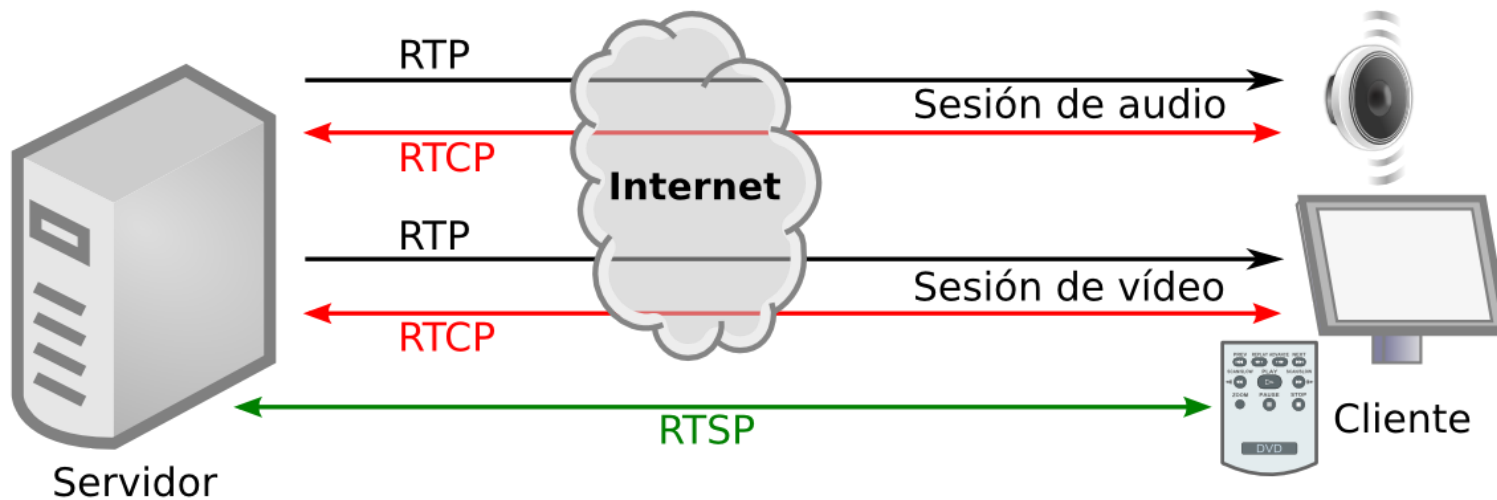
La decisión la toma el servidor, en función de los reportes que recibe de los clientes.

A diferencia de los métodos de HTTP streaming, en los que es el cliente quien debe solicitar una fuente de menor calidad si detecta pérdidas.

RTSP

Este protocolo actúa como "control remoto" permitiendo:

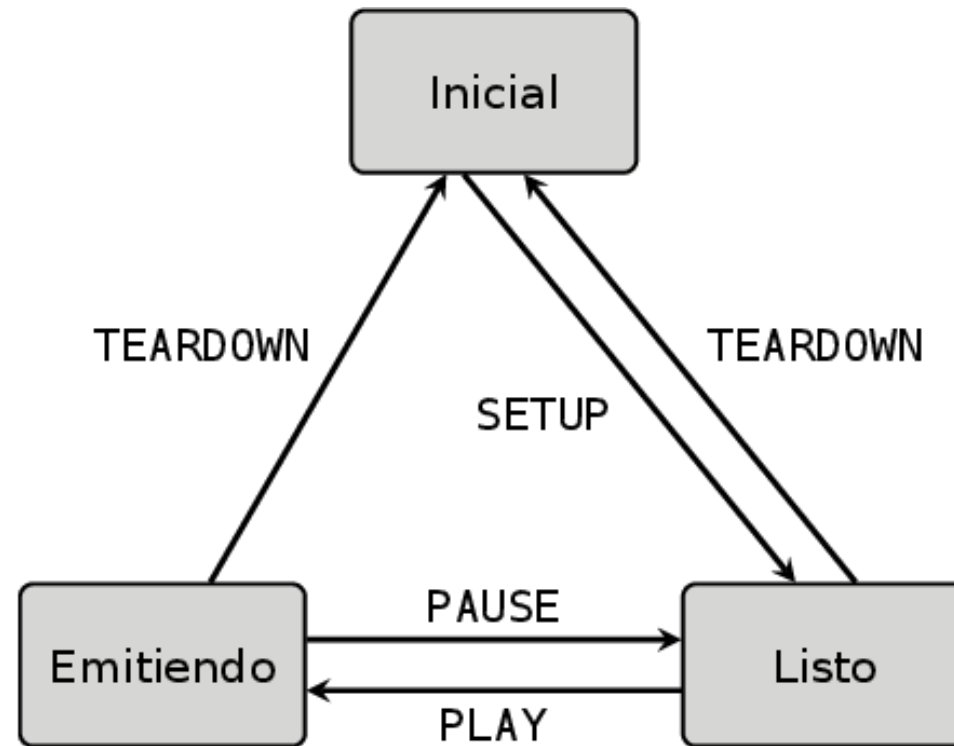
- Iniciar conexión con un servidor de *streaming*, acordando qué puertos usar para cada sesión (audio, vídeo), qué formatos (*codecs*) prefiere, etc.
- Pausar, continuar, avanzar, rebobinar en los flujos de las sesiones
- Añadir/cambiar flujos (ejemplo, cambiar el idioma del doblaje)



RTSP: características del protocolo

- Va sobre TCP
- Es un protocolo de texto, similar a HTTP
- Pero mantiene la conexión de forma permanente
- Tanto cliente como servidor pueden iniciar peticiones al otro
- Es un protocolo que *mantiene estado* según el último comando recibido

RTSP: Estados y comandos



- **DESCRIBE** obtiene información sobre un recurso
- **SETUP** fija los parámetros de la conexión RTP/RTCP
- **PLAY** inicia la reproducción
- **PAUSE** parada temporal (no se libera la sesión)
- **TEARDOWN** parada final (se libera la sesión)

RTSP: Sintaxis del protocolo

Las peticiones tienen una sintaxis similar a HTTP

- Primera línea: METODO * URI VersiónRTSP\r\n
- Más líneas opcionales de cabeceras de la forma *NombreCabecera: Valor*
 - Cada METODO puede requerir cabeceras específicas.
 - Una cabecera habitual es CSeq: con un número de secuencia para cada petición/respuesta
- Línea en blanco (\r\n) que marca el fin de las cabeceras
- Cuerpo opcional

Ejemplo:

```
DESCRIBE rtsp://servidor.ejemplo.com/carpeta/video RTSP/1.0
Cseq:312
```


RTSP: Sintaxis del protocolo (II)

Las respuestas también tienen sintaxis similar a HTTP

- Primera línea: VersionRTSP CódigoNumérico MensajeExplicativo
- Líneas opcionales de cabeceras, con la misma sintaxis que en las peticiones
- Línea en blanco (`\r\n`) que marca el fin de las cabeceras
- Cuerpo opcional
 - Pero este cuerpo nunca contiene el audio o vídeo, que va aparte por RTP u otros transportes.

Ejemplo:

```
RTSP/1.0 200 OK
CSeq: 5
Session: 12345678
```

RTSP, uso de SDP

SDP es otro protocolo (o más bien sintaxis de datos) para Descripción de Sesiones (**S**ession **D**escription **P**rotocol)

- SDP permite proporcionar información sobre un recurso audiovisual
- SDP es un formato de texto, muy conciso. Es una secuencia de líneas:
 - letra=cadena
- Cada letra tiene un significado concreto (ej: v es el número de versión, s el nombre de sesión, i su título, etc.) y la cadena asociada ha de tener una sintaxis concreta.
- SDP suele usarse desde RTSP en su respuesta al método DESCRIBE
 - Pero no está restringido a RTSP, también HTTP puede devolver una respuesta en SDP
 - Utilizado también en VoIP y videoconferencia

SDP: Ejemplo

El cliente envía:

```
DESCRIBE rtsp://servidor.ejemplo.com/carpeta/seminar RTSP/1.0  
Cseq:2
```

El servidor responde:

```
RTSP/1.0 200 OK  
CSeq: 2  
Content-Base: rtsp://servidor.ejemplo.com/carpeta/seminar  
Content-Type: application/sdp  
Content-Length: 460
```

```
v=0  
o=jdoe 2890844526 2890842807 IN IP4 10.47.16.5  
s=SDP Seminar  
i=A Seminar on the session description protocol  
u=http://www.example.com/seminars/sdp.pdf  
e=j.doe@example.com (Jane Doe)  
a=recvonly  
m=audio 49170 RTP/AVP 0  
m=video 51372 RTP/AVP 99  
a=rtpmap:99 h263-1998/90000
```

SDP: Ejemplo (sigue)

De especial interés en el ejemplo anterior son las tres últimas:

- **m** describe un recurso, el número de puerto para RTP y el *codec* usado
 - Tenemos audio en el puerto 49170 (y 49171 para RTCP)
 - Tenemos audio en el puerto 51372 (y 51373 para RTCP)
- El último número indicado en **m** está estandarizado
 - **0** indica un encoding de audio usando PCM μ -Law
 - **99** indica un formato *dinámico* (que se va a definir a continuación)
- **a** permite definir los formatos dinámicos.
 - En este caso se define **99** como el *codec* de vídeo **h263-1998**, con una resolución de reloj de 90KHz

Ejemplo completo: mostrando un vídeo en una página web

- ❶ El cliente solicita la página web (protocolo HTTP)
- ❷ El servidor le devuelve un documento HTML (protocolo HTTP)
- ❸ Dentro del documento, hay *tags* para incrustar un reproductor, capaz de manejar RTSP (el navegador web no puede). Por ejemplo QuickTime o VLC.

```
...  
<OBJECT classid='clsid:02BF25D5-8C17-4B23-BC80-D3488ABDDC6B'  
        width="320" height="240"  
        codebase='http://www.apple.com/qtactivex/qtplugin.cab'>  
    <param name='src' value="rtsp://ejemplo.com/carpeta/video.mov">  
</OBJECT>  
...
```

- ❹ El navegador carga el proceso (*plugin*) que va a reproducir el vídeo (en este ejemplo QuickTime) y le "cede" un área en la página donde pintar.

(...sigue)

- ⑤ El *plugin* analiza el tag `param` de donde obtiene la URI del servidor de streaming.
- ⑥ El *plugin* se conecta al puerto 554 (puerto por defecto de RTSP) de la máquina `ejemplo.com`
- ⑦ El *plugin* envía el comando `DESCRIBE` (protocolo RTSP) para obtener información acerca del recurso `/carpeta/video.mov`
- ⑧ El servidor (de Apple en este caso) abre el fichero contenedor, analiza los flujos almacenados, determina sus codecs y proporciona una respuesta en el lenguaje SDP (sobre protocolo RTSP)
- ⑨ El *plugin* envía el comando `SETUP` (protocolo RTSP) para iniciar la transmisión
- ⑩ El *plugin* abre puertos UDP para las sesiones RTP de audio y vídeo (según los datos que obtuvo en el paso 8). El servidor también.
- ⑪ Comienza la recepción del audio y vídeo (protocolo RTP/RTCP)

(...sigue)

- ⑫ El plugin sincroniza audio y vídeo y los muestra, a medida que llegan. Los *frames* se van descartando sin almacenarse a medida que se reproducen.
- ⑬ El *plugin* envía estadísticas (protocolo RTCP) al servidor, con las cuales éste puede modificar la calidad de la transmisión.
- ⑭ El usuario pausa el vídeo. El *plugin* recibe la interacción del usuario, y envía el comando **PAUSE** (protocolo RTSP) al servidor.
- ⑮ El servidor detiene el envío de frames de las sesiones RTP de audio y vídeo. No se está consumiendo ancho de banda.
- ⑯ El usuario continúa la reproducción. El *plugin* envía el comando **PLAY** (protocolo RTSP) y los flujos RTP se reanudan.
- ⑰ En algún momento el usuario detiene el vídeo o lo cierra. El *plugin* envía el comando **TEARDOWN** (RTSP). El servidor cierra las sesiones RTP y libera los recursos asociados.

