Tema 3: Aplicaciones Multimedia en la Red Internet

Albert Banchs
Redes Multimedia
Universidad Carlos III de Madrid

Palabras clase: Redes Multimedia, aplicaciones multimedia, aplicaciones interactivas, aplicaciones streaming

En este tema estudiaremos las aplicaciones multimedia en la red Internet actual y las técnicas y algoritmos que éstas utilizan para adaptarse al servicio Best Effort que la red proporciona.

3.1. El modelo Best Effort y las aplicaciones multimedia

Veamos primero con más detalle el modelo Best Effort en que se fundamenta la red Internet actual. En la arquitectura Internet, no se limita el número de usuarios que pueden utilizar la red ni la cantidad de tráfico que cada uno de ellos puede mandar. Como consecuencia, en caso de que en un momento determinado haya demasiados usuarios o la cantidad de tráfico enviada por éstos sea demasiado elevada, se producirá una situación de congestión en la red y la calidad del servicio percibida por todos los usuarios que estén utilizando la red será insuficiente.

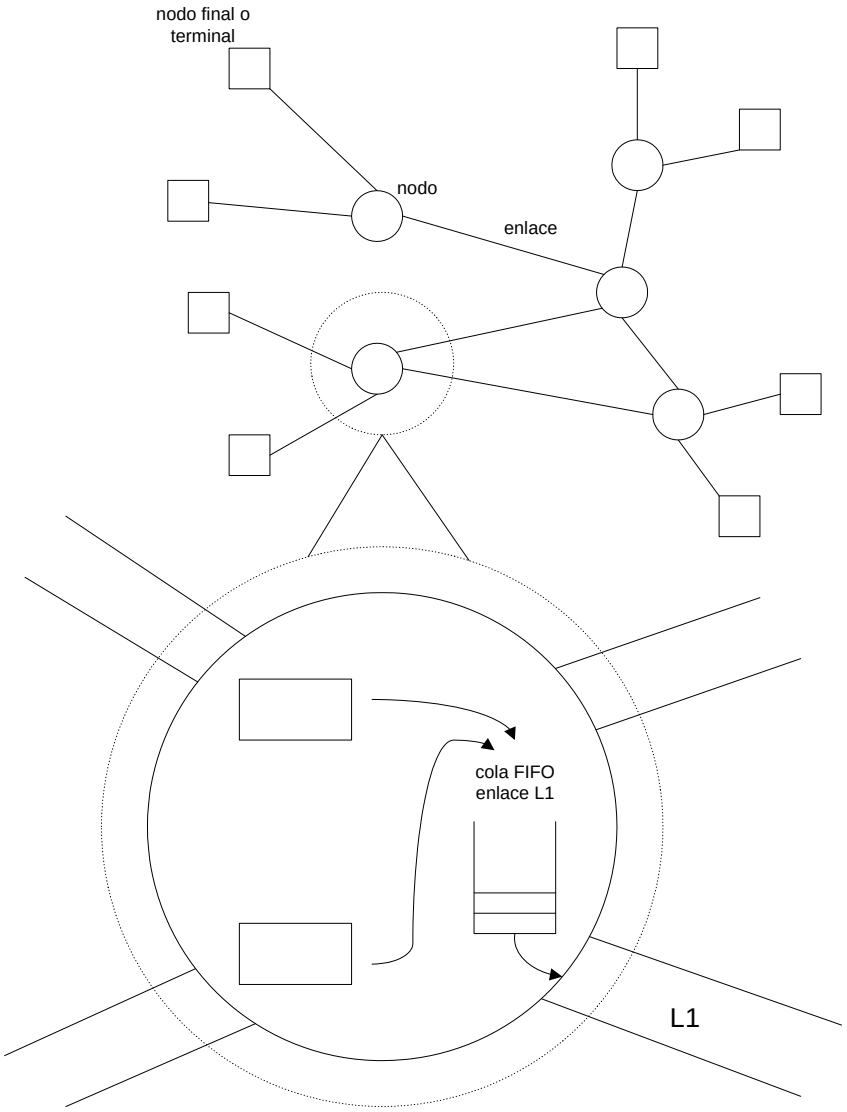
El modelo de servicio descrito anteriormente difiere claramente por ejemplo del modelo del servicio de telefonía. En este último se limita el número de usuarios que pueden utilizar la red (en caso de una sobrecarga, no se puede establecer una llamada). Se limita también la cantidad de tráfico que un usuario puede inyectar en la red (el correspondiente a una llamada de voz). Con ello, se puede conseguir un servicio con calidad de servicio que contrasta con el servicio Best Effort de Internet. Efectivamente, una llamada de voz establecida exitosamente disfruta siempre de una calidad constante y satisfactoria.

La red Internet se compone de enlaces y nodos. Los usuarios se ubican en los nodos finales. En la arquitectura no se impone ninguna restricción en la cantidad de tráfico que manda cada usuario, y ésta únicamente se ve limitado por el caudal físico de los enlaces de acceso.

La operación en los nodos es la siguiente. Al recibir un paquete, se determina cuál es el enlace de salida en base a la dirección destino del paquete. En caso de que este enlace de

salida esté libre, se procede a transmitir el paquete por este enlace. De todas formas, puede producirse que el enlace de salida esté transmitiendo otro paquete. En este caso, el paquete se almacena temporalmente en una cola asociada a este enlace de salida y cuando se libera el enlace se procede a mandar el paquete. Si antes de que se libere el paquete llegan otros paquetes dirigidos al mismo enlace de salida, éstos se almacenan en la misma cola a la espera de su turno. La disciplina de cola es la llamada FIFO (First-In First-Out) de modo que los paquetes se envían con el orden de llegada, independientemente de la naturaleza del paquete (voz, vídeo o datos). El tamaño de estas colas es limitado, de tal forma que si un paquete llega en un momento en que la cola está completamente llena, procedemos a descartar este paquete. Este comportamiento se denomina Drop-Tail.

La operación de un nodo IP se ilustra en la siguiente figura:



Una de las grandes ventajas que tiene la arquitectura descrita anteriormente es su simplicidad. La arquitectura Internet tal y como se ha descrito puede cursar fácilmente flujos de naturaleza (datos, voz, vídeo) y volúmenes diversos. Además la complejidad asociada a la operación, gestión y configuración de los nodos es mínima, lo que facilita enormemente la implantación de la red a gran escala. Todos estos factores han contribuido de forma muy decisiva al éxito y gran implantación de la red Internet.

Una de las desventajas de la arquitectura descrita anteriormente es que no permite proporcionar a un usuario garantías de caudal y retardo, con lo que por ejemplo no podemos asegurar que una llamada de voz recibirá una calidad mínima para que sea inteligible. Tampoco permite diferenciar entre paquetes, de tal forma que no se puede priorizar un paquete de voz con requisitos de retardo respecto a uno de datos sin requisitos. Estas limitaciones, si bien no suponían una limitación importante en los inicios de la red Internet, en la que ésta era concebida como una red primordialmente dedicada a los datos, sí que suponen un problema en la red Internet actual en la que el uso de aplicaciones multimedia está creciendo de forma sostenida.

Para hacer frente a la incapacidad de la red Internet para satisfacer los requisitos de las aplicaciones multimedia, los desarrolladores e investigadores arquitectos de la red Internet han tomado dos direcciones diferentes que corresponden a paradigmas distintos:

1. **Evolución de la red Internet**. Principalmente en la década de los 90 se propusieron múltiples arquitecturas para proporcionar calidad de servicio en Internet. Las arquitecturas más conocidas y que llegaron a estandarizarse fueron la de Servicios Integrados y la de Servicios Diferenciados, y esta última llegó a implementarse en la mayoría de nodos de la red Internet aunque su uso sigue siendo muy limitado. Con estas arquitecturas, la red Internet puede proporcionar el servicio que las aplicaciones multimedia requieren con lo que se soluciona el problema expuesto anteriormente. Es decir, con esta solución la red Internet se adapta a las necesidades de las aplicaciones.

2. **Adaptación de las aplicaciones**. Con este paradigma no se cambia la naturaleza y arquitectura de la red Internet sino que se cambia el diseño de las aplicaciones multimedia para adaptarlas al servicio proporcionado por la red Internet. La gran ventaja de esta solución es su simplicidad, ya que es mucho más fácil cambiar las aplicaciones que no cambiar la red. La desventaja es que al reducirse los caudales y aumentar los retardos, la calidad percibida por el usuario se degrada. Esta es la solución utilizada actualmente en la red Internet.

En este tema estudiaremos las aplicaciones multimedia basadas en el paradigma de la adaptación, mientras que las aplicaciones multimedia en una red evolucionada se estudiarán en el tema 4 de la asignatura.

A continuación se resumen las limitaciones del modelo Best Effort respecto al servicio ofrecido y se enumeran las diferentes técnicas para enfrentarse a ellas que se verán en detalle en los siguientes capítulos:

- **Retardo y jitter**. De acuerdo con la arquitectura vista anteriormente, tenemos un retardo en la entrega de un paquete que se compone básicamente de dos componente: el *retardo de propagación*, que tiene un valor fijo, y el *retardo de encolamiento*, que tiene un valor variable dependiendo del estado de las colas por las que pasa el paquete en su camino de origen a destino. Las técnicas para abordar y adaptarse a estos retardos van a depender de la naturaleza de la aplicación multimedia, puesto que hemos visto que las aplicaciones streaming e interactivas tienen requisitos distintos respecto al retardo. En el capítulo 2.2 de este tema veremos las técnicas utilizadas por las aplicaciones streaming, y en el capítulo 2.3 las utilizadas por las aplicaciones interactivas.

- **Caudal**. El caudal disponible para cada flujo multimedia dependerá del número de flujos multimedia utilizando la red y el volumen de cada uno de ellos. En caso que el volumen total de tráfico sea superior a la capacidad de la red, las colas se llenarán y se descartarán aquellos paquetes que no se pueden cursar con la capacidad disponible. Para adaptarse a ello, las aplicaciones multimedia (ya sean de naturaleza interactiva o streaming) tendrán que adaptar la tasa de transmisión al caudal disponible en la red. Estas técnicas de adaptación de la tasa de envío se verán en el capítulo 2.5.

- **Pérdidas y errores**. En caso de que no estemos en una situación de congestión permanente sino que únicamente se produzcan situaciones de congestión puntual a causa de la naturaleza a ráfagas del tráfico multimedia, no es necesario disminuir la tasa de transmisión. De todas formas, el problema que nos encontramos en este caso es que se pueden producir pérdidas puntuales a causa de que en un momento determinado se llene una cola. Esta situación es parecida desde el punto de vista de la aplicación multimedia al caso en que se producen errores de bits. En ambos casos, la aplicación tiene que enfrentarse a la incorrección o ausencia de parte de la información. A causa de los requisitos de retardo, esta información no se puede retransmitir. Las técnicas para recuperarnos de estas situaciones las veremos en el capítulo 2.4.

3.2. Aplicaciones streaming

Las aplicaciones streaming tienen como objetivo reproducir en recepción los contenidos multimedia que se envían desde el transmisor. El hecho que la comunicación en estas aplicaciones sea unidireccional permite un cierto margen de retardo, a diferencia de las aplicaciones multimedia, lo que hace que estas aplicaciones se puedan adaptar más fácilmente a la naturaleza Best Effort de Internet.

La operación de las aplicaciones streaming se resume en los siguientes puntos:

- El contenido multimedia se transmite del emisor al receptor mediante alguno de los siguientes protocolos de transporte: RTP/UDP, directamente UDP o TCP. Las consecuencias de elegir uno u otro protocolo de transporte se analizarán más adelante.

- Los paquetes recibidos se almacenan en el cliente (*client buffering*) durante un tiempo hasta el momento de la reproducción (*playout time*). Con esto, eliminamos el jitter introducido por la red, ya que mientras el retardo no sea tal que el paquete llegue después del playout time, el jitter no tiene impacto alguno.

- En el momento de reproducción, se decodifican los paquetes almacenados, y, en caso de errores o pérdidas de paquetes, se utilizan técnicas de ocultación de errores para minimizar el impacto sobre la calidad resultante.

- Se permite una cierta interactividad mediante un interfaz gráfico con los controles correspondientes.

Las aplicaciones streaming se pueden utilizar *integradas* en web browsers, de tal forma que al seleccionar una url se descarga el contenido multimedia. El diseño integrado de las aplicaciones multimedia con un browser se puede realizar de acuerdo con las siguientes estrategias:

- La solución más simple es la de descargar el fichero de forma completa en el web browser y una vez descargado entregárselo al reproductor multimedia (*media player*) para su reproducción. Esta solución es claramente ineficiente porque se

tiene que esperar mucho tiempo (hasta que la descarga completa ha terminada) para empezar a reproducir el fichero. Una solución claramente mejor es ir reproduciendo el fichero a medida que recibimos los datos, con lo que podemos tener tiempos de espera muy reducidos.

- Una solución que incorpora la idea anterior es la que se expone a continuación. Con esta solución, el browser obtiene el metafile (como veremos más adelante el metafile indica la ubicación del fichero multimedia) del fichero multimedia y se la proporciona al reproductor multimedia, que a partir del metafile se descarga el fichero multimedia correspondiente y lo reproduce a medida que lo va descargando. Esta solución conlleva tiempos de espera mucho menores que la solución anterior. De todas formas, adolece aún de algunos inconvenientes: al realizarse la descarga directamente desde el web server, el reproductor multimedia realiza la descarga utilizando el protocolo HTTP, que utiliza TCP en el nivel de transporte. Como veremos posteriormente, el protocolo TCP no es adecuado para las descargas multimedia, por lo que sería deseable una solución mediante la cual la descarga se pudiera realizar desde un servidor distinto al web server.

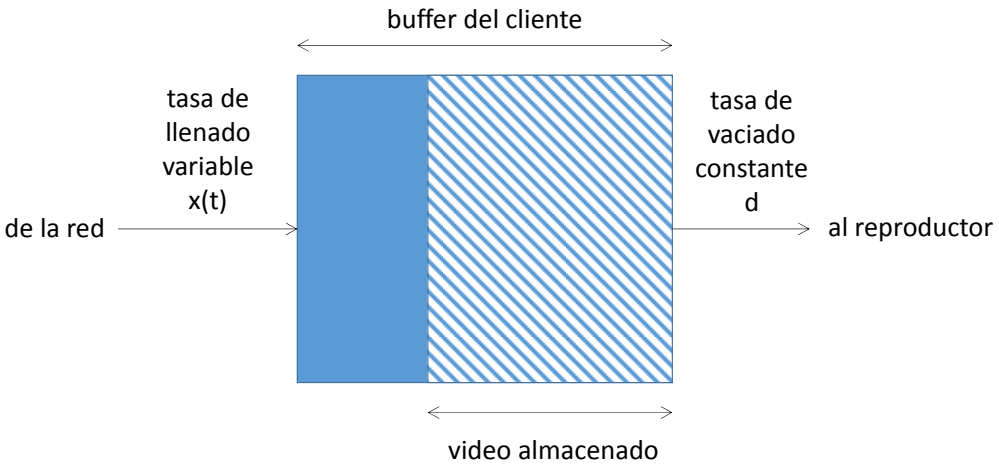
- La solución siguiente supera los problemas descritos anteriormente y es la más adecuada para integrar las aplicaciones streaming con el servicio web. Con esta solución, el servicio web y la aplicación streaming se separan completamente. Mediante el protocolo HTTP, el web browser obtiene la ubicación del fichero multimedia **el cual está ubicado en un servidor streaming y no en el servidor web**. Una vez obtenida la referencia, el web browser se la pasa al reproductor multimedia el cual procede a descargar el fichero multimedia del servidor streaming **utilizando un protocolo distinto a HTTP**. De esta forma se puede realizar la descarga mediante el protocolo de transporte UDP en lugar de TCP, aunque en esta última solución podría optarse también por utilizar el protocolo TCP para la descarga del fichero multimedia.

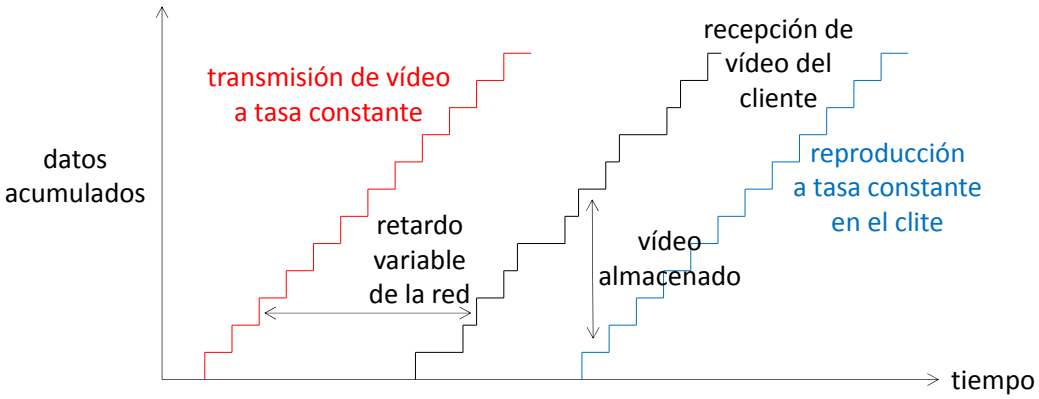
En la solución de la figura anterior, existen diferentes opciones para la transmisión del contenido de audio y vídeo desde el *streaming server* hasta el *media player*. Entre otras, disponemos de las siguientes opciones:

- Enviar el contenido de audio/vídeo con UDP a una tasa constante igual a la tasa de reproducción del receptor de acuerdo con la codificación utilizada. Por ejemplo, si se usa la codificación GSM, cuya tasa es de 13 Kbps, entonces el servidor envía el fichero audio a una tasa de 13 Kbps. Al recibir el contenido

audio, el receptor lo reproduce inmediatamente. Para que esta opción funcione bien, la tasa de llegadas **x(t)** tiene que ser exactamente igual a la tasa de reproducción **d**. En caso de que la red introduzca un cierto jitter, las llegadas no se producen a una tasa exactamente constante por lo que esta opción no funcionará bien.

- Para solventar el problema anterior, una opción es esperar un tiempo de *playout* de entre 1-10 segundos en el *media player* para eliminar el jitter introducido por la red. Esto se hace almacenando en un **buffer** el contenido multimedia recibido por la red. Una vez hemos almacenado unos cuantos segundos de audio/vídeo en el receptor, empieza la reproducción a una tasa constante. De esta forma, variaciones puntuales en la tasa de llegada x(t) ocasionadas por el jitter son absorbidas por el **client buffer** y son transparentes para la reproducción.



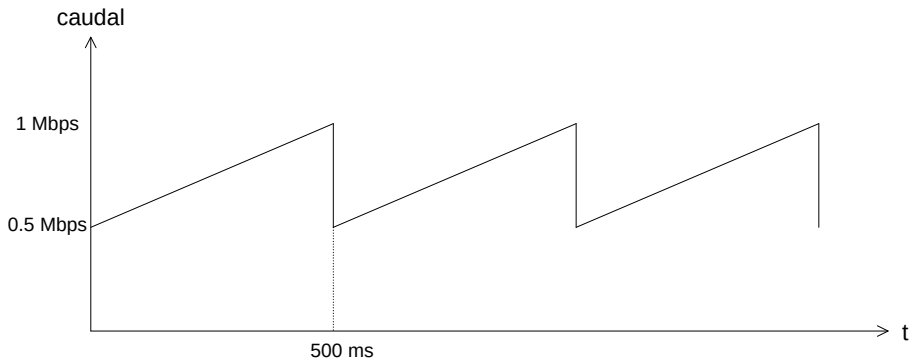


- La última opción es utilizar el buffer de cliente como en el caso anterior pero transmitiendo el contenido multimedia mediante el protocolo TCP en lugar de UDP. En este caso (al igual que el anterior) vamos a esperar un retardo inicial de entre varios segundos y, pasado este tiempo, vamos a proceder a vaciar el buffer a una tasa constante igual a la tasa de reproducción. La ventaja en este caso es que, dado que TCP retransmite los paquetes perdidos, potencialmente tenemos la capacidad de proporcionar una calidad de sonido superior que con UDP. La desventaja es que vamos a tener una mayor fluctuación en la tasa de llegada a causa de los algoritmos de control de congestión y de flujo ejecutados por TCP. Efectivamente, una pérdida en TCP reduce la tasa de emisión con lo que puede ocurrir que durante períodos prolongados x(t) sea inferior a d. En este caso puede que se vacíe el buffer resultando una pausa no deseada en la reproducción del fichero de audio o vídeo.

En la tercera opción (TCP) el comportamiento dependerá de manera crítica de la memoria disponible para el buffer del cliente. En caso de que esta memoria sea suficientemente grande, TCP hará uso de todo el caudal disponible y x(t) puede ser muy superior a d. Así, una parte importante del contenido multimedia puede almacenarse en el buffer, con lo que es poco probable que éste se llegue a vaciar. Por otro lado, si el buffer del cliente es pequeño, la tasa de llegadas x(t) fluctuará alrededor de d y la probabilidad de que se vacíe el buffer será muy superior.

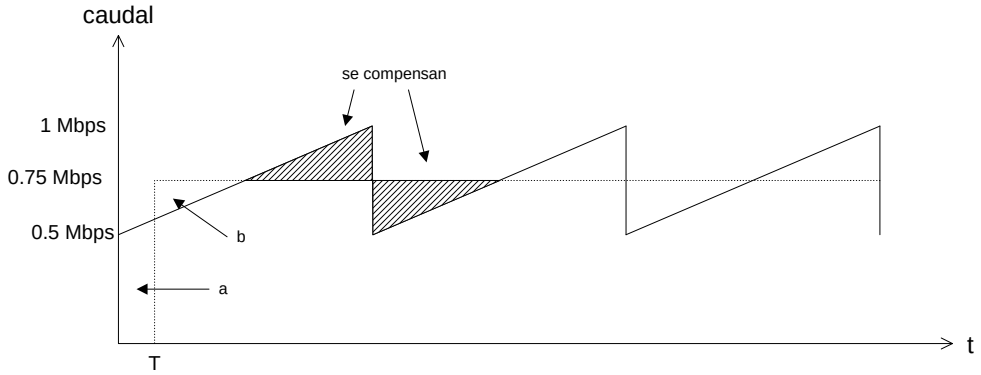
Ejemplo 1: Se utiliza una aplicación streaming para reproducir un vídeo de tasa 750 Kbps sobre un enlace vacío de caudal 1 Mbps. ¿Qué tiempo de playout se tendrá que utilizar para que no se produzcan interrupciones si se utilizan los protocolos UDP y TCP respectivamente?

NOTA: Asuma el siguiente modelo para el comportamiento de la tasa de envío del protocolo TCP:



Respuesta: Para el caso de UDP, y siempre que el enlace permanezca desocupado, podemos transmitir a una tasa de 750 Kbps sin sufrir pérdidas y con un jitter nulo, por lo que no es necesario ningún tiempo de espera para reproducir los paquetes recibidos y por lo tanto el playout será igual al tiempo de propagación más el de transmisión.

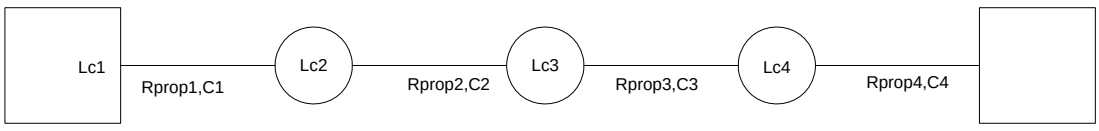
Para el caso TCP, la condición que debe cumplirse es que el área por debajo de la curva x(t) (volumen de datos recibidos) sea siempre superior al área por debajo de la curva d(t) (volumen de datos reproducidos). Esto se producirá cuando las áreas *a* y *b* sean iguales, puesto que las demás se compensan (ver gráfico):



Si se soluciona el problema trigonométrico a = b, se puede ver que el valor de T para el que se cumple esta condición es 41,6 ms. El tiempo de playout, por lo tanto, tendría que tomar este valor más los tiempos de propagación y transmisión.

Observación: El valor resultante del ejemplo anterior es muy inferior a los valores de playout que se manejan en la práctica. El motivo es que el comportamiento del protocolo TCP en el caso genérico de enlaces compartidos con otros flujos puede ser mucho peor al del modelo utilizado en el ejemplo para el caso de enlace libre. Lo mismo para el protocolo UDP.

Ejemplo 2: ¿Cuál es el tiempo de playout que nos garantiza que no se va a vaciar nunca el buffer en la siguiente red si utilizamos el protocolo **UDP**?



Respuesta: Los retardos de transmisión y propagación son constantes y por lo tanto no afectan al jitter. El único retardo que es variable y afecta el jitter es el de encolamiento. El

buffer deberá ser suficiente para absorber el jitter máximo que puede haber en la red, que corresponde al caso en que un paquete encuentra todas las colas vacías y el siguiente las encuentra todas llenas. Para compensar este caso, deberemos tener almacenados en el buffer del cliente suficientes paquetes para que la reproducción no se detenga en este intervalo. Esto corresponde a:

$$B = d\sum_{i=1}^{4}\frac{l_{c,i} - l}{C_i}$$

Observación: En la práctica, las aplicaciones desconocen la configuración de la red a través de la cuál transmiten, por lo que no es posible calcular el playout time de la forma que se ha hecho en el ejemplo 2. Lo que se hace es utilizar para el playout time valores conservadores que sean superiores a los jitters máximos que podamos tener en la red Internet. Para el caso UDP, esto se puede calcular reproduciendo el cálculo del ejemplo 2 para un número máximo de nodos (colas) en el camino de transmisión. La cota superior que se utiliza en estos casos está entre 1 y 10 segundos[1]. Para el caso TCP, como se ha razonado anteriormente, se requieren playout times superiores. Esto se puede hacer para el caso de aplicaciones streaming en las que tener un valor conservador para el playout time no supone ningún problema. Para el caso de las aplicaciones interactivas el tener valores del playout time pequeños es mucho más crítico por lo que este criterio no es apropiado. La selección del playout time para aplicaciones interactivas se estudiará en el siguiente capítulo.

Real Time Streaming Protocol (RTSP)

Una de las funcionalidades que se requiere en las aplicaciones streaming es la de poder controlar la reproducción, entre otras acciones: congelar la transmisión, reposicionar la transmisión a instantes anteriores o posteriores, o aumentar la velocidad de reproducción. Esta funcionalidad es similar a la que ofrece un aparato de vídeo. Para este fin, se ha diseñado el Real Time Streaming Protocol (RTSP), cuya función es la de intercambiar los mensajes de control correspondientes a este tipo de acciones.

De entre las funciones que el protocolo RTSP **no** realiza, tenemos las siguientes:

- RTSP no define los codecs utilizados para codificar audio/vídeo.

[1] Efectivamente, si consideramos una cota superior para la longitud de la cola de 300 paquetes de 1500 bytes cada uno, una cota inferior del caudal de 10 Mbps y un máximo de 60 saltos en el camino extremo a extremo, esto nos da un retardo de unos pocos segundos, lo que corresponde al orden de magnitud utilizado en las aplicaciones reales.

- RTSP no define como encapsular los paquetes transmitidos. Para ello se puede utilizar el protocolo RTP o bien un protocolo propietario.

- RTSP no define como se realiza el transporte. Éste se puede realizar con TCP o bien con UDP.

- RTSP no define la política de buffering y reproducción seguida. El contenido multimedia se puede reproducir inmediatamente desde que llega, puede tardar varios segundos o se puede esperar a que finalice la descarga antes de reproducir el contenido.

RTSP es un protocolo que se denomina *fuera de banda* o *out-of-band* puesto que los paquetes RTSP se transmiten de forma separada e independiente al contenido multimedia. Concretamente, los mensajes RTSP utilizan un puerto diferente (el 544) al del contenido multimedia. Los paquetes RTSP se pueden transmitir por UDP o TCP.

Los mensajes que se transmiten mediante el protocolo RTSP incluyen las acciones pause/resume, reposicionamiento de la reproducción y fast forward/rewind.

3.3. Aplicaciones interactivas

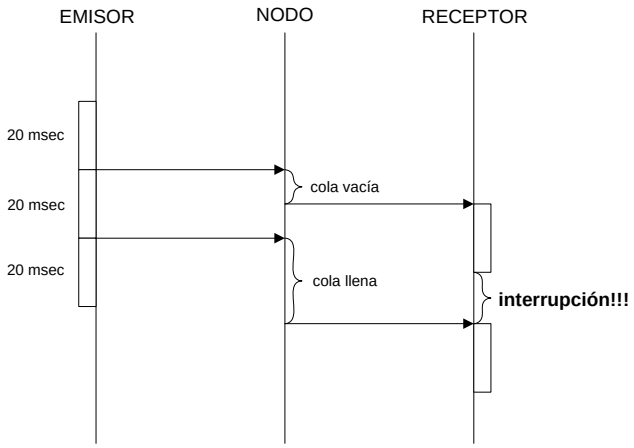
La técnica de utilizar un tiempo de playout conservador de entre 1 y 10 segundos, tal y como se hacía en el caso de aplicaciones streaming, no es apropiada para las aplicaciones interactivas. Éstas tienen unos requisitos temporales más restrictivos, por lo que será necesario utilizar playout times lo más pequeños posibles. En este capítulo vamos a estudiar las técnicas que se utilizan para calcular el playout time en este tipo de aplicaciones. El estudio se va a centrar en una aplicación de telefonía sobre Internet (*Internet phone*) que describimos a continuación, aunque las técnicas utilizadas en el contexto de vídeo son similares.

En nuestra aplicación Internet phone, cada participante alterna períodos de actividad con otros de silencio. Al tráfico generado en un período de actividad lo vamos a llamar *talk spurt*. Durante los períodos de actividad, el emisor genera bytes a una tasa de 8 Kbytes por segundo, y cada 20 msec se envía un paquete con los bytes generados durante este período. Por lo tanto, el número de bytes incluidos en un paquete es de 20 msec x 8 Kbytes/sec = 160 bytes. Este paquete, con la cabecera correspondiente, se encapsula dentro de un segmento UDP y se envía a través de la red. Por lo tanto, durante los talk spurts se envía un segmento UDP cada 20 msec.

Para aplicaciones altamente interactivas, como Internet phone, retardos inferiores a 150 msec no son percibidos por el oído humano. Retardos de entre 150 y 450 msec, aunque no son ideales, producen una calidad aún aceptable con este tipo de aplicaciones. Retardos superiores a 450 msec dificultan de forma seria la interactividad y por lo tanto no son aceptables. El diseño de la aplicación Internet phone tiene que cumplir estos requisitos. Por ejemplo, un paquete que llegue con un retardo superior a 450 msec no se podrá reproducir y se tendrá que descartar.

Como hemos visto en el capítulo anterior, en la red Internet se pueden producir pérdidas. Una de las opciones que se contemplaba en el caso de las aplicaciones streaming era utilizar el protocolo TCP para retransmitir los paquetes perdidos. De todas formas, los mecanismos de retransmisión no son apropiados para el caso de las aplicaciones interactivas, puesto que conllevan retardos demasiado elevados. Además, TCP reduce la tasa de emisión al detectar pérdidas, lo que puede impactar de forma crítica la calidad de la voz si esta tasa de transmisión es inferior a la de reproducción. Es por estos motivos que todas las aplicaciones interactivas existentes utilizan el protocolo UDP y no incluyen mecanismos de reproducción. Sí que incluyen otros mecanismos para minimizar el impacto de las pérdidas, tal y como veremos en el capítulo 2.4.

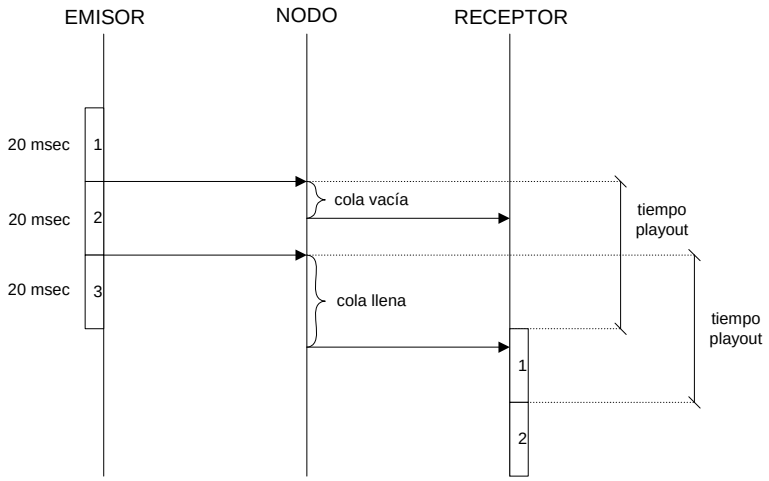
El principal problema a abordar en el diseño de la aplicación Internet phone es similar al que teníamos con las aplicaciones streaming: el **jitter**. A causa del jitter, tenemos que el retardo que se produce desde que se transmite un paquete hasta que éste llegue al receptor fluctúa. Por ejemplo, podemos tener que generemos un primer paquete y un segundo paquete 20 msec después pero que en el receptor el espacio entre el primer y el segundo paquete sea superior a 20 msec. Esto se produciría, por ejemplo, si el primer paquete encuentra una cola vacía y el segundo la encuentra llena. El problema surge si el receptor ignora los efectos del jitter y reproduce el primer paquete en el mismo instante en el que es recibido. Al terminar la reproducción del primer paquete, el segundo no ha llegado todavía y por lo tanto se produce una interrupción. Como en una red de jitter no despreciable, como es Internet, esto se produce a menudo, la voz resultante de aplicar esta técnica sería fácilmente ininteligible.



La solución al problema anterior es análoga a la que habíamos visto en el capítulo 2.2 para las aplicaciones interactivas:

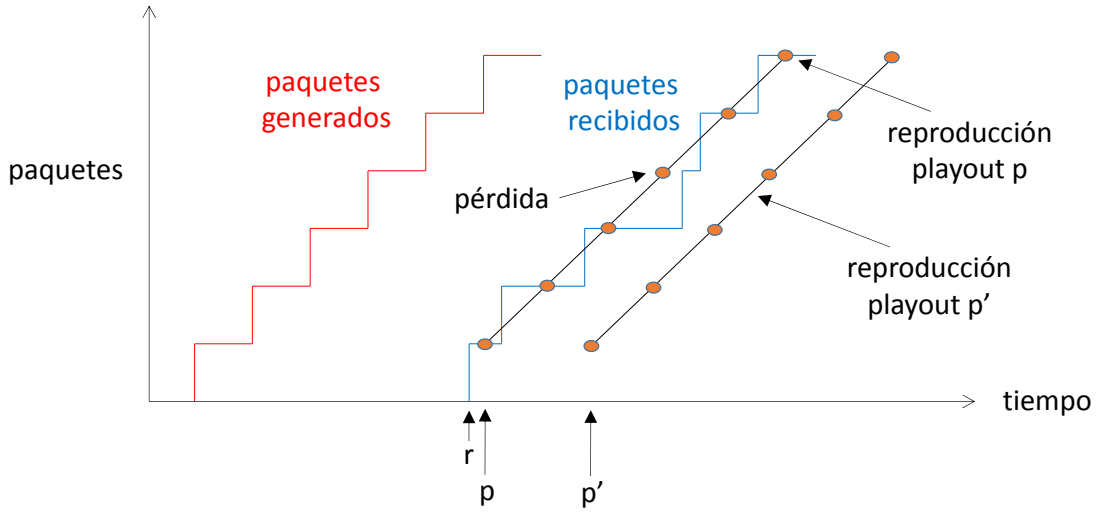
- Cada vez que mandamos un paquete, incluimos en el mismo un sello indicando el instante en el que se generó el paquete (*timestamp*). Éste se puede incluir en el campo correspondiente, en la cabecera RTP o bien en algún campo del paquete, en caso de que se utilice un protocolo propietario.

- En el receptor no se reproduce el contenido del paquete en el instante en que se recibe sino que se retarda la reproducción del mismo en un tiempo llamado *playout* respecto a la referencia temporal indicada por el transmisor. El tiempo de playout tiene que ser suficientemente grande como para que los paquetes lleguen antes que el instante de reproducción correspondiente, ya que los paquetes que llegan después del tiempo de playout son descartados.

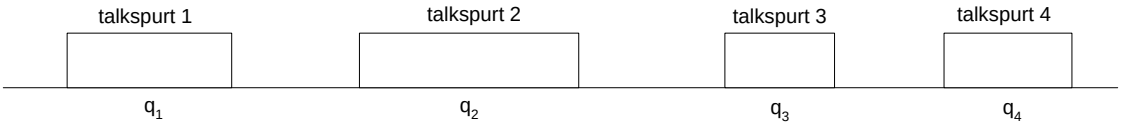


Con la técnica anterior, un paquete con timestamp *t* se reproducirá en el instante *t+q*, siendo *q* el playout time. La elección del valor del playout es crítico para el rendimiento de la aplicación. El valor máximo que puede tomar *q* de acuerdo con el razonamiento anterior es de 450 msec, ya que si toma valores superiores el rendimiento se degradará. De todas formas, nos interesa que el valor de *q* sea lo más pequeño posible para maximizar la sensación de interactividad percibida por el usuario. Por otro lado, si el valor de *q* se escoge demasiado pequeño, muchos paquetes pueden llegar demasiado tarde al receptor y no se reproducirán, degradando así el rendimiento percibido. En general, si el jitter es elevado nos interesará configurar *q* con un valor grande, y si es pequeño nos interesará configurar *q* con un valor pequeño.

El compromiso en la elección del playout time se muestra en la siguiente figura. En la figura se puede observar que en un talkspurt los paquetes se generan y se reproducen a una tasa constante. Se consideran dos tiempos de reproducción distintos: *p* y *p'*. Con el primero, el cuarto paquete no llega a tiempo y el receptor considera que se ha perdido. Con el segundo, todos los paquetes llegan a tiempo por lo que no se producen pérdidas.



El ejemplo anterior muestra la importancia de la elección del tiempo de playout. Idealmente, nos gustaría minimizar este tiempo bajo la condición que el número de paquetes perdidos sea pequeño. La forma natural de abordar este compromiso es medir el retardo y jitter de la red y ajustar el tiempo de playout de acuerdo con las medidas realizadas. El ajuste del tiempo de playout se realiza al principio de un talkspurt y se mantiene a lo largo de todo el talkspurt. En función de como se reajuste el tiempo de playout, el período de silencio se comprimirá o alargará, pero esto no es perceptible por el usuario.



A continuación describimos un algoritmo genérico de ajuste de los tiempos de playout. Definamos las siguientes variables:

t_i = timestamp (tiempo de generación) del paquete *i*
r_i = tiempo de recepción del paquete *i*
p_i = tiempo de reproducción del paquete *i*

El retardo del paquete *i* viene dado por $r_i - t_i$. A causa del jitter, este retardo puede variar de un paquete a otro. En la variable d_i estimamos el retardo medio de acuerdo con la siguiente formula:

$$d_i = (1-u)\, d_{i-1} + u\, (t_i - r_i)$$

donde *u* es una constante fija (e.g. u = 0.01) y t_i se obtiene del timestamp del paquete recibido. La fórmula anterior lo que hace es actualizar la media anterior (d_{i-1}) con respecto al último retardo observado ($t_i - r_i$) utilizando la constante u para fijar el peso de cada uno de estos dos factores.

La otra variable que estimamos es la desviación del retardo respecto a la media:

$$v_i = (1-u)\, v_{i-1} + u\, |t_i - r_i - d_i|$$

Las estimaciones de d_i y v_i se hacen para cada paquete recibido, aunque únicamente se utilizan para calcular el tiempo de playout al inicio de cada talkspurt. El tiempo de reproducción al inicio de un talkspurt se calcula de la siguiente forma:

$$q_i = d_i + K\, v_i$$

donde K es una constante (e.g. K = 4) que representa la salvaguarda que usamos para asegurar que todos los paquetes llegarán a tiempo. De esta forma, el instante de reproducción de todos los paquetes *j* del mismo talkspurt se calcularán de acuerdo con:

$$p_j = t_j + q_i$$

Una de las condiciones para la ejecución del anterior algoritmo es que el receptor pueda detectar el primer paquete de un talkspurt como tal. En caso de que no se produzcan pérdidas, esto podemos detectarlo simplemente con los timestamps: si el timestamp de dos paquetes consecutivos difieren en 20 msec forman parte del mismo talkspurt, y si la diferencia es superior, el segundo paquete es el primero de un nueva talkspurt. En caso de que se puedan producir pérdidas, necesitamos utilizar números de secuencia además de los timestamps para determinar si una diferencia en timestamps de más de 20 msec es consecuencia de los paquetes perdidos o del inicio de un nuevo talkspurt.

El algoritmo visto anteriormente funciona bien siempre que no se produzcan variaciones drásticas en los retardos observados. Si a partir de un momento determinado, todos los retardos pasan a ser muy superiores, la adaptación de los tiempos de playout con el algoritmo anterior puede ser demasiado lenta. En [4] se presentan extensiones del algoritmo descrito aquí para este caso.

3.4. Pérdidas: Técnicas de recuperación

En este capítulo abordamos los mecanismos que utilizan las aplicaciones multimedia para enfrentarse a las pérdidas. En el contexto de estas aplicaciones, el concepto de pérdida engloba tanto a las pérdidas propiamente dichas (es decir, aquellas que se producen por descartes en la red, por ejemplo a causa de overflow en las colas) como a aquellos paquetes que llegan al destino en un instante de tiempo posterior al planificado para su reproducción de acuerdo con el tiempo de playout empleado.

La forma más inmediata de recuperarse de las pérdidas es retransmitir aquellos paquetes que se han perdido. Esta es la técnica empleada, por ejemplo, por el protocolo de transporte TCP. Tal y como se detalla a continuación, este mecanismo no es factible para las aplicaciones interactivas, y si bien es factible para las aplicaciones streaming, no es siempre deseable.

En las aplicaciones interactivas, independientemente de si una pérdida es causada por un descarte en el buffer o por un retardo superior al tiempo de playout, la retransmisión no es una solución adecuada. En el primero de los casos, se produce un descarte a causa de que la cola está llena. El hecho de que la cola esté llena implica retardos de encolamiento elevados, por lo que en caso de utilizarse retransmisiones, el paquete retransmitido llegaría con toda probabilidad con un retardo superior al tiempo de playout, y por lo tanto no podría ser utilizado. En el segundo caso (retardo superior al playout), la retransmisión tampoco representa solución alguna por motivos obvios.

En la otra familia de aplicaciones multimedia, las streaming, hemos visto que las retransmisiones sí pueden utilizarse, aunque su uso conlleva una serie de desventajas (concretamente, la necesidad de emplear tiempos de playout superiores). El uso de las retransmisiones en las aplicaciones streaming tiene que combinarse con mecanismos de regulación de la tasa de envío (como TCP) que eviten un efecto de avalancha. Estos mecanismos pueden llevar a que durante ciertos intervalos de tiempo la tasa de envío sea inferior a la de reproducción, y, para que esto no lleve a que se vacíe el buffer de las aplicaciones streaming, es preciso utilizar un tiempo de playout mayor, tal y como se ha visto en el capítulo 2.2.

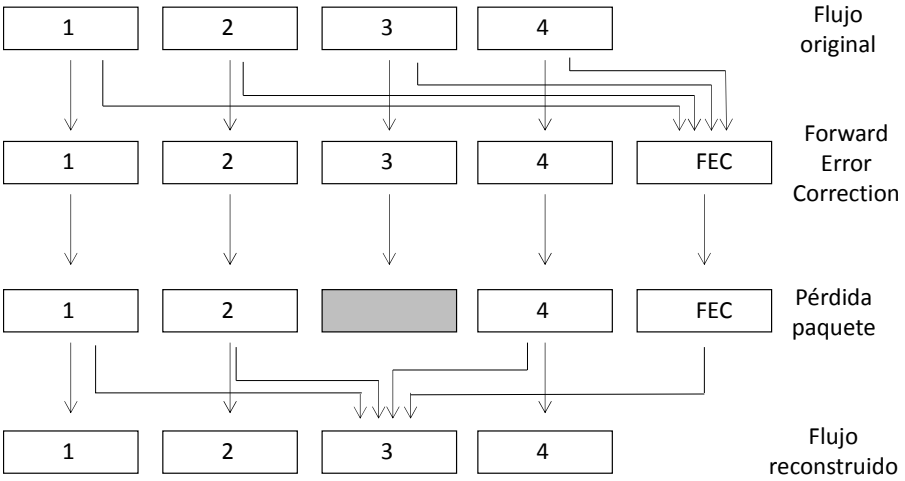
En el resto de este capítulo nos centraremos en técnicas de recuperación de pérdidas alternativas a la retransmisión. Concretamente, abordaremos las técnicas de Forward Error Correction (FEC), Entrelazado o Interleaving, y Reconstrucción de secuencias dañadas.

Forward Error Correction (FEC)

La idea básica de los mecanismos FEC es la de añadir información redundante al flujo multimedia enviado. Al precio de un aumento marginal del caudal enviado, esta información redundante permite, en caso de que se produzcan algunas pérdidas, reconstruir o bien la señal original o bien una aproximación de ésta. A continuación describimos tres mecanismos FEC. El primero de ellos es genérico y puede ser tanto utilizado para aplicaciones de audio como de vídeo, mientras que los otros dos son específicos para audio y vídeo respectivamente.

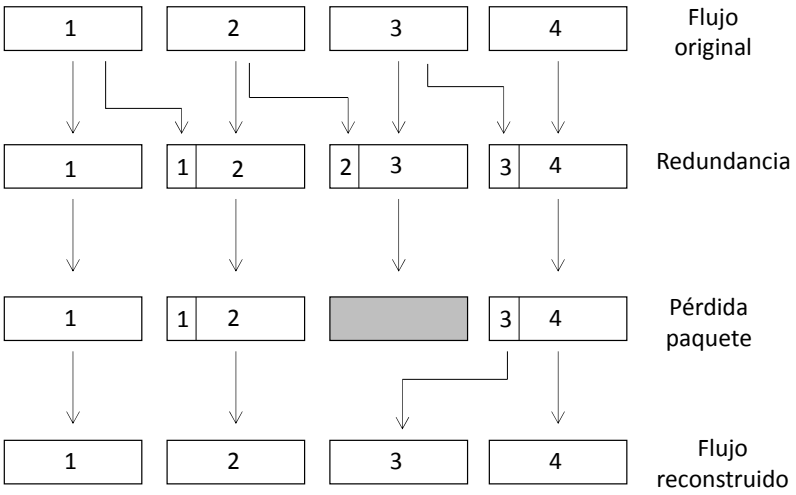
La primera de las técnicas FEC funciona de la siguiente forma. Por cada n paquetes enviados, se envía un paquete redundante. Este paquete redundante se obtiene de realizar una OR exclusiva de los n paquetes. Con esta operación, el número total de 1's que tenemos en una misma posición entre los n+1 paquetes es un número par. De esta forma, si cualquiera de los n+1 paquetes del grupo se pierde, el receptor puede reconstruir el paquete perdido. De todas formas, si se pierden dos o más paquetes, la reconstrucción no se puede llevar a término. Siempre que mantengamos el valor del grupo (n+1) pequeño, podremos recuperar la mayoría de los paquetes perdidos siempre que las pérdidas no sean excesivas. De todas formas, esto tiene un precio, ya que cuanto más pequeño sea el grupo, mayor es el aumento de la tasa de transmisión. Concretamente, la tasa de transmisión aumenta en un factor 1/n. Por ejemplo, si escogemos n = 3, entonces tenemos un aumento de la tasa de transmisión de un 33%. Otra desventaja de esta técnica es que

aumenta el tiempo de playout, ya que el receptor tiene que esperar a recibir todos los paquetes de un grupo antes de iniciar la reproducción del primer paquete del grupo.



Un segundo mecanismo FEC, utilizado principalmente en aplicaciones de audio, es el de enviar como información redundante un flujo audio de más baja resolución. Por ejemplo, se podría enviar como flujo nominal una codificación PCM de 64 Kbps y como flujo redundante de más baja calidad una codificación GSM de 13 Kbps. Esta técnica no es apropiada para vídeo puesto que, a diferencia del audio, en vídeo los cambios de resolución repercuten negativamente en la calidad resultante. De entre las herramientas de audio que utilizan esta técnica encontramos Free Phone (http://www-sop.inria.fr/rodeo/fphone/) y Robust Audio Tool (RAT) (http://www-mice.cs.ucl.ac.uk/multimedia/software/rat/).

El funcionamiento de esta técnica se ilustra en la siguiente figura. El transmisor construye el paquete *n*-ésimo juntando la secuencia *n*-ésima del flujo nominal y la (*n*-1)-ésima del flujo redundante. De esta forma, cuando se produce la pérdida de paquetes no consecutivos, el receptor puede disimular las pérdidas reproduciendo la secuencia del flujo redundante que llega en el siguiente paquete. Evidentemente, la secuencia correspondiente al flujo redundante produce una calidad inferior a la del flujo nominal. De todas formas, un flujo formado principalmente por secuencias de alta calidad, ocasionalmente secuencias de baja calidad, y sin secuencias perdidas, proporciona una calidad global buena.



Con esta técnica, el receptor únicamente tiene que esperar 2 paquetes antes de empezar la reproducción, por lo que el aumento del tiempo de playout es pequeño. Además, como la tasa de envío del flujo redundante es mucho menor que la del flujo nominal, el incremento de la tasa de envío con esta técnica es pequeño.

Para poder afrontar pérdidas consecutivas (las cuales son comunes en Internet), se puede utilizar una simple variación de la técnica descrita anteriormente. En lugar de incluir únicamente la secuencia (*n*-1)-ésima del flujo redundante en el paquete *n*-ésimo, podemos incluir la (*n*-1)-ésima y la (*n*-2)-ésima. De esta forma, aumentamos la robustez de la transmisión y garantizamos un rendimiento aceptable en entornos hostiles, aunque el precio a pagar es un aumento tanto del caudal enviado como del tiempo de playout.

Una tercera técnica FEC es aquella que añade distintos niveles de redundancia a distintas partes de la información dependiendo de su importancia, de tal forma que se protege más a aquella información más redundante. Esta técnica se llama Priority Encoding Transmission (PET). Resulta adecuada para vídeo, ya que la mayoría de codificadores de vídeo (a diferencia de los de audio) establecen distintas prioridades en la información que envían. El ejemplo que emplearemos para explicar esta técnica es el codificador MPEG, aunque la técnica puede utilizarse también para otros codificadores de vídeo.

Con MPEG se diferencian diferentes tipos de trama. Las tramas I son las más importantes puesto que se codifican como una imagen independiente y no dependen de ninguna otra trama. Las tramas P se producen a partir de la trama I o la trama P anterior más cercana, y finalmente las tramas B se interpolan a partir de las dos tramas P e I más cercanas. Por lo tanto, lo más perjudicial en un flujo MPEG es perder una trama I, puesto que se pierde la

información correspondiente a esta trama y toda la de las trama P y B referidas a ella. De la misma forma, perder una trama P es más perjudicial que perder una trama B.

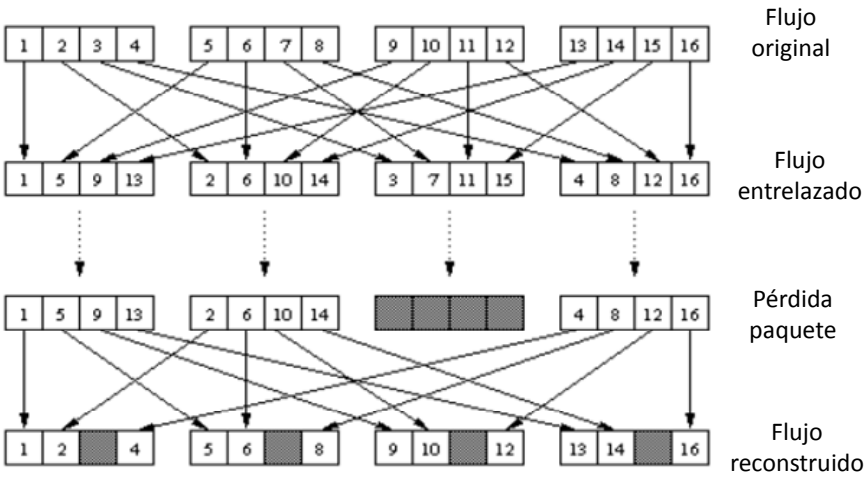
Priority Encoding Transmission (PET) funciona como se explica a continuación. Cada trama I, P y B se divide en bloques de un tamaño que depende de la importancia de la trama: cuanto más importante la trama, menores los bloques. Por ejemplo, se pueden tomar para las tramas I bloques de 60 palabras, para las tramas P bloques de 80 palabras y para las tramas B bloques de 95 palabras. A partir de aquí, se construyen un cierto número de paquetes (por ejemplo 100 paquetes) de tal forma que **una palabra en cada paquete** se dedica a uno de los bloques anteriores.

Con el procedimiento descrito, para un bloque I de 60 palabras estamos enviando 100 palabras independientes, mientras que para un bloque B de 95 palabras estamos mandando también 100 palabras. De esta forma, estamos enviando más redundancia para las tramas I y por lo tanto la probabilidad de perder la información correspondiente será menor. Concretamente, utilizando los algoritmos adecuados, siempre que no perdamos más de 40 paquetes del total de 100 (es decir, recibamos un mínimo de 60), podremos reconstruir las 60 palabras de un bloque I. Lo mismo aplica para los bloques P y B, para los que podemos perder un máximo de 20 y 5 de los 100 paquetes, respectivamente.

Entrelazado (Interleaving)

Una técnica para audio alternativa o complementaria al envío de información redundante (FEC) es la llamada interleaving o entrelazado. Con esta técnica, ilustrada en la siguiente figura, el transmisor reorganiza las secuencias de audio antes de proceder a su transmisión, de tal forma que las secuencias de audio adyacentes se encuentran separadas en el flujo de audio transmitido. De esta forma, con esta técnica se reduce el impacto de las pérdidas.

Por ejemplo, si el audio se divide en secuencias de 5 ms, y cada paquete contiene un total de 20 ms de audio, estas unidades se pueden reorganizar en los paquetes transmitidos tal y como se muestra en la figura, de tal forma que si se pierde un paquete, esto repercute en múltiples huecos de 5 ms, en lugar de un hueco mayor de 20 ms. El hecho de tener varios huecos pequeños en lugar de uno grande aumenta considerablemente la calidad de audio percibida, lo que hace que esta técnica sea efectiva.



La principal ventaja de la técnica de interleaving es que no aumenta el caudal utilizado por la aplicación. La desventaja es el retardo adicional que introduce. Esto limita su aplicabilidad para las aplicaciones interactivas y hace que esta técnica sea especialmente adecuada para aplicaciones de tipo streaming.

Reconstrucción de secuencias dañadas

Para el caso de las señales de audio, existen técnicas que permiten ocultar y limitar el impacto producido por la falta de parte de la secuencia a reproducir. Estas técnicas se basan en el hecho que las secuencias de audio exhiben una fuerte correlación cuando los intervalos de audio observados son cercanos en el tiempo. Esto hace que estas técnicas sean efectivas siempre que las pérdidas no superen un umbral de alrededor del 15 % y los paquetes sean suficientemente pequeños (de entre 5 y 40 msec de audio). Para secuencias de audio más largas, estás técnicas dejan de ser efectivas, ya que pueden llevar a la pérdida de fonemas enteros. Esto hace que la aplicación de estas técnicas sea especialmente efectiva cuando su uso se combina con la técnica de entrelazado explicada anteriormente.

La forma más simple de reconstruir una secuencia de audio en la que faltan partes de la secuencia es la llamada *packet repetition* (repetición de paquetes). Esta técnica se basa en suplantar los paquetes perdidos por copias exactas de los paquetes inmediatamente anteriores. Esto produce un rendimiento razonablemente bueno y tiene la ventaja que el coste computacional es muy bajo.

Otra técnica más elaborada es la llamada interpolación. Con esta técnica se utilizan las secuencias de audio anterior y posterior a la secuencia perdida para construir una

secuencia que suplante a la perdida. Esta técnica proporciona un rendimiento algo mejor pero tiene la desventaja que requiere mayores recursos computacionales para realizar las operaciones de interpolación.

3.5. Técnicas de adaptación de la tasa de envío

En caso de que las pérdidas no sean puntuales sino que sean persistentes, causadas por un elevado nivel de congestión, las técnicas vistas en el capítulo anterior no pueden compensarlas, y se tendrán que aplicar mecanismos de reducción de la tasa de envío de tal forma que se reduzca la congestión y consecuentemente las pérdidas. Para ello, necesitaremos de aplicaciones multimedia que sean capaces de regular la tasa de envío en base al nivel de congestión detectado. Puesto que los algoritmos de regulación de la tasa de envío afectan a la distribución del caudal de la red entre las diferentes aplicaciones que la usen, el diseño de estos algoritmos tendrá que tener en cuenta cual es el criterio en base al que se quiere distribuir los recursos de red. En este capítulo abordamos estos tres temas:

- Codificadores adaptativos
- Criterios de compartición de caudal
- Algoritmos de adaptación de la tasa de envío

Codificadores Adaptativos

El primer requisito que tenemos que cumplir para poder adaptar el caudal enviado a la red en base a un cierto criterio, es el de disponer de codificadores de audio y vídeo que sean capaces de fijar la tasa de información producida en un valor deseado.

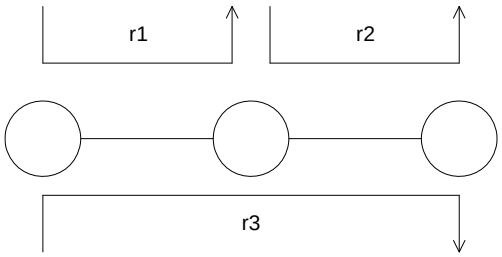
Existen diferentes codificadores de audio y video adaptativos. A continuación vamos a proceder a explicar, a modo de ejemplo, el funcionamiento del codificador lH.261. Este codificador extiende el codificador estándar H.261 con la inclusión de niveles o capas, de tal forma que seleccionando el número de niveles o capas enviados, se puede adaptar la tasa de transmisión.

H.261 transforma cada imagen al dominio frecuencial aplicando la transformada DCT a cada bloque de 8x8 pixels. Los coeficientes resultantes se ordenan y se envían como un flujo.

El codificador lH.261 modifica el funcionamiento anterior en que los coeficientes se estructuran en capas en función de su rango. El primer coeficiente corresponde a la señal continua (información básica) y cada coeficiente representa un refinamiento del coeficiente anterior. Dependiendo del número de capas que se envían, se obtiene por lo tanto una mejor calidad al precio de una de tasa de envío superior. Se disponen de tantas capas como coeficientes, es decir 64, y cada capa corresponde a una tasa de envío distinta.

Criterio de compartición del caudal

A la hora de regular la tasa de envío de las diferentes aplicaciones que usan la red, es importante hacerlo de tal forma que la distribución del caudal resultante sea óptima. A continuación vamos a estudiar el criterio de compartición de caudal en el que se basa la red Internet en base al siguiente ejemplo. En este ejemplo, tenemos un total de 3 aplicaciones que utilizan una red de 2 enlaces de tal forma que los flujos de dos de las aplicaciones atraviesan únicamente un enlace, mientras que el flujo de la tercera aplicación atraviesa ambos enlaces. Cada enlace tiene un caudal físico normalizado a la unidad.



Un posible criterio de distribución de caudal en el anterior ejemplo sería el asignar a todas las aplicaciones el mismo caudal:

$$r1 = r2 = r3 = 1/2$$

El criterio anterior corresponde al de máxima justicia (todas las aplicaciones reciben el mismo caudal) pero no maximiza el caudal total. Efectivamente, r1 + r2 + r3 = 3/2, mientras que con la siguiente distribución

$$r1 = r2 = 1$$
$$r3 = 0$$

el caudal total es 2 y por lo tanto superior.

Los dos criterios anteriores corresponden a los casos extremos de maximizar la justicia y maximizar el rendimiento global, respectivamente. El criterio de distribución que se ha

adoptado en Internet representa un caso intermedio entre estos dos extremos y se llama justicia proporcional. Una distribución proporcionalmente justa se define como aquella que, para cualquier otra distribución, la suma de cambios proporcionales es negativo:

$$\sum_i \frac{r_i^* - r_i}{r_i} \leq 0$$

Con la definición, únicamente reduciremos el caudal de una estación por ejemplo en un 10% si esto permite aumentar el caudal de otra estación en más de un 10%.

Si consideramos una perturbación alrededor de la distribución proporcionalmente justa

$$r_i \rightarrow r_i + dr_i$$

la anterior ecuación nos lleva a

$$\sum_i \frac{dr_i}{r_i} \leq 0$$

lo cual se puede reescribir como

$$\sum_i (log(r_i))' dr_i \leq 0$$

lo que lleva a que la distribución proporcionalmente justa es aquella que maximiza la suma de logaritmos

$$\sum_i log(r_i)$$

Aplicando esto al ejemplo anterior:

(log(r3)+log(1-r3)+log(1-r3))' = 0

1/r3 - 1/(1-r3) - 1/(1-r3) = 0

r3 = 1/3

r1 = r2 = 2/3

lo cual efectivamente representa un punto intermedio entre los dos extremos vistos anteriormente.

En el ejemplo anterior hemos visto que el caudal que obtiene una aplicación es inversamente proporcional al número de enlaces que atraviesa. Se puede ver que este es un criterio general que proporciona una distribución parecida a la justicia proporcional. Si asumimos que el número de enlaces atravesados es proporcional al Round Trip Time (RTT), haciendo que el caudal sea inversamente proporcional al RTT se garantiza que el caudal en la red se va a distribuir aproximadamente de acuerdo con el criterio deseado.

Tal y como vimos en el tema 1, el caudal obtenido con TCP sigue el criterio anterior. Para garantizar justicia entre las aplicaciones multimedia y las de datos, el criterio que se sigue en Internet es que las aplicaciones multimedia (que usan UDP) y las aplicaciones de datos (que usan TCP) consuman el mismo caudal. Esto asegura una repartición justa entre las distintas aplicaciones y además garantiza que el criterio seguido será (de forma aproximada) el de justicia proporcional. A continuación estudiaremos como pueden las aplicaciones multimedia basadas en UDP adaptar su tasa de envío para que el consumo de caudal sea el mismo que si usaran TCP. Estas aplicaciones se llaman *TCP-friendly*.

Algoritmo TCP-friendly de adaptación de la tasa de envío

A continuación describimos uno de los algoritmos TCP-friendly de adaptación de la tasa de envío propuestos en la literatura. El protocolo se llama MSTFP (Multimedia Streaming TCP-Friendly Protocol).

El protocolo MSTFP se basa en conocidos estudios del caudal de TCP según los cuales el caudal obtenido por una conexión TCP se ajusta a la siguiente fórmula:

$$\text{rcvrate} = \frac{C}{\text{RTT} \times \sqrt{p_L}}$$

donde C es una constante igual a 1.22, RTT es el Round Trip Time y pL es la probabilidad de pérdida.

MSTFP se basa en estimar los valores del RTT y pL y ajustar la tasa de envío (por ejemplo con el codificador de vídeo visto anteriormente, el número de capas enviado) al valor resultante de la fórmula anterior (o al valor más próximo posible en el caso de vídeo por capas).

La estimación de la tasa de pérdidas se puede realizar de diferentes formas. La manera estándar de hacerlo es utilizando las estadísticas proporcionadas por el protocolo RTCP, con lo que no se requieren algoritmos adicionales.

La estimación del RTT se realiza en base a paquetes de feedback que el receptor envía a intervalos regulares. En los paquetes de datos se añade una referencia temporal (ST1) y en los paquetes de feedback se incluye el tiempo transcurrido desde que se ha recibido el paquete hasta que se ha enviado el feedback correspondiente, con lo que la estimación del RTT se hace de acuerdo con la siguiente fórmula:

$$\text{RTT} = \alpha \times \overline{\text{RTT}} + (1 - \alpha) \times (\text{now} - ST1 - \Delta RT)$$

En base a estos datos el transmisor puede calcular la tasa de emisión deseada y ajustarla correspondientemente. Puesto que cambios continuos en la resolución de un video producen un efecto negativo en la calidad del mismo, el período de ajuste de la tasa de envío debe de seleccionarse cuidadosamente. De esta forma, si bien en media el protocolo TCP y el MSTFP obtienen el mismo caudal, las oscilaciones del primero serán muy superiores a las del segundo.

Bibliografía

[1] J. F. Kurose, K. W. Ross, "Computer Networking: A Top-Down Approach Featuring the Internet", Addison Wesley.

[2] http://www.cs.columbia.edu/~hgs/rtp/

[3] Real Time Streaming Protocol (RSP), IETF RFC 2326, disponible en http://www.ietf.org/

[4] R. Ramjee, J. Kurose, D. Towsley, H. Schulzrinne, "Adaptive Playout Mechanisms for Packetized Audio Applications in Wide-Area Networks", disponible en Aula Global.

[5] Colin Perkins, Orion Hodson, Vicky Hardman, "A Survey of Packet Loss Recovery Techniques for Streaming Audio", disponible en Aula Global.