Modelo de Aplicación de Sesión Multimedia

Proyecto Fin de Carrera

Ingeniería Informática

Escuela Técnica Superior de Ingeniería Informática
Universidad de Sevilla

Federico Montesino Pouzols

Tutores: Diego R. López[1] y Manuel Valencia[2]

13 de Septiembre de 2002

[1]RedIRIS Red Española de I+D
[2]Departamento de Tecnología Electrónica, Universidad de Sevilla

Copyright © 2001, 2002 Federico Montesino Pouzols.

Índice general

Modelo de Aplicación de Sesión Multimedia

Federico Montesino Pouzols

Tutores: Diego R. López y Manuel Valencia

13 de Septiembre de 2002

Memoria de Proyecto Fin de Carrera de la *Escuela Técnica Superior de Ingeniería Informática* de la *Universidad de Sevilla*.

Capítulo 1

Introducción

El deseo de conseguir procedimientos que permitan la comunicación a distancia de forma eficiente y unívoca es muy anterior a la aparición de los sistemas de telecomunicaciones modernos y de los ordenadores, y su relevancia se puede comprobar en algunos de los primeros escritos de la humanidad [66].

El desarrollo de Internet durante las últimas décadas ha supuesto un considerable avance en la materia. Por otra parte, la capacidad de procesamiento y almacenamiento de los ordenadores actuales es suficiente para el intercambio de información multimedia en tiempo real, función no contemplada en el diseño inicial de Internet.

Actualmente se desarrollan tecnologías, protocolos y arquitecturas con las que se pretende sentar las bases necesarias para que que la comunicación multimedia en tiempo real a través de Internet sea tan accesible como ya lo es la comunicación de texto y datos. Además, se pretende que estos nuevos sistemas puedan interoperar con el sistema de conferencias en tiempo real más extendido: la red de telefonía, ya sea fija o móvil.

Para lograr un sistema de comunicación universal en el que tengan cabida contenidos tan diversos como el correo-e, vídeo, voz o FAX, entre otros muchos, son necesarios avances en los tres campos [35] siguientes:

- *Tecnologías de red*, tales como transmisión multicast, calidad de servicio o formatos de codificación y compresión de información multimedia.
- *Middleware* o software de base que permite desarrollar más fácilmente y con mayor efectividad aplicaciones, como pueden ser bibliotecas base de aplicaciones de red o pilas de protocolos de transporte y control de sesiones multimedia.
- Definición y desarrollo de *aplicaciones y servicios* en torno a la información multimedia: aplicaciones de sonido, vídeo, edición compartida y distribuida, vídeo bajo demanda, etc.

Este proyecto se enmarca dentro del segundo campo y, más concretamente, en el modelo que para los sistemas de comunicación multimedia en tiempo real se viene desarrollando en el IETF [64]. Este modelo se ha diseñado para sesiones multimedia con múltiples participantes en las que se utilizan múltiples medios y formatos de datos intercambiados mediante redes y equipamiento heterogéneos. Se caracteriza por ser abierto, flexible, modular, escalable y adaptable a la heterogeneidad de Internet.

El modelo que se va a seguir se comenzó a desarrollar a principios de los 90, con los primeros protocolos y aplicaciones experimentales de conferencia multimedia en Internet (aplicaciones de Mbone[35, parte 3] y [180]), que han terminado dando lugar a un conjunto sólido de protocolos de transporte en tiempo real y control de sesiones que ya permite la realización de aplicaciones como los teléfonos IP.

A pesar de la solidez y madurez del modelo comentado, en la actualidad no existen implementaciones –ya sean libres o propietarias– que lo lleven a la práctica, según nuestra opinión, con el suficiente grado de conformidad con los estándares, modularidad y flexibilidad deseables en un sistema que debe servir de base a una nueva generación de aplicaciones.

Por ello, el objetivo de este proyecto es avanzar en el desarrollo de una implementación libre de los protocolos de nivel de transporte y aplicación definidos en la arquitectura de sistemas multimedia del IETF. Se hará especial énfasis en el desarrollo de una implementación libre del protocolo de transporte en tiempo real RTP, ya que consideramos que los principios de diseño que se siguieron para la definición de este protocolo no se han llevado a la práctica satisfactoriamente hasta la fecha.

En el capítulo 2 se tratará el protocolo de transporte desarrollado por el IETF y adoptado asimismo por la ITU-T para sus recomendaciones: RTP, estudiándose en primer lugar los elementos básicos de transporte sobre los que se asienta y las características de las aplicaciones a las que debe dar soporte.

El capítulo 3 estudiará las arquitecturas de control de sesiones multimedia del IETF (con especial atención al protocolo SIP) y, en menor detalle, de la ITU-T. Asimismo, se analizarán los protocolos de descripción de sesiones multimedia y negociación de capacidades, especialmente el más extendido en la actualidad: SDP.

Los capítulos 4 y 5 describen el diseño e implementación del sistema en el cual se integran los componentes implementados durante este proyecto: GNU ccRTP y FreeSDP, implementaciones de los protocolos RTP y SDP, respectivamente. Junto con ambas bibliotecas, se describirá GNU Common C++, base del sistema y a cuyo desarrollo se ha contribuido, así como GNU oSIP.

Capítulo 2

Transporte en Tiempo Real

En general, los protocolos de las redes de computadores se estudian por niveles [55, 178, 163], a pesar de que la aplicación de una jerarquía estricta al desarrollo de protocolos es una estrategia de diseño con claros inconvenientes [30, 83], especialmente en los niveles de transporte y superiores[1]. De entre los diferentes modelos existentes, en esta memoria se sigue el modelo TCP/IP, estructurado en cuatro capas: enlace, red, transporte y aplicación. Este capítulo trata el transporte de datos en tiempo real en redes IP, entendiéndose por transporte el nivel tres del modelo TCP/IP, cuya función fundamental es proporcionar un servicio de entrega de datos de extremo a extremo.

En las siguientes secciones se analiza por qué en la actualidad las aplicaciones multimedia para Internet utilizan mayoritariamente UDP como protocolo básico de transporte en lugar de TCP, los problemas derivados y las soluciones planteadas. Posteriormente se caracteriza temporalmente el transporte en tiempo real, presentándose por último el protocolo RTP.

2.1. Alternativas para Transporte de Datos y Señalización

Si diez o quince años atrás el uso de TCP estaba significativamente más limitado que en la actualidad como consecuencia de las bajas prestaciones derivadas de los entonces escasos y rudimentarios mecanismos de control de congestión , en la actualidad, si ignorásemos la existencia de aplicaciones multimedia en Internet, se podría afirmar que el protocolo UDP ha quedado relegado a un papel secundario, siendo minoría las aplicaciones que se basan el él.

Consideremos como caso ilustrativo la evolución del sistema NFS, cuya primera versión [176], de mediados de los años ochenta, define UDP como único protocolo de transporte. La decisión vino motivada por las sensiblemente menores prestaciones en cuando a tasa de transferencia de grandes volúmenes de datos que ofrecía TCP, junto con la preponderancia de las redes locales sobre las de área extensa.

La versión 3 de NFS [22], sin embargo, concedía mayor importancia a TCP, y en efecto a partir de entonces las realizaciones empezaron a utilizar tanto UDP como TCP. Esta modificación vino motivada por la evolución de TCP, que durante la primera mitad de la década de los noventa, gracias a las mejoras en control de congestión , sobrepasó a UDP en tasa de transferencia de grandes volúmenes de datos, dado que los nuevos mecanismos de control de congestión de TCP eran superiores a los

[1] Este problema se tratará en el capítulo 4.

simples ajustes utilizados en realizaciones basadas en UDP. La desventaja de TCP frente a UDP, que lo había relegado a un puesto secundario, había desaparecido, primando ahora la mayor capacidad de TCP para adaptarse a redes de área extensa.

Finalmente, la versión 4 de NFS [156], que, a diferencia de las anteriores, ha sido desarrollada por un grupo de trabajo siguiendo los procedimientos habituales del IETF, da clara preferencia a TCP sobre UDP –que queda prácticamente descartado[2]–, estableciendo como obligatorio salvo en circunstancias especiales su uso en todas aquellas realizaciones que funcionen en plataformas con pila TCP. Asimismo, pasa a ser obligatoria en todo caso la existencia de mecanismos de control de congestión (véase la sección 2.2.3) comparables con los de TCP [2] en el protocolo de transporte subyacente. Compárese esta evolución con la que se describe en la sección 2.3.6 para el protocolo RTP, donde se retomará el tema del diseño de protocolos compatibles con TCP en cuanto al uso de ancho de banda.

Del ejemplo anterior, se podría extraer como conclusión que la fiabilidad y control de congestión ofrecidos por TCP lo hacen ser un protocolo de elección preferente frente a UDP, quedando este último como solución para casos muy específicos[3] No obstante, la mayoría de las aplicaciones multimedia –con la notable excepción de algunas muy extendidas en Internet, basadas en RTSP y TCP– no utilizan TCP como protocolo de transporte, sino UDP junto con algunos mecanismos de control de nivel de aplicación.

Algunas razones que pueden llevar a basar un servicio en UDP en lugar de TCP se discuten en [170, capítulo 20]; a continuación se presentan dos razones básicas en el ámbito de este proyecto:

- Las aplicaciones que requieren funciones de broadcast o multicast deben usar UDP, puesto que el diseño de TCP no las contempla[4]. Tal es el caso de las aplicaciones de Mbone [180], cuyo desarrollo está estrechamente ligado al del protocolo RTP.

- Usando UDP es posible aprovechar mejor los recursos y realizar transacciones de forma más eficiente. En primer lugar, se elimina el retraso de establecimiento de conexión. Un intercambio de petición y respuesta que con UDP se realiza por medio de dos segmentos requeriría alrededor de diez segmentos TCP. En segundo lugar, TCP duplica la aportación del intervalo de ida y vuelta entre transmisor y receptor al tiempo total de transacción, si bien este problema se puede mitigar con T/TCP ([16, 17], tratado en [167]).

Sin embargo, al elegir UDP en lugar de TCP se prescinde de funciones tan importantes como:

1. Detección de paquetes duplicados, retransmisión de paquetes perdidos, confirmación afirmativa y reordenamiento de paquetes desordenados al transitar por la red.

2. Control de flujo con mecanismo de ventana deslizante, arranque progresivo, retroceso ante signos de congestión, y otros algoritmos de control de congestión [2] empleados habitualmente en las realizaciones de TCP actuales.

[2] De hecho, UDP no aparece nombrado en la referencia dada.

[3] Como pueden ser los protocolos DHCP [42], BOOTP [34] o TFTP [161], pensados para redes locales, y en los que la simplicidad es clave. El caso de mayor relevancia es el DNS [104, 105], sistema para el que es más conveniente el intercambio de datagramas UDP, en especial en las búsquedas iterativas.

[4] El desarrollo de protocolos multicast fiables [35, 98] es un problema complejo que queda fuera del alcance de este proyecto.

No obstante, la solución dada por TCP a estos problemas no es adecuada al transporte en tiempo real, puesto que:

1. El comportamiento fiable de TCP, más que una solución, es un problema adicional desde el punto de vista del cumplimiento de las restricciones temporales.

2. En general, las aplicaciones multimedia no pueden adaptar de forma flexible la tasa de transferencia en función del estado de la red, sino que admiten a lo sumo la selección de unas pocas tasas de transferencia fijas (por ejemplo, en una sesión multimedia, el vídeo no se puede transmitir a cualquier tasa de transferencia).

Considérense los problemas que puede plantear el uso de TCP para una aplicación multimedia en tiempo real. En especial, en aplicaciones interactivas el retraso admisible de extremo a extremo –por ejemplo, desde que un micrófono registra un sonido en un transmisor hasta que un altavoz lo reproduce en un receptor– es muy pequeño y a menudo significativamente menor que el tiempo necesario para una retransmisión TCP. Dado que el nivel TCP sólo entrega paquetes siguiendo la secuencia estricta de transmisión, para obtener un segmento que en principio se ha perdido se ha de esperar su retransmisión. En esta situación, el paquete perdido inicialmente no es el único inutilizable, sino además todos aquellos que el nivel TCP no entrega hasta completar la retransmisión[5].

Es decir, TCP no tiene en cuenta ninguna restricción temporal, por lo que en ningún caso se admite una pérdida ocasional, aunque ello implique mayor retardo en la recepción. Por tanto, las carencias del protocolo UDP han de ser compensadas añadiendo mecanismos de control de errores y congestión adaptables a las necesidades concretas de cada aplicación de tiempo real. La solución desarrollada por el IETF es el protocolo de transporte en tiempo real RTP.

Como se describirá más adelante, las implementaciones actuales de RTP se basan en la práctica en el protocolo UDP, aunque RTP es independiente del protocolo subyacente. Debido a las carencias de UDP, durante los últimos años se ha puesto de manifiesto que, para extender ampliamente el uso de RTP, la incorporación de mecanismos de control de congestión equivalentes a los de TCP es imprescindible (lo que se tratará en mayor detalle en las secciones 2.2.3 y 2.3.5). La introducción de mecanismos de control de congestión en sistemas basados en RTP conlleva complejidad adicional en la pila RTP de las aplicaciones. Asimismo, la gestión de errores que llevan a cabo los protocolos tradicionales entra en conflicto a menudo con los intereses de las aplicaciones de tiempo real. Por este motivo, se han desarrollado varios protocolos de transporte [124] diseñados para dar una base más sólida y adecuada a los servicios de transporte en tiempo real, tanto para transporte de datos como para transporte de señalización. Estos protocolos, en especial SCTP, UDP Lite y DCCP, pueden coexistir e incluso sustituir en un futuro próximo a UDP como protocolo de transporte base para los sistemas en tiempo real en Internet.

El protocolo SCTP [171], aunque diseñado para el transporte de los mensajes de señalización de la red telefónica tradicional (mensajes SS7) sobre redes IP, es válido para un rango mucho mayor de aplicaciones [31]. SCTP proporciona, entre otras, las siguientes funciones: transferencia de datos fiable, definición de múltiples flujos para una misma conexión, posibilidad de seleccionar entrega ordenada o desordenada por separado para cada flujo, monitorización, fragmentación y reunificación de datos, y control de congestión y flujo compatibles con TCP. Asimismo, contempla funciones adicionales de seguridad, como protección contra ataques por denegación de servicio o por enmascaramiento.

[5]En el capítulo 4 se comentará el efecto favorable que tiene en ciertas aplicaciones la posibilidad de recibir los segmentos en el nivel de aplicación independientemente de si llegan en orden.

Por su diseño, es muy adecuado para protocolos de señalización de sistemas de tiempo real, como SIP (tratado en la sección 3.4). Es especialmente útil en los casos en que las entidades SIP intercambian un gran volumen de mensajes, como ocurren en las pasarelas y proxys SIP. El estudio de su uso como protocolo de transporte para SIP [132] ha puesto de manifiesto algunas de sus posibles ventajas respecto a UDP en el contexto de los sistemas en tiempo real, derivadas fundamentalmente del hecho de que SCTP es un protocolo orientado a la conexión:

- Proporciona mecanismos para retransmitir con rapidez segmentos perdidos y control de congestión. Con esto, se mejora sensiblemente el tiempo necesario para retransmitir paquetes perdidos –con SIP basado en UDP se necesita al menos medio segundo para detectar una pérdida– y permite controlar la tasa de transmisión de mensajes entre dos entidades SIP.
- Fragmentación de nivel de transporte. En aquellas situaciones en las que la MTU es menor que el tamaño de un mensaje SIP, no se recurre a fragmentación IP, mejorando la tasa de pérdidas y el comportamiento ante cortafuegos y traductores de direcciones de red.

Nótese que estas ventajas frente a UDP también las presenta TCP. Por otro lado, las principales ventajas de SCTP frente a TCP son:

- Al estar basado en mensajes, y no en flujos como TCP, el uso del servicio sin orden permite procesar mensajes tan pronto como llegan, sin necesidad de esperar a otros, lo que da lugar a un significativo aumento de prestaciones cuando varias conexiones SIP van multiplexadas en una única conexión TCP, aunque las prestaciones siguen siendo menores que con UDP.
- Una conexión SCTP se puede asociar con múltiples direcciones de red; si falla una dirección, los datos pasan a otra, mejorando así la tolerancia a fallos. Sin embargo, este mismo comportamiento se puede conseguir con SIP sobre TCP mediante el uso del DNS [131].

La conclusión general que se ha obtenido es que SCTP proporciona mejores prestaciones cuando se producen pérdidas. En otros casos, las prestaciones son similares a las de TCP[6].

UDP Lite [91, 92] se ha diseñado pensando en aplicaciones para dispositivos móviles inalámbricos, en las que, respecto a la mayoría de las aplicaciones de Internet, existe una mayor frecuencia de errores. Es un protocolo de transporte que difiere del UDP tradicional únicamente por la sustitución del campo longitud de la cabecera UDP por un nuevo campo que indica el número de octetos sobre los que se debe aplicar el algoritmo de comprobación de validez. Es decir, la comprobación es parcial, de este modo los segmentos quedan divididos en una parte sometida a verificaciones (generalmente las cabeceras de los distintos niveles) y una parte en la que se aceptan errores (datos de la aplicación). Si el valor del rango de comprobación es igual al de la longitud del segmento, UDP Lite no difiere de UDP; si el valor es menor, se admiten errores en la parte del segmento no cubierta por el algoritmo de comprobación[7].

[6] La comprobación experimental del porcentaje de pérdidas a partir del cual la diferencia es relevante está aún por realizar.

[7] Nótese que, aunque podría pensarse que los errores se detectarían en el nivel de enlace, se ha comprobado [91] que muchos errores no se generan en los enlaces, sino en los nodos. Además, algunos enlaces –particularmente los inalámbricos– no realizan comprobaciones. No obstante, se reconoce la conveniencia de modificar los gestores de dispositivo de algunos nodos de modo que el nivel de enlace no realice comprobaciones sobre los segmentos UDP Lite, al menos en puntos estratégicos de la red.

Esta modificación permite que las aplicaciones multimedia reciban el mayor número posible de los segmentos que componen los flujos de sonido o vídeo. De este modo, el impacto negativo en la calidad de reproducción en los receptores se reduce drásticamente para muchos formatos de codificación. Algunos experimentos [91] han demostrado que el número de paquetes descartados en los entornos con conexiones inalámbricas se puede reducir hasta un 40 %.

Cabe destacar que el cambio en la estrategia de tratamiento de errores que el uso de UDP Lite introduce en niveles inferiores al de transporte contradice la definición tradicional de las funciones de estos niveles y está directamente relacionado con los principios de segmentación de nivel de aplicación y procesamiento integrado de niveles, que se presentarán en el capítulo 4.

Finalmente, DCCP es un protocolo aún en fase inicial de desarrollo con el que pretende proporcionar dos funciones: control de congestión y establecimiento, mantenimiento y terminación de flujos de paquetes con transporte no fiable.

2.2. Transporte en Tiempo Real

A grandes rasgos, la característica que diferencia a un sistema en tiempo real frente a otros sistemas es que, además de corrección lógica, debe presentar *corrección temporal*. Es decir, no sólo se consideran perdidos los datos que no llegan al receptor, sino también los que llegan tarde. Los sistemas de transporte en tiempo real objeto de este proyecto se pueden catalogar como flexibles, en contraposición a los sistemas destinados a control de procesos, que requieren soporte especifico por parte de los niveles inferiores.

Si al hecho de que los sistemas operativos generalmente empleados para dar soporte a las aplicaciones multimedia –sistemas de propósito general– no son adecuados para aplicaciones de tiempo real unimos que la infraestructura de red habitualmente utilizada es asimismo inadecuada (ninguno de ellos proporciona determinismo temporal), llegamos a la conclusión de que la definición de tiempo real en los sistemas tratados ha de ser muy vaga, del tipo *"un sistema que introduce el menor número posible de retrasos apreciables o incómodos para quienes lo utilizan"*.

En estos sistemas existe un cierto nivel de pérdidas tolerables que depende del tipo de datos que se transporta. En algunos casos recogidos en [94], se puede transmitir voz de baja calidad con pérdidas de hasta un 50 %. Sin embargo, para sonido de alta calidad, el máximo admisible está alrededor del 10 %, para voz, y el 5 %, para música. Por el contrario, transportar vídeo correctamente requiere reducir el porcentaje de pérdidas por debajo del 1 %. Para algunos formatos de codificación complejos y poco adecuados para transmisión por redes, como MPEG I y II, estos porcentajes se pueden ver reducidos drásticamente. Del mismo modo, algunos formatos (como Ogg Vorbis[106]) o formatos junto con esquemas de empaquetado en RTP (como MPEG IV sobre RTP[86]) pueden llegar a tolerar mayores porcentajes de pérdidas, al menos en parte de los datos transmitidos.

Desde el punto de vista cuantitativo, las restricciones temporales sobre el sistema dependen fundamentalmente de si éste debe ser interactivo o no, además del tipo de información que se ha de transmitir. en sistemas interactivos –por ejemplo telefonía– se puede aceptar un retraso de extremo a extremo de alrededor de 40 milisegundos en transmisión de sonido sin eliminación de eco [128]. Si se aplican algoritmos de eliminación de eco, el retraso puede aumentar a unos 150 milisegundos. Por otra parte, en sistemas de reproducción de conferencias pregrabadas, se pueden admitir retrasos de alrededor de medio segundo, obteniéndose una respuesta de tipo reproductor de cassette. No obstante, en ambos casos se requiere limitar la variación del retraso.

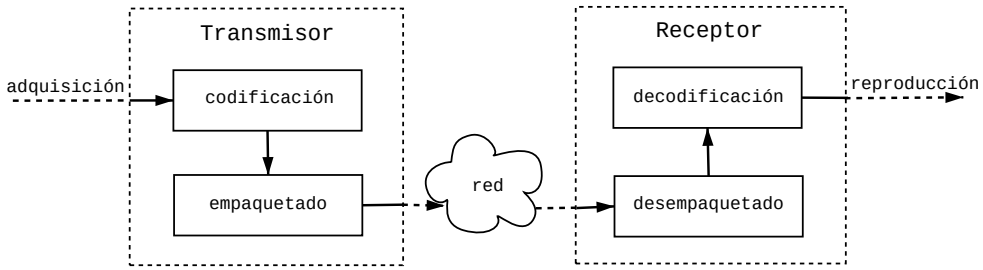


Figura 2.1: Etapas de un sistema de transporte en tiempo real

En la actualidad, las conexiones físicas más comunes, *fast Ethernet*, son suficientemente rápidas y fiables como para transportar datos en tiempo real. Por tanto, el objetivo de los últimos desarrollos relativos a transporte en tiempo real no es la velocidad o fiabilidad, sino afinar la sincronización entre los equipos conectados en red (idealmente hasta el grado de precisión necesario para distinguir el periodo de tiempo correspondiente a cada muestra de sonido o imagen –del orden de 48KHz). Sin embargo, es inevitable que aparezcan errores de sincronización del orden de unos pocos microsegundos incluso en redes de área local, del mismo modo que en un auditorio el sonido se desplaza alrededor de $1{,}5\ mm$ cada $5{,}2\ \mu s$. Son problemas que quedan fuera del alcance de los protocolos de transporte.

2.2.1. Etapas de transporte

Para presentar los componentes de un sistema de transporte en tiempo real, consideraremos el caso de un sistema de retransmisión de conferencias en formato Ogg Vorbis de sonido [106, 193, 194]. Siguiendo el esquema mostrado en la figura 2.1, desde que el sonido se registra en un micrófono en la sala de conferencias hasta que un asistente remoto lo escucha, ha de ser adquirido, codificado, empaquetado, transportado de extremo a extremo, desempaquetado, decodificado y, finalmente, reproducido.

El proceso descrito se repite para cada una de las unidades de datos[8] que componen el flujo de sonido. En la figura 2.2 se detallan los instantes e intervalos de tiempo relevantes para el estudio del tratamiento de una unidad de datos en un sistema de transporte en tiempo real. Por simplicidad, el cronograma muestra un transmisor y un receptor; no obstante, los sistemas con mayor número de receptores y/o transmisores, están caracterizados por los mismos parámetros, aunque con distintos valores concretos para cada componente. Nótese que, por el mismo motivo, se considera que no se produce fragmentación ni reunificación de las unidades de datos en ninguna de las etapas. Asimismo, por ahora no tendremos en cuenta las diferencias existentes, adelantadas anteriormente, entre los distintos tipos de datos transportados, como sonido o imágenes, e ignoraremos las posibles pérdidas de paquetes durante la transmisión.

La tabla 2.1 recoge los parámetros temporales relevantes en el extremo del transmisor. Para cada unidad de datos, los parámetros anteriores verifican las siguientes igualdades, que expresan las condiciones impuestas por los algoritmos utilizados para codificar y empaquetar las unidades de datos:

$$t_{c,i} = t_{a,i} + T_{c,min} + \Delta_{c,i} \tag{2.1}$$

[8] En el capítulo 4 se distinguirá entre unidad de datos de transporte, tal y como se entiende en el modelo TCP/IP, y unidad de datos de aplicación, concepto más adecuado como unidad de transporte en aplicaciones de tiempo real, y que es el utilizado en el modelo RTP.

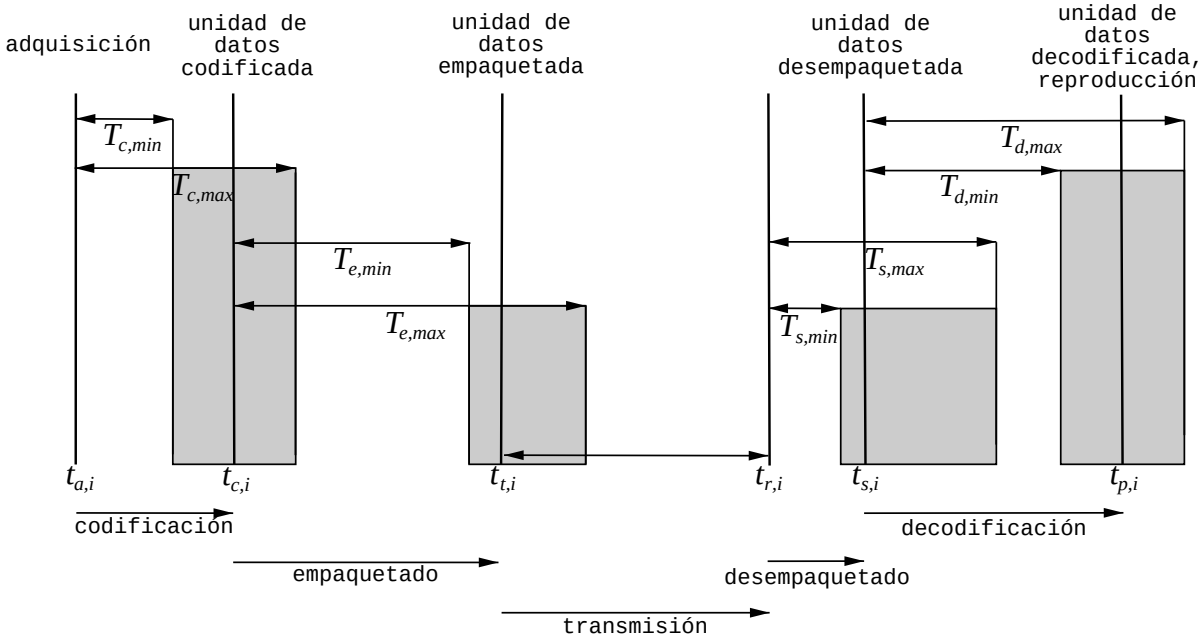


Figura 2.2: Cronograma de un sistema de transporte en tiempo real

$$t_{t,i} = t_{a,i} + T_{c,min} + \Delta_{c,i} + T_{e,min} + \Delta_{e,i} = t_{a,i} + \delta_{t,i} \tag{2.2}$$

Teniendo en cuenta el rango de valores de $\Delta_{c,i}$ y $\Delta_{e,i}$, en la ecuación 2.2, el retardo total provocado por el sistema transmisor para cada unidad de datos, $\delta_{t,i}$, está acotado inferiormente por $T_{c,min}+T_{e,min}$. La variación del retardo total para cada unidad respecto al mínimo global viene dada por $\Delta_{c,i} + \Delta_{e,i}$. Nótese que $T_{c,min}$ será siempre mayor o igual que el intervalo de tiempo cubierto por cada unidad de datos. Es decir, si una aplicación de telefonía transmite paquetes de sonido correspondientes a intervalos de 20 milisegundos, desde el primer paso (codificación) se introduce un retardo mayor o igual que 20 milisegundos, el tiempo necesario para formar una unidad de datos.

En general [94], en un sistema de transporte en tiempo real son deseables las siguientes características: bajo retraso, baja variación en el retraso, capacidad de adaptación a las condiciones de la red, alta utilización del ancho de banda disponible y reducidos requisitos de procesamiento en los elementos externos e intermedios de la red.

Ante este esquema, cabe destacar que:

- El retardo total del sistema así como su fluctuación están acotados inferiormente por el retardo y las fluctuaciones en el transmisor. Por tanto, con objeto de reducir el retardo total del sistema y sus fluctuaciones, tanto el algoritmo de codificación como el de empaquetamiento han de ser sencillos en dos aspectos: requerir poco tiempo y requerir un tiempo uniforme. El nivel de cumplimiento de estos requisitos está condicionado, en principio, por el formato de codificación de los datos.

- Cuando sea admisible aumentar deliberadamente el retardo de extremo a extremo ya desde la etapa de transmisión, se puede formar una *cola de transmisión periódica* de unidades de datos. Esta técnica de amortiguación pretende garantizar la mayor precisión posible en el periodo de transmisión retrasando la transmisión de las unidades tanto como sea necesario, formándose así

$t_{a,i}$	instante de *adquisición* de la i-ésima unidad de datos por transmitir.
$T_{c,min}$	mínimo intervalo de tiempo invertido en *codificar* una unidad de datos.
$T_{c,max}$	máximo intervalo de tiempo invertido en *codificar* una unidad de datos. Por definición, $T_{c,max} \geq T_{c,min}$.
$\Delta_{c,i}$	fluctuación en el intervalo de *codificación*, en general distinta para cada unidad de datos, y verificando $0 \leq \Delta_{c,i} \leq T_{c,max} - T_{c,min}$.
$t_{c,i}$	instante en el que se dispone de la unidad de datos tras su *codificación*
$T_{e,min}$	mínimo intervalo de tiempo necesario para *empaquetar* una unidad de datos en el formato de transporte.
$T_{e,max}$	máximo intervalo de tiempo necesario para *empaquetar* una unidad de datos en el formato de transporte. Por definición, $T_{c,max} \geq T_{c,min}$.
$\Delta_{e,i}$	fluctuación en el intervalo invertido en *empaquetar* una unidad de datos. En general distinta para cada unidad y cumpliendo $0 \leq \Delta_{c,i} \leq T_{e,max} - T_{e,min}$.
$t_{t,i}$	instante a partir del cual la unidad de datos i-ésima está codificada y empaquetada, pudiéndose realizar su transmisión.

Cuadro 2.1: Parámetros temporales de un transmisor en tiempo real

una cola de paquetes en espera de turno. Los métodos aplicables al ajuste de la duración de la cola son similares a los que se discutirán en la siguiente sección para el caso del receptor. La utilidad de esta técnica es menor cuanto mayor sea la variabilidad en el retardo de transporte de extremo a extremo en comparación con el tiempo de procesamiento en el transmisor.

- Para aprovechar el ancho de banda disponible, dado que los elementos intermedios de la red suelen ser dispositivos de almacenamiento-retransmisión, como se hace notar en [166], conviene que las unidades transmitidas sean del menor tamaño posible, de modo que las operaciones de transmisión se inicien cuanto antes en cada uno de los nodos de la red[9]. Asimismo, cuando el lapso de tiempo representado por cada unidad es menor, el retraso de extremo a extremo se ve reducido. Reduciendo el tamaño de las unidades es además posible reducir la tasa de pérdidas Por ejemplo, el tamaño de la celda ATM, 48 octetos, se eligió de modo que fuese adecuado para aplicaciones de transmisión de voz, especialmente con objeto de evitar el uso de equipos de cancelación de eco en enlaces continentales.

De forma análoga, en el extremo del receptor se distinguen los parámetros especificados en la tabla 2.2. Los algoritmos utilizados para desempaquetar y decodificar cada una de las unidades de datos imponen las siguientes restricciones:

$$t_{s,i} = t_{r,i} + T_{s,min} + \Delta_{s,i} \tag{2.3}$$

$$t_{p,i} = t_{r,i} + T_{s,min} + \Delta_{s,i} + T_{d,min} + \Delta_{d,i} = t_{r,i} + \delta_{r,i} \tag{2.4}$$

En la ecuación 2.4, el retardo introducido por el sistema receptor para cada unidad de datos, $\delta_{r,i}$ está acotado inferiormente por $T_{s,min} + T_{d,min}$, y fluctúa según $\Delta_{s,i} + \Delta_{d,i}$. Nótese que el intervalo

[9]Téngase en cuenta que los dispositivos de almacenamiento-retransmisión no inician la retransmisión de las unidades de datos hasta que su recepción se completa.

$t_{r,i}$	instante de *recepción* de la unidad de datos i-ésima, en el formato de transporte.
$T_{s,min}$	mínimo intervalo de tiempo necesario para *desempaquetar* una unidad de datos.
$T_{s,max}$	máximo intervalo de tiempo necesario para *desempaquetar* una unidad de datos. Por definición, $T_{s,max} \geq T_{s,min}$.
$\Delta_{s,i}$	fluctuación en el intervalo invertido en desempaquetar, distinta para cada unidad, y cumpliendo $0 \leq \Delta_{s,i} \leq T_{s,max} - T_{s,min}$.
$t_{s,i}$	instante en el que se dispone de la unidad de datos *tras desempaquetarla*.
$T_{d,min}$	mínimo intervalo de tiempo invertido en *decodificar* una unidad de datos.
$T_{d,max}$	máximo intervalo de tiempo invertido en *decodificar* una unidad de datos. Por definición, $T_{d,max} \geq T_{d,min}$.
$\Delta_{d,i}$	fluctuación en el intervalo de *decodificación*. En general es distinta para cada unidad y verifica $0 \leq \Delta_{d,i} \leq T_{d,max} - T_{d,min}$.
$t_{p,i}$	instante a partir del cual la unidad de datos i-ésima está decodificada y desempaquetada, pudiéndose realizar su *reproducción*.

Cuadro 2.2: Parámetros temporales de un receptor en tiempo real

de disponibilidad de unidades de datos en el receptor viene dado por $T_i = t_{p,i} - t_{a,i}$. Asimismo, el retardo introducido por la red que une a transmisor y receptor es $t_{r,i} - t_{t,i}$.

En la práctica, teniendo en cuenta que, salvo contadas excepciones, los dispositivos empleados habitualmente pueden llevar a cabo los algoritmos de codificación y decodificación y empaquetado en tiempo inferiores al milisegundo, cabe esperar que esta magnitud sea sensiblemente superior a $\delta_{t,i}$ y $\delta_{r,i}$, tanto en valor como en variabilidad. Piénsese por ejemplo en una conversación transoceánica basada en transmisión de sonido en formato PCMU (PCM de ley μ) sobre paquetes RTP. En este caso, el retardo introducido por la codificación y decodificación en circuitos electrónicos y la construcción e interpretación de paquetes por parte de las aplicaciones es despreciable frente al retardo de propagación de la red. Sin embargo, no todos los formatos permiten que la etapa de codificación se complete en un tiempo despreciable [86, 106].

Como primeras conclusiones, se ha de destacar que:

- Si el receptor debe modificar el tratamiento de las unidades de datos en función del retardo total de extremo a extremo que éstas sufren, debe disponer de información acerca del tiempo de adquisición de cada unidad de datos, $t_{a,i}$. Esta información se desconoce en el extremo del receptor, a menos que se transporte junto con los datos en las unidades transmitidas.

- Como en el caso del transmisor, la complejidad del formato de codificación y empaquetado condiciona el retardo y las fluctuaciones temporales introducidos por el sistema receptor. Ahora bien, generalmente, y al igual que en muchos otros sistemas de telecomunicaciones, conviene que los procesos de decodificación y desempaquetado sean más sencillos que los de codificación y empaquetado, favoreciéndose a los equipos receptores. Esto se debe a que el número de receptores suele ser mayor, incluso en varios órdenes de magnitud. Aunque puede parecer que esta asimetría en detrimento de los transmisores entra en contradicción con las conclusiones del

análisis de los parámetros temporales del transmisor, realmente no implica más que concentrar el gasto de recursos en los equipos de transmisión. De este modo es posible mantener bajo y estable el retardo introducido por el transmisor, al tiempo que se reduce la complejidad de los receptores.

- A la vista de la variabilidad mostrada por los parámetros temporales del sistema, si los receptores se limitasen a reproducir las unidades de datos tras ser desempaquetadas y decodificadas, no se podría garantizar regularidad alguna en el intervalo de reproducción. La siguiente sección analiza este problema y sus posibles soluciones.

2.2.2. Amortiguación en el receptor

En las aplicaciones multimedia de tiempo real, el ritmo de reproducción debe ser el mismo que el de adquisición; de lo contrario, se producen efectos incómodos –silencios, cortes de sonido, deformación de imágenes, etc– comparables a los producidos cuando faltan unidades de datos en el receptor. Es decir, los receptores deben amortiguar las unidades de datos recibidas, de modo que se pueda mantener un intervalo de reproducción constante independiente de la variación en el intervalo de tiempo comprendido desde la adquisición de cada unidad de datos en el transmisor, $t_{a,i}$, hasta que está disponible en el receptor para ser reproducida, $t_{p,i}$.

Los mecanismos de amortiguación en el receptor están basados en formar una cola de paquetes recibidos y listos para ser reproducidos, retrasando artificialmente su reproducción el tiempo oportuno para que el número de unidades que se encuentren disponibles a tiempo sea lo mayor posible. Con este procedimiento se pretende *resincronizar* y *reordenar* las unidades, recibidas y procesadas a intervalos variables, antes de su reproducción. En la figura 2.3, generalizada a partir de [116], se muestra, sobre la misma escala temporal, la adquisición de unidades en el extremo del transmisor y la correspondiente reproducción en el extremo del receptor. Nótese que no se consideran pérdidas, sino sólo desorden en la llegada de unidades. Se presentan varios posibles valores para el retardo de amortiguación, poniéndose de manifiesto los diferentes resultados que se pueden obtener. El valor etiquetado como “óptimo” permitiría reproducir los datos iniciales sin distorsión alguna, si todas las unidades adquiridas llegan al receptor.

En todo caso, se ha de tener en cuenta que la capacidad de almacenamiento del equipo receptor, y, por tanto, el tamaño máximo de la cola de amortiguación pueden estar limitados. Para algunos formatos de vídeo, las aplicaciones requieren tasas de transferencia del orden de megabits por segundo, por lo que se pueden producir pérdidas por saturación de la cola de recepción. Teniendo en cuenta además que en general el retardo total de extremo a extremo debe ser lo menor posible, se concluye que el valor óptimo para maximizar el número de unidades presentes a tiempo no es siempre el más adecuado para la aplicación.

El retardo artificial introducido por la cola de amortiguación puede ser *estático* o *dinámico*. En ciertos casos es posible determinar con suficiente precisión un valor de retardo de amortiguación fijo, en función de las fluctuaciones temporales conocidas del sistema de transmisión, de la red y del sistema de recepción. Esta opción es preferible frente a un retardo ajustable de forma dinámica, puesto que todo ajuste, aunque persigue adaptar el comportamiento a las condiciones actuales, implica interrumpir la reproducción, en caso de aumento del retardo, o descartar algunas unidades de datos, en caso de disminución del retardo.

Cuando los parámetros que determinan el valor óptimo del retardo de amortiguación no están

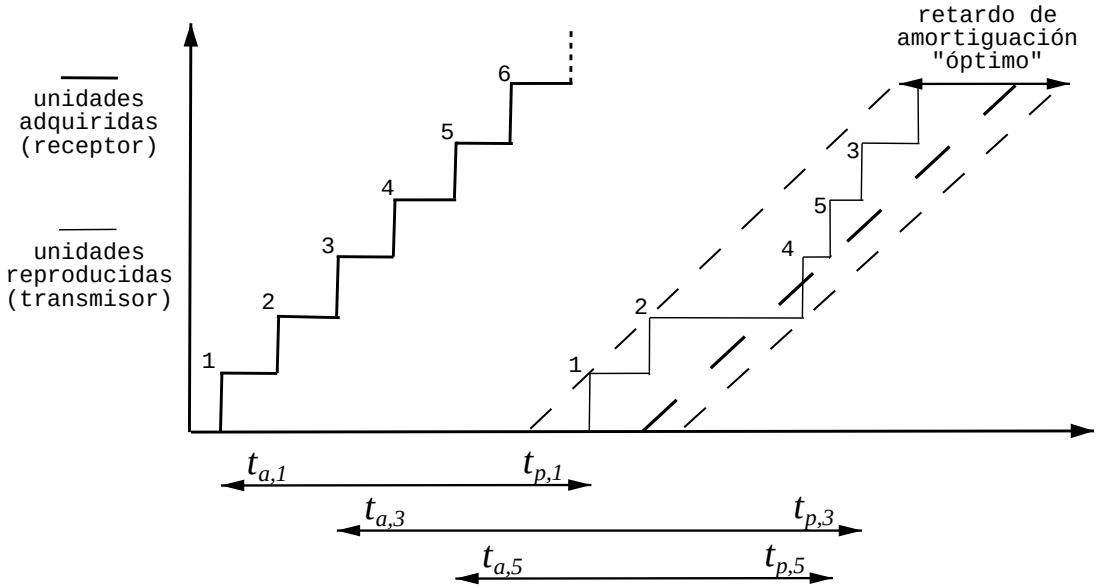


Figura 2.3: Cálculo del retardo de amortiguación

suficientemente acotados, el equipo receptor debe calcular tal valor de forma dinámica. El cálculo se ha de realizar a partir del valor del retardo total del sistema, o retardo de extremo a extremo, es decir, $t_{p,i} - t_{a,i}$. Los algoritmos propuestos hasta la fecha obtienen el valor como una combinación lineal de estimaciones de la media y la desviación típica del retardo total. El procedimiento seguido para obtener las estimaciones determina la velocidad de la reacción ante cambios en las condiciones del sistema.

El valor del retardo total puede ser distinto para cada unidad de datos; sin embargo, no es factible modificar el tamaño del retardo de amortiguación para cada nueva unidad. En función de la aplicación se debe escoger puntos de actualización del retardo de amortiguación de modo que sus potenciales efectos molestos se aprecien lo menos posible. En [116] se proponen cuatro algoritmos aplicados a transmisión de voz, diferenciados por el procedimiento de cálculo de las estimaciones citadas anteriormente; en este caso, los puntos de actualización del retardo de amortiguación se sitúan tras los silencios de los interlocutores, aprovechando que un silencio algo mayor o menor no es apreciable.

Este enfoque, generalmente válido para aplicaciones de transmisión de voz, caracterizadas por tener tasa de transferencia constante, CBR, a intervalos, no suele ser útil en todas las aplicaciones de sonido, y en especial en las de transmisión de vídeo, caracterizadas por tener tasa de transferencia variable, VBR, y tráfico continuo.

Por último, durante la discusión anterior no se ha tenido en cuenta que el tamaño, y, por tanto, el intervalo representado por las unidades de datos puede ser variable. Si incorporamos este factor al modelo descrito, la escala temporal mostrada en la figura 2.3 no sería uniforme. La solución más simple a este problema es considerar que la variación en el tamaño de las unidades de datos es un factor más que contribuye a aumentar las fluctuaciones del retardo total del sistema. En la sección 2.3.1 se trata este tema en el contexto del protocolo RTP.

2.2.3. Control de Congestión

Al principio de este capítulo expusimos la evolución de los mecanismos de control de congestión de TCP. Desde su definición inicial [120], TCP incluye mecanismos de control de flujo entre extremos de la transmisión basados en una ventana de transmisión [166, 66, 178] que indica el número de octetos que se puede transmitir sin haber recibido confirmación. Cuando se transmiten octetos, el tamaño de la ventana se reduce, mientras que cuando se reciben acuses de recibo, aumenta. De este modo, el transmisor ajusta la tasa de transmisión de datos de acuerdo con la tasa de recepción de segmentos de acuse de recibo, considerada por tanto como una estimación de la tasa de recepción de datos en el receptor.

La mayoría de las aplicaciones utilizadas actualmente en Internet y, con ello, la mayoría del tráfico se basa en TCP. Como se expuso, la extensión del uso de TCP a la mayoría de las aplicaciones fue posible conforme se fueron introduciendo y mejorando mecanismos de control de congestión para TCP. Estas mejoras consistieron en el ajuste y ampliación del mecanismo de ventana de transmisión de modo que, además de permitir la sincronización entre los extremos de la transmisión, se tuviesen en cuenta modelos de comportamiento de la red desde un punto de vista global [2]. El objetivo es evitar situaciones de congestión de la red; esto es, situaciones tales que *un incremento en el tráfico supone un descenso del rendimiento de la red [66, capítulo 4]*. Las mejoras comentadas, cuya necesidad se hizo patente durante varios periodos de congestión en la Internet de finales de los 80, se han venido incorporando durante más de una década y han dado lugar a un esquema de control de congestión en Internet.

En efecto, en la actualidad el control de congestión se ha identificado como una función esencial en la gestión de redes [137, 7]. Ahora bien, TCP u otros protocolos de similares características no son adecuados para la transmisión de información multimedia, por lo que actualmente las aplicaciones multimedia en tiempo real de Internet están basadas mayoritariamente en UDP[10]. Estas aplicaciones, junto con las basadas en protocolos de transporte multicast fiable pueden incrementar significativamente el porcentaje de tráfico basado en protocolos distintos a TCP que, además, rara vez aplican técnicas de control de congestión de forma compatible con TCP, ni utilizan mecanismos de reserva de ancho de banda. Se teme que esta evolución pueda afectar al tráfico TCP y desembocar en situaciones de colapso de la red [46].

No obstante, el concepto de control de flujo tal y como se define para TCP, no es aplicable en el caso de los sistemas multimedia en tiempo real, que se caracterizan por admitir tan solo un conjunto generalmente prefijado y reducido de posibles tasas de transferencia. Por este motivo, se están desarrollando protocolos y técnicas de control de congestión con comportamiento compatible con TCP. El uso de estas técnicas en los sistemas de tiempo real es conveniente no sólo porque las aplicaciones de tiempo real deben coexistir en Internet con otros tipos de aplicaciones, sino porque además las aplicaciones que emplean mecanismos de control de congestión:

- Utilizan los recursos de la red de forma más eficiente, mejorando sus prestaciones.
- Pueden funcionar ante un mayor rango de anchos de banda y condiciones de la red.

Asimismo, existe la tendencia a introducir en los elementos intermedios de Internet mecanismos de defensa ante aplicaciones que no pongan en práctica estas técnicas [15]. En un futuro próximo, los

[10] Ya sea como protocolo base para RTP o como único protocolo de transporte.

elementos intermedios de la red se configurarán de manera que aquellas aplicaciones que no empleen mecanismos de control de congestión sean penalizadas (probablemente descartando sus unidades de datos en primer lugar cuando la red esté sobrecargada).

Los desarrollos realizados hasta la fecha en esta materia se pueden clasificar según se basen en mecanismos de ventana o en el ajuste dinámico de la tasa de transferencia de acuerdo con un modelo que permita estimar las condiciones de la red. Las técnicas basadas en mecanismos de ventana, al igual que TCP, no son válidas en general para los sistemas en tiempo real.

Las técnicas basadas en el ajuste dinámico de la tasa de transferencia utilizan un modelo matemático para calcular, en función de parámetros de la red (como el porcentaje de pérdidas o el intervalo de retorno de acuses de recibo), una tasa de transferencia tal que su uso sea compatible con flujos TCP. El modelo matemático puede reproducir el comportamiento de los mecanismos de control de congestión de TCP o, por el contrario, proporcionar la tasa de transferencia de TCP para unas determinadas condiciones en el estado estacionario. El segundo caso es más adecuado para sistemas de tiempo real, en los que la alta variabilidad de la tasa de transferencia de TCP no es admisible, esto es, la tasa de transferencia se debe ajustar minimizando el número y la intensidad de los cambios.

En general, la dependencia de la tasa de transferencia de un flujo TCP con respecto a la fracción de pérdidas en la transmisión, p, verifica la siguiente relación:

$$T \propto \frac{1}{\sqrt{p}}$$

No obstante, la obtención de ecuaciones que modelen el comportamiento de TCP en el estado estacionario, es un problema complejo para el que se vienen planteando diversas soluciones. Actualmente, el modelo más completo y que ha sido evaluado experimentalmente de manera más extensa [47, 190], define la siguiente ecuación para hallar la tasa de transferencia compatible con TCP:

$$T(t_{RTT}, t_{RTO}, s, p) = \min\left(\frac{W_m \cdot s}{t_{RTT}}, \frac{s}{t_{RTT}\sqrt{\frac{2Dp}{3}} + t_{RTO}\min\left(1, 3\sqrt{\frac{3Dp}{8}}\right) p(1+32p^2)}\right) \tag{2.5}$$

donde t_{RTT} es el intervalo de retorno de acuses de recibo, t_{RTO} es el periodo de retransmisión de paquetes del protocolo TCP bajo las mismas condiciones, s es el tamaño de las unidades de datos, y p es la fracción de pérdida de unidades de datos. Nótese que para aplicar este modelo, los parámetros citados se deben conocer en el transmisor, por lo que es necesario que los receptores los proporcionen. Por otra parte, W_m es el tamaño máximo de la ventana de congestión de TCP, y D denota el número de unidades de datos cuya recepción se confirma en cada paquete de acuse de recibo.

Este modelo es asimismo ampliable a entornos multicast [191], lo que requiere definir diversas técnicas para que sea posible la realimentación hacia el transmisor de información acerca de las condiciones de la red para los diferentes receptores.

2.3. Protocolo RTP. Perfil para Transporte de Sonido y Vídeo

El protocolo RTP [143, 144, 140] tiene como función el transporte de extremo a extremo para aplicaciones que requieren transmisión de datos en tiempo real (ya sea o no en régimen interactivo),

como pueden ser sonido, imágenes o datos de simulación. Está diseñado para que sus realizaciones se basen en otros niveles de red y transporte previamente establecidos, sin presentar dependencias de ninguno en concreto, por lo que se puede realizar sobre UDP/IP, IPX o ATM-AALx.

RTP se ha diseñado junto con su perfil de sonido y vídeo como protocolo de transporte para conferencias multimedia con múltiples participantes, aunque es asimismo válido para otras aplicaciones, como simulación interactiva, almacenamiento de flujos continuos de datos o aplicaciones de control de procesos.

Aunque generalmente se habla de RTP como un protocolo de transporte de nivel de aplicación, sería más riguroso definirlo como un modelo de protocolos, puesto que su especificación es deliberadamente incompleta y flexible[11], proporcionando una estructura y un conjunto de mecanismos comunes, en lugar de algoritmos concretos. De este modo se pretende seguir un enfoque de procesamiento integrado de los diferentes niveles que componen las aplicaciones [30], principio de diseño que se analizará en el capítulo 4. El desarrollo de una aplicación basada en RTP requiere seguir la especificación general conjuntamente con un perfil que la complete (como pueden ser el perfil general para transporte de sonido y vídeo [141], o el perfil para transporte seguro [10]), y, para los formatos no cubiertos por el perfil, especificaciones adicionales que definan reglas de empaquetado para el formato de datos[12].

En efecto, siguiendo únicamente la especificación de RTP [143] no se puede desarrollar el módulo de transporte de ninguna aplicación de conferencia multimedia, sino que, al menos, es necesario aplicar conjuntamente las directrices dadas en el perfil para transporte de sonido y vídeo [141]. Si la aplicación emplea algún formato audiovisual relativamente elaborado –como vídeo H.261 o H.263, o sonido Ogg Vorbis o Speex–, en la mayoría de los casos deberá seguir asimismo un formato de empaquetado que permita resolver problemas específicos del formato audiovisual.

Como nota general acerca de RTP, su diseño le hace ser aplicable tanto en entornos unicast como multicast, lo que en la práctica requiere que todos los mecanismos contemplados por el protocolo sean escalables. Asimismo, las funciones de reserva de recursos y garantía de calidad de servicio, necesarias en gran parte de las aplicaciones de conferencia multimedia, quedan fuera del alcance de RTP. Esto es, es necesario el uso de protocolos auxiliares que presten estos servicios, como RSVP[18, 9].

Por otra parte, la especificación inicial de RTP contempla mecanismos de soporte directo para cifrado, puesto que se finalizó antes de que la definición del modelo IPSec[35, 85] se completase.

RTP proporciona, entre otras, funciones de sincronización, detección de pérdidas, seguridad e identificación de participantes y contenidos. Estas funciones están repartidas entre dos componentes: el protocolo de transporte de datos RTP y el protocolo de control ligero RTCP, descritos en los siguientes epígrafes. La definición de RTCP como componente adicional permite que aplicaciones que requieran un mayor nivel de control que el proporcionado por RTCP lo reemplacen con otros protocolos de control de conferencia específicos.

[11] Así, RTP admite variaciones con objeto de adaptarlo a diversas aplicaciones, lo que permite, entre otras cosas, desarrollar perfiles adecuados a entornos multicast [27].

[12] Como pueden ser el formato de vídeo MPEG 2 [44], o el de empaquetado de sonido con codificación lineal y muestras de 24 bits [87].

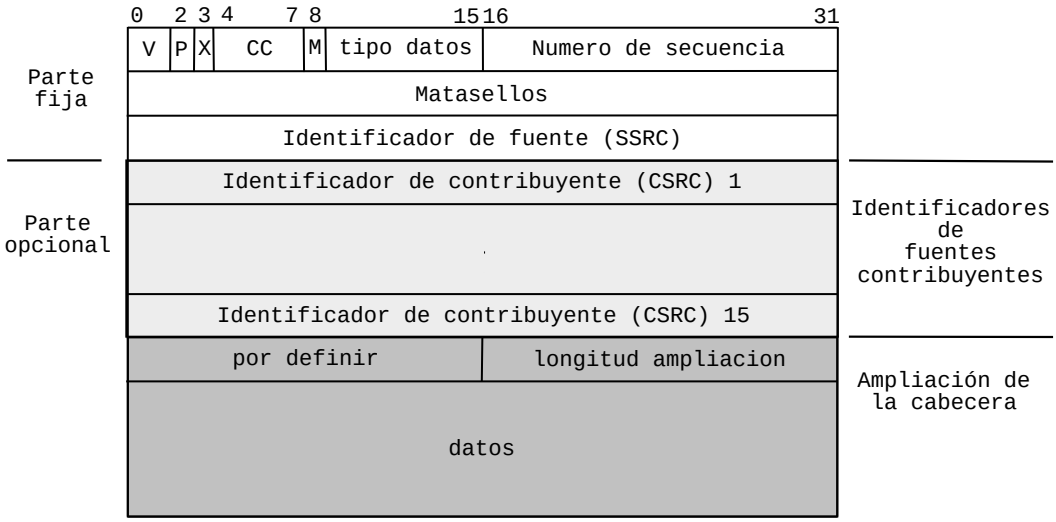


Figura 2.4: Estructura de las cabeceras de los paquetes RTP

2.3.1. Transporte de datos

La unidad de transporte de datos de nivel RTP es el paquete RTP, formado por una cabecera seguida del contenido o unidad de datos de la aplicación.

Cada sesión multimedia puede tener asociadas una o más sesiones RTP, cada una de las cuales corresponde a un tipo de datos empleado en la sesión. Una sesión RTP queda identificada por un par de direcciones de transporte. Por ejemplo, en una sesión multimedia en la que se transmite vídeo nv y sonido PCMA, se tendrán dos sesiones RTP, a cada una de las cuales se asocia un par de puertos distinto. Cada par de puertos comprende el puerto de recepción de paquetes RTP –que, en general, debe ser un número par- y el puerto de recepción de paquetes RTCP –que, en general, deber ser el número impar inmediatamente superior[13]. Nótese que esta definición de sesión excluye aquellas situaciones en las que, por razones prácticas, es inevitable que un mismo medio se transmita mediante más de una sesión RTP, como puede ser el caso de las redes de acceso celular o conexiones que utilicen dígitos DTMF [146]. Estas situaciones requieren ampliaciones en los protocolos de descripción de sesiones multimedia [23].

En la figura 2.4 se esquematiza la estructura de las cabeceras RTP. La cabecera RTP no contiene ningún campo que indique la longitud total del paquete, puesto que se asume que el protocolo subyacente realiza segmentación y que en cada una de sus PDU se transporta un sólo paquete RTP. Generalmente, RTP se realiza sobre UDP, por lo que estas condiciones se cumplen; sin embargo, es posible realizar RTP sobre protocolos que no proporcionan directamente el tamaño de cada paquete RTP. Por ejemplo, la especificación de RTSP [147] define un mecanismo para transportar paquetes RTP sobre TCP, consistente en añadir la información de control necesaria para identificar la longitud de cada paquete RTP dentro del flujo TCP.

La cabecera añadida por RTP a cada unidad de datos transportada consta de una parte fija y dos opcionales. Los campos V y P de la parte fija indican, respectivamente, la versión de RTP -actualmente 2- y si el paquete contiene octetos de relleno al final de los datos, en cuyo caso el último octeto del

[13]En las últimas revisiones de RTP estas restricciones se han relajado, véase la sección 2.3.6.

paquete indica cuántos octetos constituyen el relleno[14]. La parte fija contiene la información necesaria para realizar las siguientes funciones, identificadas anteriormente como requisitos de un sistema de transporte en tiempo real:

- *Identificación de fuentes de sincronización*. El campo SSRC es un identificador numérico, único dentro de cada sesión RTP[15], para cada aplicación que participa en la sesión. Este mecanismo básico de identificación está apoyado por otros mecanismos de RTCP, más elaborados, que proporcionan información adicional acerca de cada participante.

  Nótese que la fuente de sincronización no se identifica mediante la dirección de transporte para permitir que de una misma dirección se reciban múltiples flujos para un mismo medio, por ejemplo, imágenes de varias cámaras de vídeo. Esto es, del mismo modo que un número de puerto proporciona multiplexión a nivel de transporte, el identificador SSRC proporciona multiplexión de flujos. Esta característica permite además el uso de traductores RTP (ver sección 2.3.4), que modifican la dirección de transporte de origen de los paquetes correspondientes a los flujos que traducen.

  Asimismo, el campo CC es un contador del número de identificadores de fuentes contribuyentes especificados en la primera parte opcional de la cabecera RTP. Este contador toma valores entre 0 y 15. Las aplicaciones de carácter general no combinan datos procedentes de distintas fuentes, por lo que el valor del campo CC sólo será distinto de 0 para elementos que actúen como mezcladores o traductores, véase la sección 2.3.4.

- *Identificación del tipo de contenido*. El campo tipo de datos es un identificador del formato de los datos transportados. Los valores más comunes, especificados en [141, 28], están reservados a través de la IANA; otros, se asignan de forma dinámica. Cuando las aplicaciones utilizan identificadores dinámicos, deben negociar su uso mediante mecanismos de control ajenos a RTP, como pueden ser los protocolos SIP [60] o MGCP [4], que generalmente especificarán el contenido mediante descripciones SDP[57] (véase el epígrafe 3.6.1). La indicación explícita del tipo de contenido en cada paquete permite cambiar el formato de codificación del medio de una sesión en cualquier instante de ésta.

- *Reordenación de la secuencia de paquetes*, basada en un número de secuencia de 16 bits, incrementado en uno por cada paquete y situado al final de la primera palabra de cuatro octetos de la cabecera. El campo número de secuencia es la base de funciones como la reordenación de paquetes o el filtrado de paquetes cuyo número de secuencia presenta una correlación demasiado baja respecto a los últimos recibidos.

- *Detección de pérdidas* para estimar la calidad de la transmisión y activar mecanismos de recuperación. Los cálculos se basan en el mismo campo número de secuencia y en la información proporcionada en los informes RTCP de transmisión. del mismo modo, los receptores informan a los transmisores sobre la calidad de la recepción mediante paquetes RTCP.

- *Sincronización dentro de una sesión RTP*, basada en el campo matasellos, un contador de 32 bits correspondiente al parámetro $t_{a,i}$ presentado en el epígrafe 2.2.1. La escala temporal de este campo depende el formato concreto transportado. Asimismo, el campo M indica en general

[14] Incluyendo el último octeto.

[15] Cada participante debe generar su identificador como un número aleatorio, existiendo mecanismos para detectar y solventar posibles conflictos.

unidades de datos a partir de las cuales se pueden situar puntos de resincronización y actualización del retardo de sincronización. El significado y uso concreto de este campo ha de definirse en perfiles de RTP.

- *Posibilidad de experimentación con nuevos formatos*. El campo X permite especificar si la segunda parte opcional de la cabecera RTP, ampliación de cabecera, está presente. En tal caso, se puede incluir información útil para formatos aún en desarrollo. Las aplicaciones que no conozcan una ampliación concreta utilizada en un paquete RTP deben ignorarla, pero deben interpretar correctamente el resto del paquete. De este modo, aplicaciones experimentales pueden coexistir con aplicaciones ya existentes que desconocen el nuevo formato. Este mecanismo está contemplado únicamente para experimentar con nuevos formatos durante su desarrollo. Una vez finalizada la fase de desarrollo del formato, éste se debe especificar en forma de RFC.

La estructura de la cabecera RTP facilita su compresión, lo que permite transmitir de forma eficiente datos en tiempo real a través de enlaces de ancho de banda bajo, como pueden ser los enlaces SLIP o PPP. Al comprimir las cabeceras RTP/UDP/IP como un bloque [196], se explota una idea conocida y aplicada desde hace tiempo, véase [75] o [170, capítulo 20]. Si, para el caso de las cabeceras TCP/IP se consigue reducir el tamaño de 40 a 3 octetos, en el caso de las cabeceras RTP/UDP/IP, se puede conseguir resultados equivalentes.

2.3.2. Control de sesiones

RTCP es un protocolo de control de la transmisión basado en el intercambio periódico de paquetes de control entre los participantes. Su uso requiere que el protocolo de transporte subyacente realice la multiplexión de los paquetes de control y datos, que se transmiten por separado (por ejemplo, hacia diferentes puertos UDP). RTCP cumple las siguientes funciones, complementarias a las descritas para RTP:

- *Identificación de participantes y sincronización entre distintas sesiones RTP*. Los paquetes RTCP contienen un identificador textual, nombre canónico, único para cada participante en una sesión multimedia. Es decir, el nombre canónico de un participante, a diferencia del identificador numérico SSRC es único para todas las sesiones RTP que pueden componer una sesión multimedia. Por tanto, el nombre canónico permite determinar qué flujos de datos –por ejemplo, voz e imágenes– proceden de un mismo participante, de modo que sea posible sincronizar distintas sesiones RTP dentro de una sesión multimedia. Además, el nombre canónico permite identificar a cada participante de forma persistente, independientemente de si en algún momento se produce un conflicto de identificadores SSRC.

- *Realimentación acerca de la calidad de recepción*. Esta información, proporcionada en distintos tipos de paquetes de informe, sirve de base para monitorizar y diagnosticar problemas de transmisión, en especial en entornos multicast. Los mecanismos de control de congestión que se describen en el epígrafe 2.3.5 requieren asimismo esta información.

- *Estimación del número de participantes*. El envío de paquetes de control RTCP por parte de todos los participantes en una sesión permite que cada uno de ellos pueda estimar de forma dinámica el número total de participantes. Así, pueden ajustar la frecuencia de transmisión de paquetes RTCP para mantener el límite de ancho de banda de control establecido en la sesión.

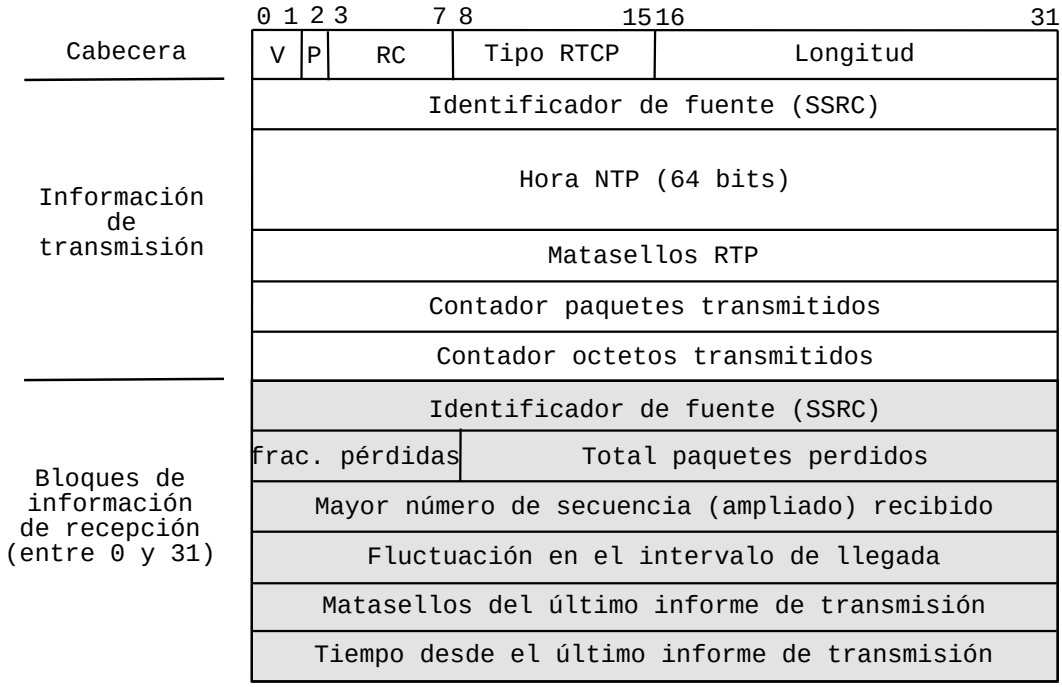


Figura 2.5: Estructura de los paquetes RTCP tipo SR

En las últimas actualizaciones de RTP, el ancho de banda reservado para paquetes de control de la sesión es un parámetro más dentro de su descripción, existiendo ampliaciones de SDP [26] para este parámetro, véase la sección 3.6.1.

- *Proporcionar información de control de la sesión*, como nombre, dirección o teléfono de cada participante. esta función opcional se basa en la definición de unos pocos tipos de paquete RTCP que proporcionan la información necesaria para realizar un control mínimo de la sesión. Las aplicaciones con requisitos especiales de control pueden sustituir o complementar estos mecanismos con protocolos de control más complejos.

La información necesaria para realizar estas funciones se transporta en los cinco tipos de paquete RTCP definidos inicialmente:

- *Informes de transmisión (SR)*, con los que los participantes que actúan como transmisores proporcionan información acerca de cuántos y cómo transmiten paquetes RTP. La figura 2.5 muestra la estructura de los paquetes RTCP de este tipo, formados por un primer bloque relativo a los paquetes RTP transmitidos, y un segundo bloque de hasta 31 informes relativo a los paquetes RTP recibidos de otros tantos transmisores. Se contempla la posibilidad de añadir un tercer bloque con ampliaciones específicas definidas en perfiles de RTP.

  Los campos V y P son análogos a los campos de las cabeceras RTP. El campo RC es un contador que indica el número de informes de recepción que constituyen el segundo bloque del paquete. El campo tipo discrimina los cinco tipos de paquetes RTCP definidos.

  Al proporcionar el matasellos NTP [103] –hora absoluta– junto con el matasellos RTP –con la frecuencia específica de la sesión RTP–, los receptores pueden calcular la correspondencia entre ambos, teniendo así un método para hallar el valor del parámetro $t_{a,i}$ para cada paquete

RTP recibido. Asimismo, los receptores utilizan el contador de paquetes enviados contenido en el paquete RTCP SR junto con los números de secuencia de los paquetes RTP para calcular la fracción de paquetes perdidos.

- *Informes de recepción (RR)*, generados por los participantes que no transmiten datos y con estructura y funciones idénticas al segundo bloque de los informes de transmisión.
- *Descripción de fuentes (SDES)*, que contienen el texto correspondiente a los siguientes datos del participante: nombre canónico, nombre, dirección de correo electrónico, teléfono, situación, aplicación, comentario y ampliaciones específicas de la aplicación.
- *Funciones específicas de la aplicación (APP)*, con formato libre y orientado al uso en aplicaciones experimentales, de forma similar a las ampliaciones de cabeceras RTP. Del mismo modo, las aplicaciones deben ignorar paquetes RTCP APP desconocidos, y una vez completado el desarrollo de un nuevo tipo de paquete RTCP mediante este mecanismo, es recomendable registrarlo públicamente a través de la IANA [71].
- *Abandono de sesión (BYE)*, que pueden contener, de forma opcional, una cadena de texto indicando la razón del abandono.

Con el fin de maximizar el uso útil de ancho de banda, los paquetes RTCP se transmiten en paquetes compuestos, formados al menos por un informe de transmisión si procede, o de recepción en otro caso, y por una descripción de fuente de tipo nombre canónico. Por este motivo, todos los tipos de paquete RTCP incluyen un campo que indica su longitud, siendo así posible localizar cada uno de los paquetes RTCP que forman un paquete compuesto.

Como se puede concluir a la vista de las funciones expuestas, RTCP es un componente básico en cualquier sistema que emplee RTP como protocolo de transporte, aunque en algunos casos, por ejemplo si los enlaces utilizados son unidireccionales, su uso puede quedar limitado o incluso ser imposible. En estas situaciones, RTP sigue siendo un protocolo de transporte válido, aunque muchas de sus funciones quedan limitadas. En particular, los sistemas que deben ser fiables y tolerantes a fallos dependen en gran medida de las funciones que proporciona RTCP.

2.3.3. Perfiles

Hasta la fecha sólo se ha llevado a la práctica un perfil para RTP en forma de RFC, el perfil para transmisión de sonido y vídeo en conferencias con control mínimo [141]. El perfil de RTP para sonido y vídeo completa y adapta la especificación de RTP [143] para su uso en aplicaciones de conferencia multimedia. En general, la mayoría de las opciones por omisión consideradas en la especificación general de RTP no se modifican.

Además de registrar valores del campo tipo de datos para los formatos de sonido y vídeo más comunes y especificar otros tipos dinámicos, completa la definición del campo M de la cabecera RTP. Este queda definido como identificador del primer paquete tras un intervalo de silencio para aplicaciones con supresión de silencios. En otros casos, como el formato definido para transportar vídeo MPEG sobre RTP [44], puede identificar además los paquetes que contienen el final de una imagen. Asimismo, se especifica la fracción de ancho de banda de control a dedicar para cada tipo de paquete RTCP. Los mecanismos previstos para ampliar las cabeceras RTP y añadir nuevos tipos de paquetes RTCP no se utilizan.

Actualmente existen dos propuestas de nuevos perfiles para RTP; ambas definen realmente ampliaciones del perfil para sonido y vídeo:

- *Perfil para sonido y vídeo con realimentación basada en RTP (AVPF)[114]*. Introduce modificaciones en los algoritmos de regulación del periodo de transmisión de paquetes RTCP así como nuevos tipos de paquetes RTCP, de modo que sea posible retransmitir paquetes RTP en respuesta a solicitudes enviadas mediante RTCP. Los nuevos tipos de paquetes RTCP permiten, entre otras cosas, que los receptores puedan enviar acuses de recibo tanto afirmativos como negativos.
- *Perfil para RTP seguro (SRTP) [10]*. Este perfil define ampliaciones con el objetivo de garantizar la confidencialidad de los datos transmitidos y de la información de control de la sesión, así como la integridad de todos los paquetes transmitidos y la protección de las respuestas. A diferencia del perfil anterior, no define nuevos tipos de paquetes, sino que se hace uso de la posibilidad de definir ampliaciones para paquetes RTP y RTCP, incorporándose campos adicionales para indicar claves de cifrado y etiquetas de autenticación.

Ambos perfiles mantienen las restricciones en los algoritmos de regulación del uso de ancho de banda y de gestión de participantes establecidos para RTP y su perfil de sonido y vídeo, por lo que no reducen la escalabilidad ni la adecuación a entornos multicast.

2.3.4. Componentes intermedios de nivel RTP

El diseño de RTP prevé la existencia de sistemas intermedios de nivel RTP, además de los sistemas terminales o aplicaciones de usuario. Los experimentos realizados en redes multicast durante el desarrollo de RTP han puesto de manifiesto la utilidad de estos elementos, en particular para solucionar los problemas derivados de la presencia de cortafuegos y conexiones de bajo ancho de banda, con mayor flexibilidad que la permitida por traductores de niveles inferiores.

Dentro de los diversos escenarios de aplicación de RTP, existen casos en los que todos los participantes en una sesión RTP pueden no querer recibir la información multimedia en el mismo formato, especialmente en sesiones que se extienden a varias redes, unas con ancho de banda significativamente mayor que otras. En lugar de que todos tengan que utilizar la configuración más restrictiva, se puede introducir un elemento repetidor de nivel RTP, un mezclador RTP, en el punto de enlace con las zonas de ancho de banda limitado. Estos sistemas deben sincronizar los datos entrantes, en general procedentes de varias fuentes de sincronización, mezclar los diferentes flujos entrantes en uno sólo, traducir el formato a otro de menor ancho de banda y retransmitir el flujo obtenido.

Asimismo, algunos participantes pueden disponer del ancho de banda necesario para seguir una sesión multicast, pero no tener acceso multicast por encontrarse tras un cortafuegos. Para estos casos, un repetidor de nivel RTP, un traductor RTP, puede hacer de intermediario. En estos casos, se requiere instalar dos traductores (o un traductor bidireccional), uno a cada lado del cortafuegos; el primero introduce los flujos procedentes de la red multicast en la red unicast, y el segundo realiza la operación inversa.

Los traductores se diferencian de los mezcladores en que no modifican los identificadores SSRC de los flujos que retransmiten, mientras que los mezcladores introducen su propio identificador en los flujos que retransmiten e insertan los identificadores originales en la lista de fuentes contribuyentes, véase la sección 2.3.1.

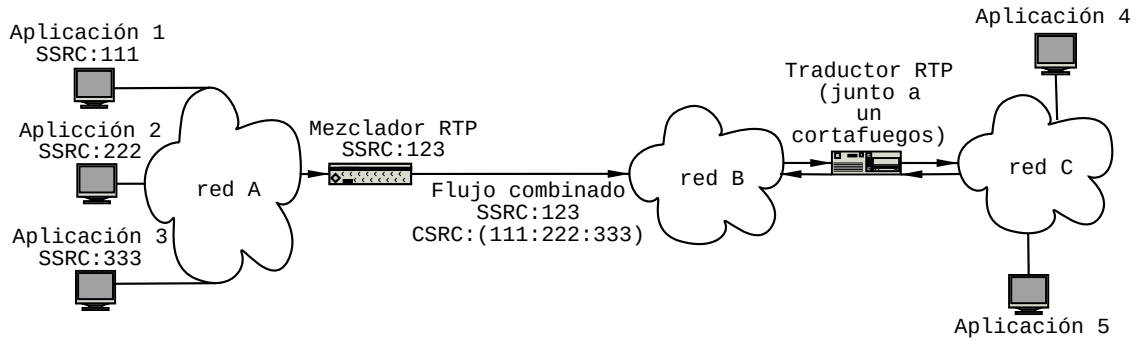


Figura 2.6: Ejemplo de una red RTP con traductores y mezcladores

En la figura 2.6 se muestra un esquema de un sistema RTP en el que se ha instalado varios traductores y mezcladores. El mezclador RTP que conecta las redes A y B combina los flujos procedentes de las aplicaciones 1, 2 y 3 (con identificadores de fuentes de sincronización 111, 222 y 333, respectivamente) en un flujo que tienen como identificador de fuente de sincronización el del mezclador (123), y como identificadores de fuentes contribuyentes los de las tres aplicaciones mencionadas. El traductor bidireccional que conecta las redes B y C no modifica los identificadores de fuentes.

En estos sistemas, se definen varias reglas de configuración [144] con el fin de evitar la formación de bucles[16]. Asimismo, el procesamiento y retransmisión de paquetes RTCP debe seguir normas [144] diferentes para traductores y mezcladores. En general, estas reglas, que afectan tanto a los datos estadísticos contenidos en los informes RTCP como a la temporización del envío de informes, se deben adaptar a la finalidad para la que se ha instalado el sistema, que puede ser conversión de formatos multimedia, cifrar/descifrar los flujos, o adaptar entornos unicast y multicast, entre otras.

2.3.5. Control de Congestión

En secciones anteriores se ha justificado la creciente necesidad de añadir mecanismos de control de congestión a las aplicaciones basadas en RTP al igual que todas aquellas que no utilizan TCP y pueden suponer un porcentaje significativo del tráfico en Internet. En particular, los esquemas de control de congestión basados en ecuaciones son los más adecuados a las aplicaciones multimedia en tiempo real. No obstante, la ecuación 2.5 de la sección 2.2.3 requiere que los transmisores conozcan datos como la fracción de pérdidas observada por los receptores o el intervalo de retorno de acuses de recibo.

Estos parámetros se pueden obtener a partir de los informes RTCP de tipo RR. El primero se proporciona directamente en los informes de recepción remitidos por los receptores[17]. El intervalo de retorno se puede calcular a partir de dos campos de los informes RR: tiempo transcurrido desde que el receptor recibió el último informe de transmisión, y matasellos del último informe de transmisión. Puesto que el transmisor puede hallar el instante correspondiente al matasellos del último informe de transmisión, según se esquematiza en la figura 2.7, el tiempo de retorno se puede calcular restando al instante de recepción de un informe RR el instante en que se envío el último informe de transmisión

[16] Se debe tener en cuenta que todos las aplicaciones que se comunican a través de traductores y mezcladores siguen compartiendo el mismo espacio de identificadores SSRC.

[17] Aunque en algunos casos es preferible utilizar el contador total de paquetes perdidos, también presente en los informes RR, para conseguir una mayor precisión en los cálculos.

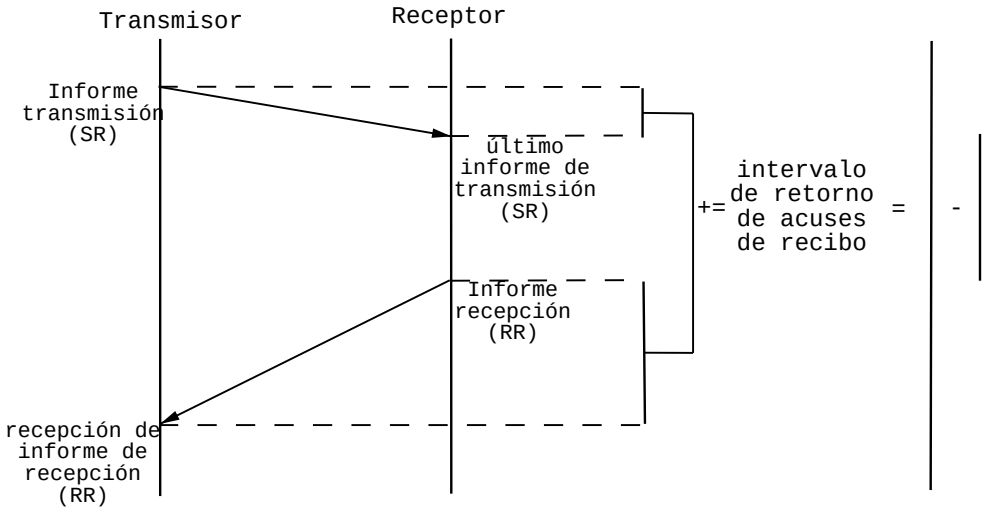


Figura 2.7: Ejemplo de una red RTP con traductores y mezcladores

(SR) recibido por el receptor y el tiempo que transcurrió entre la recepción del informe de transmisión y la transmisión del informe RR por parte del receptor.

Así, RTCP proporciona la realimentación necesaria sobre el estado de la red. Algunas variantes de las técnicas de control de congestión basadas en ecuaciones se han aplicado satisfactoriamente en sistemas basados en RTP [158, 159], comprobándose que se consigue un reparto equitativo del ancho de banda disponible entre varios flujos RTP, y entre éstos y otros flujos TCP. No obstante, la complejidad del problema y las dificultades que aparecen tanto en entornos multicast como en aplicaciones con un número elevado de receptores (y, por tanto, con una frecuencia de realimentación probablemente insuficiente), así como una evaluación más completa del efecto que la regulación de la tasa de transferencia tiene sobre la calidad de reproducción en los receptores, requieren el despliegue de estas técnicas en escenarios reales para que sean depuradas con la suficiente base experimental.

2.3.6. Evolución de RTP

La versión original de RTP, versión 0, protocolo de transporte empleado por la aplicación vat [180], data de 1991. Tras una versión intermedia, la versión 2 –vigente en la actualidad– quedó definida [143] a principios de 1996 junto en el perfil para transmisión de sonido y vídeo [141]. Desde 1998 se vienen desarrollando, como revisiones de la especificación de RTP, los borradores [144], [142] y [28], que próximamente sustituirán a los RFC originales, avanzando además al estado “Draft Standard”.

Estos borradores introducen dos modificaciones sustanciales junto con algunas de menor importancia. Ninguna de ellas conlleva incompatibilidades directas con sistemas RTP actuales, aunque sí modifican significativamente la gestión del ancho de banda.

Por una parte, se incorpora un algoritmo, presentado inicialmente en [128], para refinar el cálculo del periodo de transmisión de paquetes RTCP y del tiempo de espera antes de enviar paquetes de abandono de sesión, de modo que sus resultados sean más escalables. Con el mismo objetivo de aumentar la escalabilidad de RTP, se tienen en cuenta los problemas que puede plantear la gestión de un elevado número de fuentes de datos, incorporándose como solución los mecanismos de muestreo de fuentes propuestos en [11].

La otra modificación importante es la definición de requisitos de control de congestión para RTP, fijándose como obligatoria la aplicación de alguna técnica de control de congestión compatible con TCP. Asimismo, se relaja la restricción de usar números pares/impares para los puertos RTP/RTCP, admitiéndose la posibilidad de que las aplicaciones los seleccionen de manera explícita, de modo que sea posible el uso de aplicaciones RTP a través de cortafuegos y traductores de direcciones de red.

Capítulo 3

Control de Sesiones

La primera sección del presente capítulo describe las funciones comunes a los protocolos de control de sesiones multimedia, al tiempo que compara los dos estándares más extendidos en la actualidad desde varios puntos de vista, exponiendo las funciones específicas de ambos. Estos dos estándares son el conjunto de protocolos propuestos en la recomendación H.323 de la ITU-T y la arquitectura de control de sesiones propuesta por el grupo de trabajo MMUSIC del IETF, dentro de la cual destaca el protocolo SIP. Ambos estándares se describen con mayor detalle y por separado en las secciones 3.2 y 3.3, respectivamente. Posteriormente, se trata el protocolo de anuncio de sesiones multicast, SAP, perteneciente a la arquitectura del IETF. Por último, se analizan los protocolos de descripción de sesiones y negociación de capacidades desarrollados para la arquitectura de control del IETF.

3.1. Arquitecturas de control: H.323 y SIP

Tanto el conjunto de protocolos H.323 [37, 148, 24], propuesto por la ITU-T, como el protocolo SIP [133, 130, 131, 129, 126, 151], propuesto por el IETF, definen mecanismos de señalización para *establecer y terminar llamadas*, así como otras funciones de *control de conferencia, negociación de capacidades y servicios adicionales* sobre redes de conmutación de paquetes.

SIP se ha diseñado con posterioridad con la pretensión de que presente dos ventajas frente a H.323:

- Mayor flexibilidad para incorporar nuevas funciones.
- Implementación más fácil de realizar y depurar.

Mientras que SIP es un protocolo ligero similar a otros desarrollados previamente por el IETF y de amplia extensión, como HTTP, H.323 sigue un esquema similar al sistema tradicional de conmutación de circuitos, basándose en la señalización de la red digital de servicios integrados, ISDN[1], y otras recomendaciones de la serie H.

Las arquitecturas en las que se basan H.323 y SIP (véase las secciones 3.2 y 3.3, respectivamente) son sustancialmente distintas; sin embargo, al comparar la evolución de ambos estándares durante los últimos años [37], se extrae la conclusión de que, a medida que se definen nuevas ampliaciones a

[1]Construida tomando como base el protocolo Q.931.

SIP y aparecen nuevas versiones de H.323 cada vez se diferencian menos en cuanto a las funciones y posibilidades que ofrecen.

Un factor común a ambas arquitecturas es el protocolo de transporte utilizado para el intercambio de datos multimedia durante las sesiones que, aunque no está prefijado, en la práctica es RTP, sin que haya ninguna alternativa hasta la fecha.

Como precondiciones generales para una arquitectura que debe dar soporte, al menos, a un sistema global de telefonía que coexista con y sustituya finalmente a la red telefónica tradicional se deben tener en cuenta las siguientes [155, 150, 149, 134, 24, 76]:

- Escalabilidad hasta un gran número de usuarios de todo el mundo así como un gran número de llamadas activas de forma simultánea, del orden de millones.
- Permitir gestionar la red, de modo que sea posible aplicar políticas de control y tarificación.
- Proporcionar métodos de selección de calidad de servicio.
- Interoperabilidad entre diferentes fabricantes, protocolos y versiones de protocolos.
- Facilidad de ampliación.

Bajo estas restricciones, la arquitectura debe proporcionar mecanismos para realizar las siguientes funciones:

- *Negociación y selección de características de cada sesión*, en especial los formatos y la información multimedia que se desea y es posible intercambiar. Esta función se debe realizar tanto durante la etapa de establecimiento de la sesión como durante el transcurso de ésta.
- *Gestión de participantes*, para incorporar y dar de baja usuarios.
- *Localización de usuarios y traducción de direcciones*, lo que requiere un procedimiento de adaptación de diversos tipos de direcciones, como números de teléfono tradicionales, direcciones de correo-e, correo de voz o páginas web.

A continuación se compara, a grandes rasgos, las arquitecturas de control de sesiones multimedia H.323 y SIP, atendiendo a criterios de complejidad, ampliabilidad, escalabilidad, servicios ofrecidos y funcionamiento.

3.1.1. Complejidad

Como se destaca en [139], la complejidad de la recomendación H.323 es muy superior a la del protocolo SIP. Mientras que la especificación base de la primera supera las 700 páginas (sin contar la sintaxis), el protocolo SIP, junto con sus primeras ampliaciones y los protocolos de descripción de sesiones en los que se apoya, queda especificado en alrededor de 130 páginas. Las 37 cabeceras de SIP, todas con un reducido número de valores y parámetros, contrastan asimismo con los cientos de elementos definidos en la recomendación H.323.

Dado que SIP es un protocolo de codificación textual, es posible realizar analizadores y generadores de mensajes mediante scripts en lenguajes como Perl o Tcl, facilitando su integración con los

entornos web. Sin embargo, H.323 requiere generalmente el uso de herramientas de generación de analizadores para sintaxis ASN.1 [179].

Asimismo, el establecimiento de llamadas H.323 supone la intervención de varios de los protocolos asociados, en un proceso en el que los elementos que intervienen deben mantener el estado de la sesión. Por el contrario, todas las operaciones del protocolo SIP se basan en el intercambio de peticiones y respuestas individuales. Este factor tiene una gran influencia en la escalabilidad de la arquitectura.

En general, los autores del protocolo SIP han optado por prescindir de muchas de las opciones contempladas en la recomendación H.323, que añaden complejidad sin aportar ventajas, y son en algunos casos redundantes [148].

3.1.2. Ampliabilidad

La ampliabilidad es un factor clave para el éxito de los protocolos de sesiones multimedia, dado que al regular servicios de uso muy extendido, están sujetos a futuras necesidades de ampliación procedentes de muy diversas fuentes y campos, al tiempo que es crucial que los dispositivos diseñados para distintas versiones y por distintos fabricantes sean compatibles entre sí en la ejecución de un conjunto mínimo de funciones.

Los autores de SIP [148, 133] han tomado como precedente la evolución de dos protocolos muy extendidos: HTTP y SMTP. Este hecho les ha llevado a prever mecanismos para garantizar la compatibilidad entre diferentes versiones del protocolo, así como métodos de ampliación basados en el registro de nuevas características a través de la IANA.

En la especificación de H.323 también se reservan algunos campos para futuras ampliaciones específicas del fabricante; sin embargo, éstas sólo se pueden realizar en aquellas partes donde se ha previsto de forma explícita. Además, no existen mecanismos para determinar qué ampliaciones admite un dispositivo, por lo que el uso de estas ampliaciones conlleva riesgos de incompatibilidad entre distintos fabricantes.

En cuanto al uso de formatos multimedia estandarizados, las sesiones controladas mediante SIP pueden emplear de forma estandarizada cualquier formato multimedia registrado a través de la IANA, utilizándose para ello una descripción en formato SDP (véase la sección 3.6) o cualquier otro formato, como SMIL. En contraste con este procedimiento plenamente abierto, cuyos resultados son incuestionables, las sesiones H.323 sólo pueden emplear formatos registrados a través de una entidad central, la ITU-T, que hasta el presente sólo ha estandarizado formatos promovidos por la propia entidad, la mayoría de los cuales son de uso limitado por problemas de patentes.

Con respecto a la modularidad del sistema, el protocolo SIP comprende funciones de señalización básica, localización y registro de usuarios. Otras funciones, como garantía de calidad de servicio, descripción de contenidos, servicios de directorio y control de conferencia se delegan en otros protocolos. De este modo *cada protocolo empleado en una arquitectura SIP es plenamente intercambiable*, mientras que la arquitectura establecida por la recomendación H.323 define un conjunto de componentes acoplados que cubren un mayor rango de funciones.

3.1.3. Escalabilidad

Dado el gran número de sesiones que pueden llegar a procesar los componentes de la infraestructura telefónica de Internet, la simplicidad de esta gestión es clave. En este aspecto, SIP no requiere mantener estados en elementos intermedios y utiliza fundamentalmente conexiones UDP o de otros protocolos que requieren menos recursos que TCP, como SCTP. Por el contrario, H.323 requiere que los elementos intermedios mantengan el estado a lo largo de toda la sesión, y sólo emplea UDP a partir de la versión 3, debiendo mantener la compatibilidad con versiones previas plenamente basadas en conexiones TCP.

Tanto H.323 como SIP admiten sesiones con múltiples participantes y transporte multicast. No obstante, a diferencia de la especificación de H.323, la especificación de SIP no define ninguna entidad central para coordinar las sesiones, estableciendo un modelo de control plenamente distribuido. Además, al basarse fundamentalmente en UDP, SIP puede distribuir mensajes en modo multicast. Por ello, a igualdad de recursos disponibles, el número de participantes en una sesión controlada mediante SIP puede llegar a ser mayor que en una sesión H.323.

La versión inicial de H.323 se diseñó para utilizarse en redes de área local, por lo que, aunque las últimas versiones han incorporado métodos de localización de usuarios, carece de algunos mecanismos, como prevención de bucles. Sin embargo, SIP se ha diseñado para redes de área extensa, por lo que cuenta con mecanismos flexibles de localización y registro de usuarios definidos con suficiente generalidad, como búsqueda a través de un número indefinido de servidores proxys, búsqueda en paralelo y ramificación.

3.1.4. Servicios ofrecidos

A medida que SIP y H.323 evolucionan de forma competitiva, el conjunto de servicios que ambos ofrecen va pareciéndose más [148, 37]. Como excepción, H.323 proporciona servicios de negociación de contenidos mucho más completos que los que pretende proporcionar SIP, permitiendo especificar complejas combinaciones de formatos multimedia admitidos por los dispositivos, junto con diversos parámetros de estos formatos. SIP prevé únicamente el intercambio de conjuntos de formatos admitidos. Sin embargo, los mecanismos adicionales proporcionados por H.323 rara vez son necesarios y, en la práctica, se usa sólo un subconjunto de ellos equivalente a los mecanismos de SIP.

3.1.5. Funcionamiento

Establecer una llamada con H.323 versiones 1 o 2 requiere establecer, previamente a la conexión H.323, conexiones TCP para H.225.0 y H.245. Por el contrario, el establecimiento de una llamada SIP sólo requiere un intervalo de ida y vuelta de paquetes en el caso más común, puesto que se realiza mediante un mensaje transportado generalmente sobre UDP. A partir de la versión 3, H.323 permite iniciar llamadas mediante UDP. No obstante, el establecimiento de llamadas es aun así normalmente más rápido con SIP [37]; sin embargo, H.323 puede solventar con mayor rapidez situaciones de error. No obstante, con la progresiva incorporación de SCTP como protocolo de transporte para señalización SIP, se espera mejorar las prestaciones en situaciones con un elevado porcentaje de errores en la transmisión.

El proceso de *finalización de llamadas* se lleva a cabo mediante sendos mensajes RELEASE

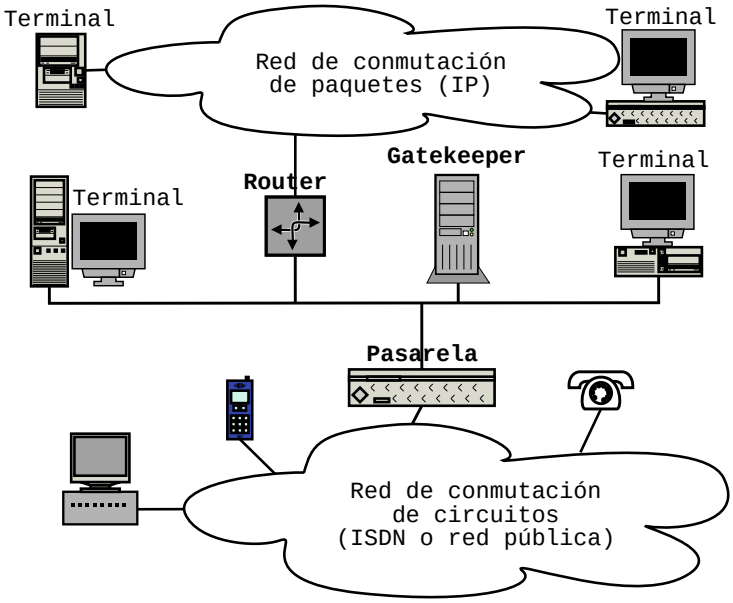


Figura 3.1: Esquema de la arquitectura H.323

COMPLETE y BYE, para H.323 y SIP, respectivamente.

En cuanto al *control durante la llamada*, tanto SIP como H.323 proporcionan servicios suplementarios que permiten modificar y comprobar el estado de una sesión durante su desarrollo, como desconexión provisional, identificación de llamadas, llamadas en espera o redirección y transferencia de llamadas.

Como excepción, SIP proporciona un servicio de establecimiento de llamada por parte de terceros que actualmente no está contemplado en H.323. Este servicio permite que un usuario establezca una llamada entre otros dos usuarios, siendo útil para operadores de telefonía, telemarketing o llamadas a través de secretarios.

3.2. Arquitectura H.323

Los sistemas basados en señalización H.323 [37, 148, 24] se articulan en torno a cuatro tipos de componentes: terminales, pasarelas, gatekeepers y unidades de control multipunto, cuya interacción se esquematiza en la figura 3.1.

Los mensajes H.323 siguen un esquema de codificación binaria, especificada mediante sintaxis ASN.1 y reglas de codificación de paquetes (PER). La tabla 3.1 resume las recomendaciones incluidas en el estándar H.323. A continuación se describe la función de cada uno de los elementos de un sistema H.323.

Terminales. Son los clientes situados en redes IP. Se comunican bidireccionalmente y en tiempo real con el resto de elementos de los sistemas H.323. Deben ser capaces de comunicarse mediante los protocolos de control H.225 y H.245, así como de transmitir datos en tiempo real sobre RTP/RTCP. Asimismo, pueden contener varios codificadores y decodificadores de formatos de sonido y vídeo, de entre los cuales, el formato de sonido G.711 (PCM) es obligatorio, con objeto de garantizar una mínima compatibilidad entre todos los terminales.

Recomendación	Función
H.245	Funciones de control
H.225.0	Establecimiento y control de conexión
H.332	Gestión de conferencias de gran tamaño
H.235	Seguridad (privacidad, autenticación, etc.)
H.246	Interoperación con servicios basados en conmutación de circuitos
H.450.1, 2 y 3	Servicios adicionales (transferencia y redirección de llamadas, etc.)
H.26x	Formatos de vídeo (H.262 y H.263)
G.7xx	Formatos de sonido (G.711, G.723, G.729, etc.)

Cuadro 3.1: Recomendaciones incluidas en la especificación H.323

Pasarelas. Encargadas de conectar redes de conmutación de paquetes con redes de conmutación de circuitos, ya sean públicas o privadas. Deben ser capaces de establecer y controlar llamadas en ambos tipos de redes, traduciendo los distintos formatos y procedimientos de transmisión. Su presencia, del mismo modo que la conexión a redes de conmutación de circuitos, es opcional.

Gatekeepers. Su presencia en un sistema es asimismo opcional; sin embargo, cuando están presentes deben desempeñar cuatro funciones:

1. Control de admisión.
2. Control de ancho de banda.
3. Traducción de direcciones alias o números de teléfono a direcciones de transporte, y viceversa.
4. Gestión de zonas.

Cuando existe un gatekeeper en el sistema, todos los dispositivos deben registrarse y obtener permiso de él antes de realizar llamadas.

Unidades de control multipunto (MCU). Permiten que se establezcan conferencias entre tres o más elementos. Generalmente constan de un controlador multipunto y un número variable de procesadores multipunto.

En terminología H.323, un canal es equivalente a una conexión de nivel de transporte. La recomendación H.323 especifica cuatro tipos de canales:

- *Canal de señalización de llamada*. Por el cual circula la información relativa al control de llamadas y servicios adicionales. El protocolo de control utilizado se especifica en las recomendaciones H.450 y H.225.0
- *Canal de control H.245*. Dedicado al control de la transferencia de información multimedia y la negociación de capacidades.
- *Canal de control RAS*. Destinado a la comunicación entre los terminales y sus guardabarreras asociados. El protocolo RAS (registro, admisión y estado) está especificado en la norma H.225.0 y permite que los terminales se registren ante un gatekeeper y soliciten permiso para establecer llamadas.

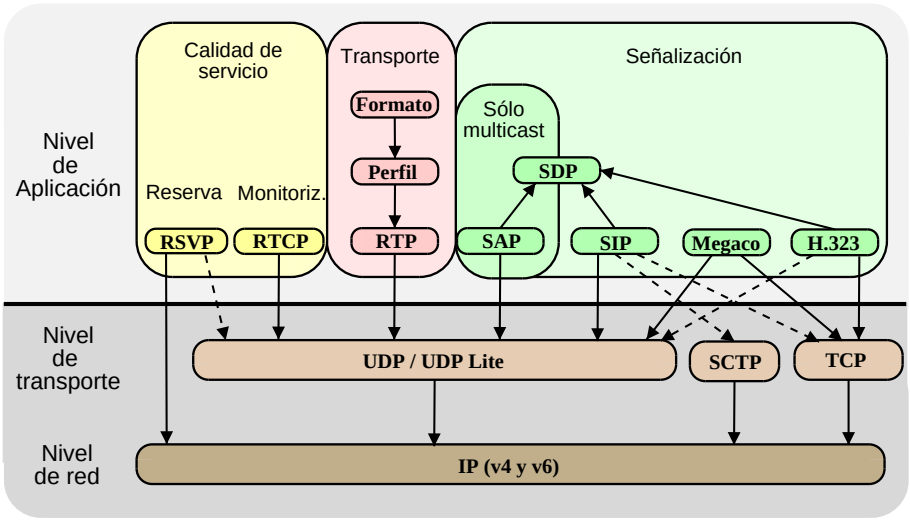


Figura 3.2: Protocolos para sesiones multimedia en Internet

- *Canales lógicos para información multimedia*. Por donde circulan los datos de sonido, vídeo y cualquier otro medio. Cada uno de ellos se transporta mediante una sesión RTP.

Las versiones 1 y 2 de H.323 asocian el canal de señalización de llamadas a protocolos de transporte fiable, como TCP. Sin embargo, la versión 3, siguiendo al protocolo SIP, admite el uso de protocolos no fiables (UDP) para este canal. Para información del canal RAS y los canales lógicos se utilizan asimismo protocolos de transporte no fiable, mientras que el canal de control H.245 está asociado a protocolos de transporte fiable.

3.3. Arquitectura de Sistemas Multimedia del IETF

La arquitectura de sesiones multimedia del IETF [35, 56, 139], a diferencia de la arquitectura H.323, está constituida por un amplio conjunto de protocolos independientes, cada uno de los cuales cumple funciones complementarias. La figura 3.2 recoge los protocolos inscritos en esta arquitectura, que engloba dos protocolos de control específicos de Internet: SIP y SAP. SIP cumple las funciones de *establecimiento de sesiones mediante invitación así como de modificación y finalización de sesiones*. SAP es un *protocolo de anuncio de sesiones desarrollado para entornos multicast*. Adicionalmente, se ha definido el protocolo MEGACO [53, 36][2] para el control de las pasarelas o sistemas de interconexión entre redes de diferentes tipos, en especial entre redes de conmutación de paquetes, como Internet, y redes de conmutación de circuitos, como la red telefónica.

En la figura se aprecia la presencia de H.323 entre los protocolos de señalización, puesto que, al igual que cualquier otro protocolo que cumpla los requisitos de control de sistemas multimedia, puede sustituir a SIP en las funciones de señalización. Asimismo, es posible utilizar de forma combinada ambos protocolos, por ejemplo, estableciendo sesiones mediante el protocolo SIP y llevando a cabo el control de la sesión mediante los protocolos de la recomendación H.323.

La arquitectura de control del IETF, basada en SIP, establece un modelo de sesiones descentralizado; es decir, no contempla la existencia de un registro central para los participantes en cada sesión,

[2]Estandarizado por la ITU-T en la recomendación H.248.

sino que éstos comparten el concepto de sesión simplemente como una abstracción en común. SIP forma parte de una arquitectura integrada y estandarizada vinculada a otros protocolos de Internet, como el correo electrónico y HTTP.

Nótese que las funciones de *garantía de calidad de servicio*, ya sea mediante reserva de recursos, o mediante servicios integrados o diferenciados [35], quedan delegadas en otros protocolos, preferentemente RSVP [18, 9] y Diffserv [14], respectivamente. Asimismo, el protocolo de *transporte en tiempo real* preferente es RTP, junto con RTCP, que desempeña algunas de las funciones de control de sesiones incluidas en la recomendación H.323. Tanto SIP como RTP son independientes del protocolo de transporte subyacente.

Las funciones comentadas quedan completadas con los siguientes servicios:

- *Servicios de directorio*, delegados en el protocolo LDAP [195].
- *Búsqueda de pasarelas*, proporcionados por el protocolo TRIP, desarrollado recientemente a tal efecto [81], pero que se ha diseñado sin dependencias de ningún protocolo de señalización concreto, por lo que se puede utilizar tanto con SIP como con H.323.
- *Autenticación, autorización y contabilidad*, que pueden desempeñar los protocolos RADIUS [25] y DIAMETER [21].

En las siguientes secciones se describe la arquitectura de los sistemas multimedia controlados por SIP; la sección 3.5 describirá el protocolo SAP.

3.3.1. Arquitectura SIP

El protocolo SIP, cuya operación se basa en mensajes de petición y respuesta, reutiliza muchas de las reglas de codificación, códigos de error y campos de cabecera de HTTP. Las funciones de control de llamadas (redirección, transferencia, cambio de formatos y codificación, etc.) que proporciona están integradas, por tanto, con la infraestructura web como los sistemas de programación que utilizan la interfaz CGI. El uso de tipos MIME para describir los contenidos tratados por los mensajes SIP hace posible, por ejemplo, devolver cualquier tipo de contenido web ante un mensaje de inicio de llamada. Por ello, mediante aplicaciones web [127] es posible programar servicios de telefonía basada en SIP, integrando, por ejemplo, aplicaciones interactivas con respuesta de voz (IVR) en sitios web.

Del mismo modo que RTP, SIP es independiente del protocolo subyacente, ya sea UDP, TCP, AAL5, X.25 o frame relay. Las sesiones multimedia controladas por SIP pueden constar de varias sesiones RTP sin que todos los participantes tengan por qué participar en todas las sesiones RTP.

SIP es un protocolo cliente-servidor de señalización al que se pueden añadir nuevos métodos y capacidades, cuyo registro se debe realizar a través de la IANA [72]. Igualmente, define respuestas de rechazo ante elementos desconocidos, de modo que aplicaciones con diferentes grados de implementación y versiones de SIP puedan interoperar en la medida de lo posible.

Los dos tipos de elementos presentes en la arquitectura SIP son los *agentes de usuario (UA)* y los *servidores*.

Los UA constan, a su vez, de dos componentes: los *agentes de usuario clientes (UAC)* y los *agentes de usuario servidores (UAS)*. Ambos se encuentran en todos los agentes de usuario, permitiendo la comunicación entre diferentes agentes de usuario mediante peticiones y respuestas de tipo

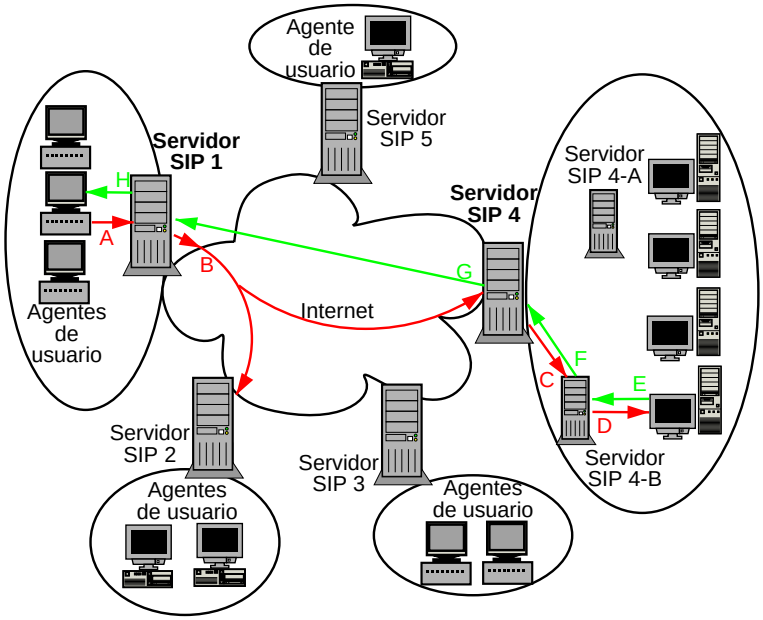


Figura 3.3: Esquema de un sistema SIP en Internet

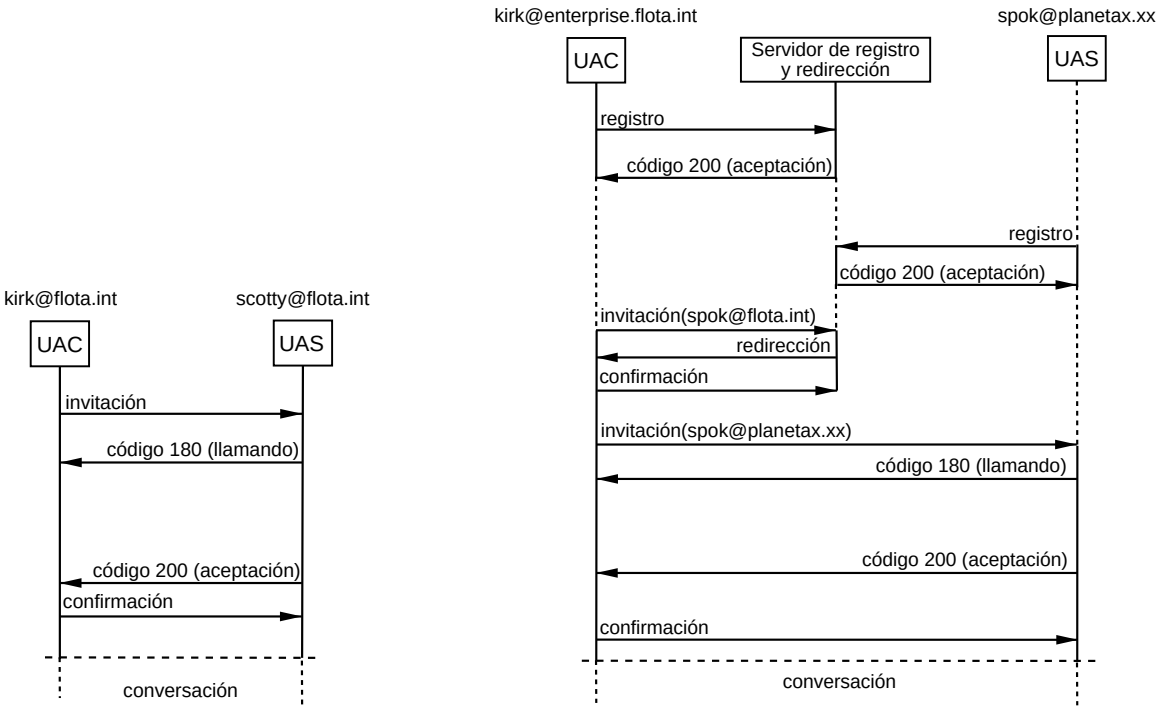
cliente-servidor. Los UAC tienen como misión el envío de peticiones SIP, mientras que los UAS están encargados de atender tales peticiones y remitir las correspondientes respuestas.

Los servidores SIP pueden ser de tres tipos no excluyentes: *servidores de redirección, de registro* y *proxys*. Aunque una llamada básica se puede realizar con SIP sin que intervengan servidores, las funciones avanzadas del protocolo no se pueden llevar a cabo sin su participación. La relación entre estos componentes [150, 24] se resume en la figura 3.3.

Las funciones básicas de los servidores SIP son la localización de usuarios y la resolución de nombres. Puesto que generalmente los agentes de usuario clientes no conocen la dirección IP del destinatario de una llamada, sino su nombre (en forma de dirección de correo-e) o un número de teléfono, necesitan enviar en primer lugar un mensaje de invitación al servidor correspondiente al nombre o número para que localice al destinatario. El servidor puede conocer la dirección del destinatario o recurrir a otros servidores para continuar la búsqueda. Cuando las llamadas se redirigen, la ruta seguida se registra en los mensajes SIP, de modo que a la hora de generar respuestas se pueda conocer el camino de retorno hasta el origen del mensaje inicial. La figura 3.3, adaptada a partir de [150], esquematiza la relación entre agentes de usuario y servidores.

Con objeto de proporcionar la mayor movilidad posible (véase la sección 3.3.3) los servidores SIP pueden emplear cualquier medio para localizar agentes de usuario, ya sea el sistema DNS, ejecutando programas auxiliares o accediendo a bases de datos. En algunos casos, un servidor puede determinar que es posible acceder a un destinatario a través de varios servidores, pudiendo bifurcar la búsqueda, con la posibilidad de unificar las múltiples respuestas recibidas antes de devolverlas al agente que inició la llamada.

Los servidores pueden mantener el *estado de las llamadas* a dos niveles de detalle o no mantener estado alguno. Al igual que en la red telefónica convencional, pueden mantener el estado completo de cada llamada; sin embargo, este comportamiento es opcional, puesto que limitaría la escalabilidad de los sistemas SIP. El modo de funcionamiento recomendado y más frecuente es que los servidores mantengan únicamente el estado de cada transacción por separado. En la sección 3.4.2 se tratan las



(a) Sin servidor intermediario (b) Con servidor intermediario

Figura 3.4: Esquema de una transacción de establecimiento de llamada con SIP

transacciones SIP, que, a grandes rasgos, se pueden definir como *el conjunto de peticiones y respuestas intercambiadas desde que un agente de usuario cliente envía una petición hasta que recibe una respuesta definitiva*. Por este motivo, si los servidores sólo mantienen el estado de cada transacción por separado, no tienen conocimiento de las llamadas existentes en un cierto instante, sino sólo de cada una de las peticiones y respuestas asociadas que la componen. De este modo, los servidores no tienen que mantener una máquina de estados para cada llamada, constituyendo así un sistema altamente escalable. Asimismo, el comportamiento de los servidores SIP en cuanto al mantenimiento de estados se puede modificar de forma dinámica en función de las circunstancias.

La figura 3.4 esquematiza una transacción de establecimiento de llamada, tanto para el caso de una comunicación directa entre dos agentes, como cuando interviene un servidor de registro y redirección. En 3.4(a) el UAC del usuario que inicia la llamada envía una petición de invitación directamente al destinatario, que, finalmente acepta la llamada. En 3.4(b), el UAC del usuario que inicia la llamada remite la invitación a su servidor de redirección, que le proporciona la dirección del destinatario. Tras esta transacción, el UAC puede ponerse en contacto directamente con el UAS del usuario destinatario. Previamente, ambos agentes de usuario deben haberse registrado en el servidor. Nótese que, por simplicidad, se supone que ambos agentes se registran en el mismo servidor, aunque SIP contempla escenarios mucho más complejos.

En los sistemas SIP, tanto los mensajes de señalización como la información multimedia intercambiada en una sesión pueden transmitirse en unicast o multicast. El caso correspondiente a señalización e información multimedia transmitida en multicast es equivalente a las sesiones iniciadas por medio del protocolo SAP (véase la sección 3.5). Sin embargo, en la práctica, SIP se ha usado hasta ahora

fundamentalmente para llamadas con dos participantes. El uso generalizado de SIP para conferencias con cualquier número de participantes presenta dificultades adicionales que requieren un esquema general para llamadas con múltiples participantes [97].

Los nombres de los usuarios de los sistemas SIP tienen forma de dirección de correo-e, y corresponden a los siguientes casos:

- *usuario@dominio*, donde *dominio* es un nombre de dominio completo.
- *usuario@equipo*, donde *equipo* es el nombre de la máquina que utiliza *usuario*
- *usuario@dirección_ip*, donde *dirección_ip* es la dirección IP del dispositivo utilizado por *usuario* para comunicarse
- *número_teléfono@pasarela*, correspondientes a equipos de la red telefónica pública con número *número_teléfono* y accesibles mediante la pasarela *pasarela*.

Estos identificadores pueden hacer referencia a una persona, a la primera persona disponible de un grupo o a un grupo completo. Las direcciones de tipo correo-e tienen la ventaja adicional de permitir el uso de servicios de directorio, como LDAP, para hallar la correspondencia entre el nombre de usuario y la dirección *usuario@dominio*.

El uso del sistema DNS permite localizar los servidores con peticiones de búsqueda de servicios (SRV [54]), intercambio de correo (MX) o registro de direcciones (A). Este procedimiento proporciona un mecanismo escalable para localizar los servidores a través de los cuales se puede acceder a los usuarios del dominio que controlan.

Asimismo, es posible especificar URL SIP de la forma *sip:usuario@dominio*, de modo que se pueden emplear en la infraestructura web, de forma similar a los URL *mailto* [65], para iniciar una llamada telefónica [181] ante la pulsación de un control. con algo más de generalidad, los URL SIP tienen la siguiente forma:

```
sip:[usuario[:clave]@]equipo[:puerto]
```

La sintaxis completa de los URLs SIP es mucho más compleja [133], permitiendo formar identificadores como el siguiente:

```
sip:+34-111-11-11-11@pasarela.es;user=phone;transport=sctp
```

donde se hace referencia al número de teléfono *111-11-11-11*, con prefijo internacional *+34*, accesible a través de la pasarela *pasarela.es*. El parámetro *user* con valor *phone* indica explícitamente que *+34-111-11-11-11* hace referencia a un número de teléfono [181] en lugar de a un nombre de usuario. El parámetro *transport* indica el protocolo de transporte que ha de usarse para iniciar una llamada a la dirección, en este caso, SCTP.

3.3.2. Tipos de agentes de usuario y servidores

Generalmente los servidores SIP actúan de forma simultánea como varios tipos de servidores. Gracias a una infraestructura de servidores SIP, es posible gestionar las llamadas de forma distribuida entre equipos personales, equipos de proveedores de servicios y pasarelas corporativas, con la consiguiente flexibilidad y control por parte del usuario, que puede mantener la privacidad de sus datos personales en todo momento. Asimismo, un mensaje SIP puede pasar por un número indeterminado

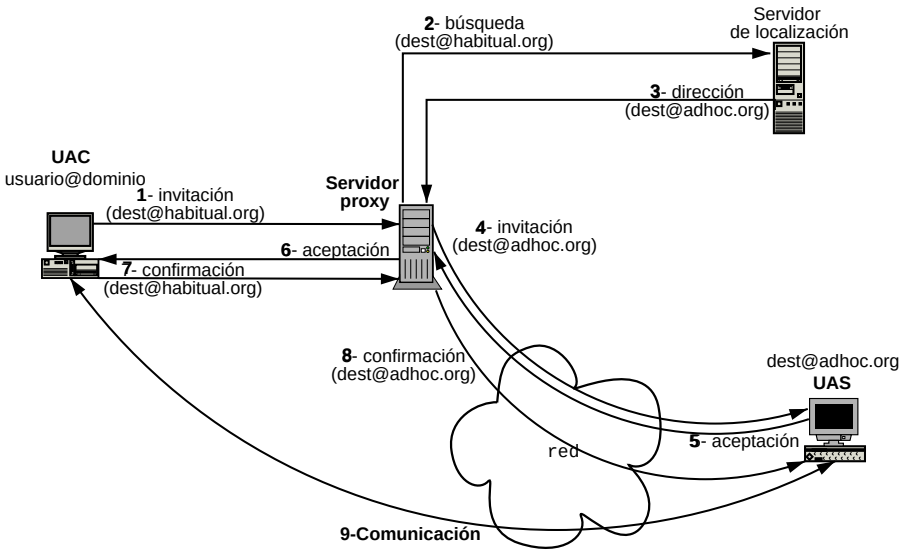


Figura 3.5: Esquema de funcionamiento de un proxy SIP

de servidores desde que un agente de usuario cliente lo envía hasta que llega al agente de usuario servidor destinatario. Los tipos de agentes de usuario y servidores SIP definidos hasta el momento son los siguientes:

Agentes de usuario clientes (UAC). Son las aplicaciones que inician las transacciones.

Agentes de usuario servidores (UAS). Son las aplicaciones servidoras que notifican al usuario la recepción de peticiones SIP y remiten las respuestas dadas por los usuarios al UAC que envió la petición.

Servidores de registro. El uso más común de estos servidores es registrar un dispositivo después de su arranque, de modo que cuando lleguen invitaciones destinadas a él, los servidores SIP puedan proporcionar su dirección. Generalmente, los proxys actúan como servidores de registro para todos los dispositivos representados por ellos. Para ello, aceptan y procesan peticiones de registro.

Se contempla la existencia de un tiempo máximo de validez de cada registro, definible por el servidor, tras el cual se debe renovar el registro. Asimismo, existen mecanismos para cancelar todos los registros contenidos en un servidor de registro.

Proxys. Su función es similar a la de los proxys HTTP o los agentes de transferencia de mensajes SMTP: recibir solicitudes y decidir a qué otro servidor se deben remitir, alterando además algunos campos de la solicitud. Por tanto, actúan como intermediarios en las transacciones que procesan.

La figura 3.5 esquematiza el funcionamiento de estos servidores. Téngase en cuenta que el elemento al que el proxy reenvía la petición puede ser tanto otro servidor proxy, como un servidor de redirección o un agente de usuario servidor, si bien, por simplicidad se presente el último caso.

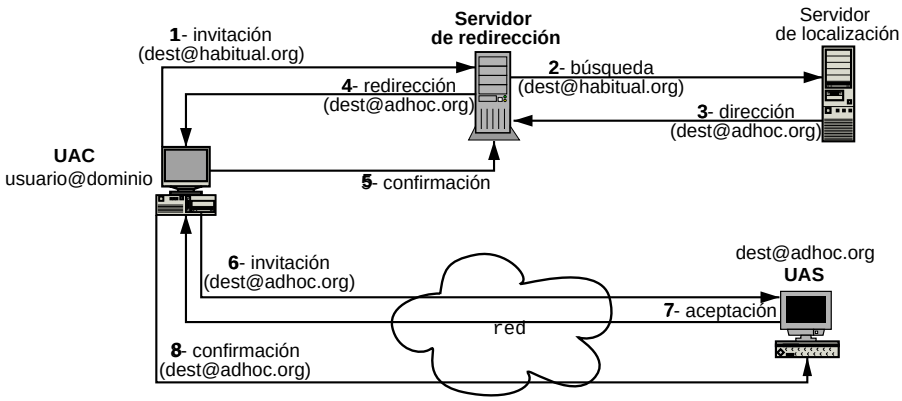


Figura 3.6: Esquema de funcionamiento de un servidor SIP de redirección

Nótese cómo el servidor proxy puede recurrir a un servidor de localización para determinar la dirección en la que actualmente está disponible el destinatario, *dest@adhoc.org*, correspondiente a la dirección original, (dest@habitual.org), indicada por el UAC que inició la llamada.

Los proxys son, al igual que en el caso del protocolo HTTP, particularmente útiles como representantes de salida/entrada de y a redes corporativas, proporcionando servicios de búsqueda de direcciones, control de cortafuegos y gestión de políticas corporativas. Asimismo, pueden cumplir funciones de control de salida a pasarelas para redes telefónicas tradicionales.

Servidores de redirección. A diferencia de los proxys, no inician transacciones, sino que, cuando reciben solicitudes desde un agente de usuario cliente, remiten al mismo agente un mensaje indicando el o los servidores con los que debe ponerse en contacto, en un procedimiento similar al de búsqueda iterativa del sistema DNS. Asimismo, a diferencia de los agentes de usuario servidores, no aceptan llamadas.

La interacción de este tipo de servidores con los agentes de usuario se describe en la figura 3.6, asimilable a la figura 3.4(b), de la página 36.

Normalmente, los servidores de redirección gestionan mayor número de mensajes que los proxys, pero con menores necesidades de procesamiento. Nótese que, puesto que en sesiones controladas por SIP la redirección se realiza mediante mensajes SIP, y no mediante otros métodos como LDAP [195, 184], RWhois [192], o Whois++ [52, 39], las respuestas se pueden generar con gran flexibilidad, modificándose en función de parámetros tales como la hora del día o el origen, urgencia de la llamada o cualquier otro criterio específico aplicado por el servidor SIP.

3.3.3. Movilidad

El protocolo SIP, gracias a sus mecanismos de redirección y su esquema de nombres, puede proporcionar servicios de movilidad en cuatro aspectos [152]:

- Movilidad personal. De modo que se pueda asociar a dispositivos, como teléfonos celulares, dispositivos PDA, ordenadores personales y teléfonos Ethernet, una misma dirección personal

cuya búsqueda proporcione la dirección de un dispositivo u otro en función de las circunstancias.

- Movilidad de servicios. Mantener servicios como libros de direcciones, botones asociados a funciones especiales, o preferencias entre distintas localizaciones.
- Sesiones. Trasladar sesiones activas entre distintos terminales a medida que el usuario se traslada, por ejemplo, por su casa.
- Movilidad de los terminales, es decir, poder utilizar un terminal que se desplaza a lo largo de diferentes subredes.

El último tipo de movilidad debe ser admitido por protocolos subyacentes, como IEEE 802.11 o IP móvil y requiere en principio el uso de protocolos adicionales como DHCP y DNS dinámico. Sin embargo, este enfoque tiene el inconveniente de que se depende de que los proveedores de servicios de los usuarios proporcionen una dirección IP permanente y soporte específico. La flexibilidad de la arquitectura SIP permite un enfoque alternativo más factible en las redes actuales, consistente en detectar cambios de dirección IP en el nivel de aplicación, y actualizar el registro del agente de usuario en los servidores de localización SIP (procedimiento válido antes de iniciar llamadas) o enviar una segunda o posterior invitación si el cambio de dirección se produce durante una llamada.

Cabe destacar que, del mismo modo que SIP ha tomado muchos de sus elementos de la infraestructura web existente, gran parte de los nuevos servicios web en desarrollo en la actualidad son asimilables a los servicios de movilidad contemplados en la arquitectura de los sistemas SIP.

3.3.4. Seguridad: privacidad y autenticación

Con objeto de garantizar la privacidad de los participantes en sesiones, SIP admite tres modos de cifrado, basando en criptografía de clave pública (en especial, en PGP [70]) los mensajes de señalización:

- Cifrado, en cada enlace por separado, del campo de la cabecera que contiene la información de la ruta seguida por los mensajes.
- Cifrado, en cada enlace por separado, de los campos de la cabecera que contienen información acerca de los participantes.
- Cifrado de extremo a extremo del cuerpo del mensaje y de algunos campos de la cabecera especialmente relevantes, de modo que ningún elemento intermedio tenga acceso a ellos.

El mayor problema que presenta este esquema es el acceso al campo de la cabecera que codifica la ruta seguida por el mensaje, requerido por los proxys, por lo que su cifrado requiere la cooperación de todos los proxys involucrados en el envío de un mensaje.

Los mecanismos criptográficos contemplados para garantizar la privacidad se pueden utilizar para autenticar los mensajes; sin embargo, no se garantiza su integridad. SIP reutiliza y amplía los campos de autenticación del protocolo HTTP, añadiendo firmas criptográficamente fuertes.

3.3.5. Programación

Como se describió en la sección 3.3.1, el protocolo SIP está plenamente integrado en la infraestructura web. Sin embargo, además de los procedimientos habituales basados en interfaz CGI o *servlets*, se ha desarrollado un lenguaje de programación específico para servicios de telefonía basados en SIP: CPL [93].

CPL es un lenguaje diseñado para permitir que los usuarios creen de forma sencilla, mediante interfaces gráficas, servicios de telefonía, generando programas que han de ejecutarse en servidores SIP. Aunque su desarrollo está ligado al de SIP, es un lenguaje independiente del protocolo de señalización utilizado. Asimismo, dispone de mecanismos que garantizan la seguridad a la hora de ejecutar programas CPL en servidores.

Es un lenguaje basado en XML [20, 32], con especial énfasis en estructuras de selección, tanto de direcciones (condiciones a comprobar sobre el origen, destinatario, tipo de dirección, usuario, etc), como de cadenas (asunto, organización, agente de usuario, etc.), como de periodos de tiempo (franja horaria, rangos de fechas, etc.).

Provee asimismo funciones para realizar y controlar búsquedas de usuarios, realizar acciones de representación, enviar correo-e, y acciones de señalización, como redirección de llamadas.

3.4. SIP

SIP [133, 130, 131, 129, 126, 151] es un protocolo de señalización de nivel de aplicación desarrollado para el establecimiento, modificación y finalización de sesiones multimedia. Su formato es textual. Por tanto, al igual que otros protocolos de Internet, como NNTP [84], RTSP [147], SMTP [122], FTP [121] y HTTP [45], se caracteriza por poderse procesar con facilidad y su ineficiencia en el uso de ancho de banda.

La sintaxis de SIP se especifica en formato ABNF [33], en contraste con los formatos binarios ASN.1 o XDR. El formato textual de SIP da lugar a implementaciones mucho más simples que las realizadas para formatos binarios característicos de protocolos como RTP [143], TCP [120], IP [119] o RSVP [18, 9]. Además, el uso ineficiente de ancho de banda es, por lo general, despreciable, puesto que el volumen de datos intercambiados en una sesión –al menos del orden de kilobits por segundo– suele ser muy superior al volumen de señales de control. Un intercambio de petición y respuesta medio requiere alrededor de 350 octetos, y todas las transacciones de una llamada normal (llamada, respuesta provisional, confirmación y cierre) consumen alrededor de 1500 octetos. Además, para los casos en los que un mensaje SIP pueda superar la unidad máxima de transferencia de la red (MTU), en [133] se define un formato alternativo más compacto que el habitual, basado en abreviar los nombres de los campos de las cabeceras.

Por tanto, a cambio de una pérdida mínima de eficiencia se consigue simplificar el desarrollo de aplicaciones SIP, que se pueden construir con comodidad con lenguajes como Perl o Tcl; asimismo, las peticiones y respuestas SIP son fáciles de interpretar, simplificándose así las tareas de depuración.

Del mismo modo que SDP, SIP admite el conjunto de caracteres ISO 10646 en la codificación UTF-8. Por tanto, la internacionalización de los mensajes SIP es mucho más directa que la de H.323 o SNMP, cuya sintaxis se especifica mediante ASN.1.

La estructura de SIP está basada en la de HTTP. No obstante, HTTP se fundamenta en un pro-

tocolo de transporte fiable, como TCP, mientras que el uso de un protocolo con control de flujo y congestión no es adecuado para un protocolo de señalización en tiempo real. Por ello, es posible basar una realización de SIP tanto en TCP como en UDP, estando recomendado el segundo caso. Recientemente se ha empezado a estudiar la posibilidad de utilizar otros protocolos, como SCTP, que se trató en la sección 2.1, o TLS [3].

Asimismo, las peticiones SIP pueden realizarse en entornos multicast, estableciendo así conferencias con múltiples participantes en las que las funciones de señalización y búsqueda de destinatarios se deben adecuar al entorno multicast. No obstante, el uso genérico de SIP para conferencias con múltiples participantes, ya sea en entornos multicast o unicast, requiere esquemas de llamadas específicos que están actualmente en proceso de definición [97, secciones 3 y 4.6])

Con objeto de evitar errores cometidos en la especificación inicial de HTTP, SIP se diferencia de este protocolo por prescindir de caminos relativos en las peticiones, es decir, usar siempre caminos absolutos[4], y carecer de mecanismos de ampliación. Estos mecanismos se sustituyen en SIP por los campos *Require* y *Request* de las cabeceras de las peticiones y respuestas, respectivamente. Mediante el campo *Require*, los clientes de usuario pueden especificar capacidades necesarias para participar en una sesión; aquellos servidores de usuario que desconozcan alguna opción especificada deben contestar con el código 401 (“Ampliación incorrecta”, véase la tabla 3.3, en la página 48), proporcionando una lista de campos *Unsupported* con todas las opciones desconocidas. Así, se pretende que no sean necesarias nuevas versiones de SIP, sino que todos los dispositivos SIP empleen la misma versión del protocolo, aunque con niveles de desarrollo y actualización diversos, sin que por ello dejen de ser compatibles.

SIP cumple cinco funciones en el establecimiento y finalización de sesiones multimedia: localización de usuarios, determinación de su disponibilidad, enumeración de las capacidades de su terminal, configuración de la llamada y gestión de la sesión (incluyendo transferencia y terminación de llamadas).

3.4.1. Llamadas

Cada llamada SIP tiene asociado un código único que identifica a *todos los participantes en una sesión multimedia invitados por un mismo agente*. Una llamada puede dar lugar a una rama de llamadas, identificada por la combinación del código de la llamada, el origen y el destinatario. Cada llamada dentro de una rama se distingue mediante un número de secuencia. Con objeto de que los proxys SIP sólo tengan que mantener el estado de cada respuesta individual, en lugar de la conferencia por completo, cada mensaje de señalización contiene el identificador de llamada.

Durante la fase de iniciación de llamadas, se realizan funciones que podemos dividir en dos bloques. Se puede optar por llevar ambos a cabo mediante SIP, o por combinar SIP con otro protocolo de señalización, como H.323:

- *Localización y determinación de disponibilidad del destinatario, y selección de terminal [131].* Las respuestas de los servidores SIP pueden contener una lista con la descripción de los distintos terminales a los que tiene acceso un destinatario, especificando asimismo las posibilidades y direcciones de cada uno de ellos.

[3] Para el cual, el identificador de protocolo *sip* usado en los URL SIP se sustituye por *sips*.
[4] De este modo, no hay problemas a la hora de definir equipos virtuales.

Especificación	Método	Función
RFC 3261 [133]	INVITE	Inicio de llamada/sesión
	ACK	Acuse de recibo final
	BYE	Terminar y transferir la llamada
	CANCEL	Cancelar búsqueda y llamada
	OPTIONS	Comprobación de capacidades del otro extremo
	REGISTER	Registro en servidores de localización
RFC 2976 [40]	INFO	Información durante la llamada
RFC 3265 [126]	SUBSCRIBE	Suscripción a una sesión de servicio
	UNSUBSCRIBE	Cancelación de suscripción
	NOTIFY	Notificación para suscriptores
RFC 3262 [130]	PRACK	Acuse de recibo provisional

Cuadro 3.2: Métodos del protocolo SIP

- *Negociación de tipos de datos y de su codificación [129].* El mensaje de invitación de SIP contiene una lista de los tipos de datos y codificaciones que se pretende utilizar en una sesión. En el caso de sesiones de dos participantes, el destinatario de la invitación proporciona en su respuesta un subconjunto de la lista de tipos original, indicando los tipos que está dispuesto a utilizar, por lo que la negociación de contenidos se realiza con la mayor rapidez posible. Este modelo de negociación, conocido como *oferta/respuesta* se ha definido por el momento para SDP, aunque es generalizable para otros formatos de descripción. Para el caso de conferencias con múltiples participantes, el organizador puede utilizar mensajes de tipo OPTIONS (véase la tabla 3.2) con objeto de conocer los medios disponibles antes de iniciar la llamada.

3.4.2. Transacciones

Una transacción SIP se realiza mediante el intercambio de mensajes entre un cliente y un servidor, prolongándose desde que el cliente remite el primer mensaje hasta que el servidor devuelve al cliente una respuesta de tipo no provisional, véase la tabla 3.3, página 48. Es decir, cada transacción consta de una petición y las respuestas generadas como consecuencia de la petición, que han de tener el mismo valor para los parámetros *From*, *To*, *Call-ID* y *CSeq*. De este modo, se pueden agrupar en la misma transacción. En la figura 3.4, de la página 36, se esquematizan dos casos concretos de transacción de establecimiento de llamada.

Cuando las peticiones y respuestas se transportan sobre TCP, todas las asociadas a una transacción fluyen por una misma conexión TCP, que se puede reutilizar para varias transacciones. Cuando el protocolo de transporte usado es UDP unicast, el campo *Via* de las cabeceras de los mensajes SIP contiene la dirección de transporte a la cual remitir las respuestas. Si los mensajes se transmiten en multicast, las respuestas se remiten a la dirección y puerto de origen.

El diseño de SIP contempla la definición progresiva de nuevos tipos de transacciones, complementarios a los tipos básicos definidos en la especificación inicial, de modo que se puedan añadir nuevos servicios cuya gestión es transparente para servidores que no los implementan.

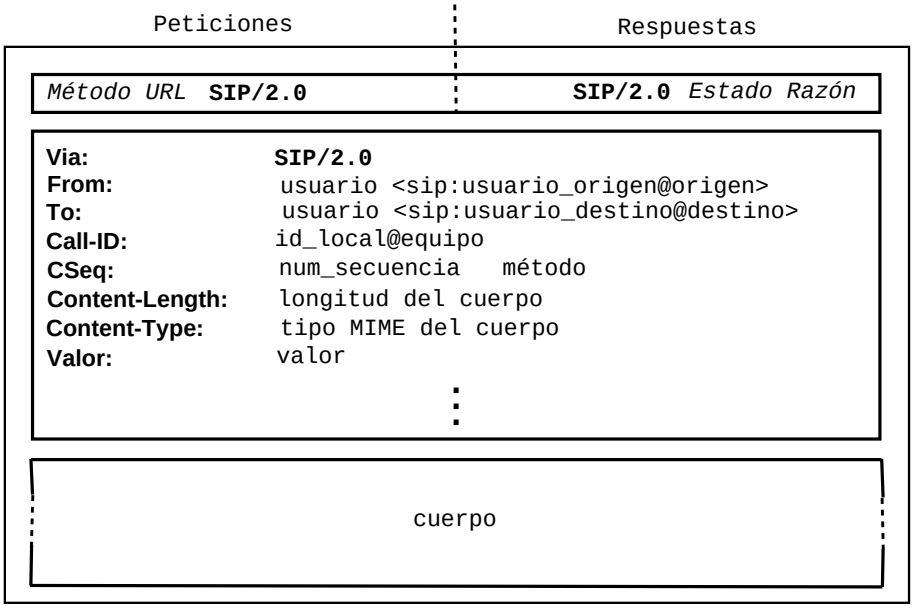


Figura 3.7: Sintaxis general de las peticiones y respuestas SIP

3.4.3. Sintaxis general

La estructura de las peticiones y respuestas SIP es similar a la de las peticiones y respuestas HTTP. Cada mensaje SIP contiene una cabecera y un cuerpo, separados por una línea en blanco, según se representa en la figura 3.7 . El contenido del cuerpo es libre y, por definición, queda fuera del ámbito del protocolo SIP.

Para todos los nombres de campos y parte del resto de los elementos, como el campo *media type*, la sintaxis no distingue mayúsculas y minúsculas. Se puede incluir espacios libremente, excepto en la primera línea. Los campos de cabecera con múltiples valores se pueden especificar en una sóla línea de valores separados por comas.

La primera línea de la cabecera indica el tipo de petición o respuesta. Los distintos tipos definidos hasta la fecha se relacionan en la tabla 3.2. Las cinco cabeceras *Via*, *From*, *To*, *Call-ID*, y *CSeq* están presentes en todos los mensajes e identifican de forma unívoca cada mensajes dentro de cada llamada.

Debido al diseño del protocolo SIP, el contenido de su cabecera es fácil de entender. El campo *From* proporciona la dirección del remitente inicial del mensaje, mientras que el campo *To* contiene la dirección del destinatario final. Las cabeceras SIP pueden contener un campo *Request-URI*, que, en tal caso, indica la dirección del destinatario actual del mensaje. Todos estos campos contienen un URL SIP.

El campo *Call-ID* es un identificador único para cada invitación o para todos los mensajes de registro procedentes de un cliente. En los mensajes de tipo OPTIONS, el campo *Call-ID* permite relacionar las peticiones y respuestas.

CSeq debe contener un número entero, expresado como número decimal, que indica el número de orden del mensaje en la llamada SIP. Cuando se retransmiten mensajes, este campo tiene el mismo valor para todas las retransmisiones; en otro caso, su valor se incrementa para cada nuevo mensaje de cada tipo.

Es destacable como nota general acerca del tratamiento de los mensajes SIP que los mensajes

de petición contienen la lista de equipos que componen la ruta seguida en el campo *Via*, del mismo modo que las respuestas contienen la ruta de retorno[5]. Por ello, a diferencia del estándar H.323, los servidores SIP no tienen por qué mantener el estado durante las llamadas. con este procedimiento, es además posible que diferentes pares peticiones/respuestas de una misma llamada puedan tomar rutas distintas, de modo que se puede recurrir a servidores de respaldo en caso de caida de servidores principales.

3.4.4. Métodos básicos

Método INVITE. Los campos de cabecera de las peticiones de inicio de llamada contienen, entre otros datos, la dirección de origen y destino de la llamada, el asunto, la prioridad, los equipos intermedios por los que se solicita que pase la llamada y las preferencias acerca de la localización del destinatario. El cuerpo contiene una descripción de la información multimedia que se pretende intercambiar en la sesión. Este componente, transparente para el protocolo SIP, se suele codificar siguiendo el formato SDP (véase la sección 3.6.1). Las respuestas a peticiones INVITE contienen en el cuerpo la descripción de la información multimedia que está dispuesto a intercambiar el destinatario de la petición inicial.

El formato de descripción de sesiones utilizado en el cuerpo de las peticiones INVITE puede ser cualquier formato que tenga asignado un tipo MIME, como los descriptores H.245, SMIL [8], o descripciones en algún otro formato XML, como SDPng (véase la sección 3.6.2. Nótese que los mensajes INVITE pueden tener como destinatarios agentes de usuario servidores asociados a un usuario real, o sistemas automáticos, como servidores de contenidos multimedia.

Asimismo, es posible modificar las características de una sesión multimedia posteriormente a su establecimiento. Las nuevas características se han de especificar en el cuerpo de un mensaje INVITE enviado con el mismo valor *Call-ID*, y nuevos valores para el cuerpo y posiblemente alguno de los campos de cabecera. Esta capacidad permite, por ejemplo, conmutar de modos de transmisión unicast a multicast o viceversa cuando varía el número de participantes en una sesión.

La figura 3.8 presenta un posible contenido de la invitación de la figura 3.4(a), de la página 36. Por brevedad, no se detalla el cuerpo del mensaje, que en este caso es una descripción SDP, según se deduce del valor de la cabecera *Content-Type*, *application/sdp*, que es el tipo MIME asignado a las descripciones SDP.

En [77] y [78] se puede encontrar un gran número de ejemplos de llamadas SIP genéricas y aplicadas a la comunicación con la red telefónica conmutada, respectivamente.

Método ACK. Los mensajes de este tipo sirven de confirmación para intercambios fiables de mensajes de invitación. Los clientes deben generar mensajes ACK para confirmar que se ha recibido el mensaje final de aceptación correspondiente a una invitación, véase la figura 3.4(a), de la página 36. El cuerpo de los mensajes ACK puede contener la descripción de la sesión si el cliente decide realizar modificaciones.

Método BYE. Estos mensajes indican a los servidores que un cliente desea finalizar la conexión entre dos participantes en una sesión. Se pueden generar tanto en los agentes que iniciaron la

[5] Al construir una respuesta, los servidores simplemente copian el campo *Via* de la petición, invirtiendo el orden de la lista.

```
INVITE sip:kirk@flota.int SIP/2.0
Via: SIP/2.0/UDP NCC-1701.flota.int
From: James Tiberius Kirk <sip:jtkirk@flota.int>
To: Sr. Scotty <sip:scotty@flota.int>
Call-ID: 55555555@NCC-1701.flota.int
CSeq: 1 INVITE
Subject: ¿Cuánto tiempo tardará?
Content-Type: application/sdp
Content-Length: ----
```

Figura 3.8: Ejemplo de cabecera de petición INVITE

llamada como en los que recibieron la invitación.

Método CANCEL. Empleado para cancelar una llamada pendiente, aunque su recepción por parte de un UAS no garantiza que éste no responda posteriormente, sino que simplemente constituye una sugerencia realizada por el remitente para optimizar el uso de la red. Estos mensajes deben contener el mismo valor para los campos *Call-ID*, *To*, *From*, y *CSeq* que el mensaje de invitación original. En todo caso, estos mensajes nunca finalizan una llamada ya establecida.

Método OPTIONS. Útil para solicitar información acerca de las posibilidades de un servidor. Los servidores de redirección y los proxys simplemente los reenvían. Otros servidores pueden responder con un mensaje en el que indiquen sus capacidades o bien con la respuesta que hubiesen dado a una invitación.

Método REGISTER. Los mensajes REGISTER proporcionan la localización de un agente de usuario a los servidores de registro. En ellos, los agentes de usuario clientes notifican a los proxys o los servidores de redirección la dirección o direcciones en las cuales se encuentra un usuario.

En estos mensajes el campo *To* de la cabecera indica la dirección que se ha de registrar, mientras que el campo *From* indica la dirección del usuario responsable del registro. Para el registro de dispositivos durante su arranque se ha reservado una dirección multicast, *sip.mcast.net*, a la que pueden enviar mensajes REGISTER.

Las respuestas a los mensajes de tipo REGISTER contienen información de configuración para los agentes de usuario, ampliable mediante las funciones de negociación de capacidades.

3.4.5. Métodos de ampliación

SIP prevé mecanismos para añadir nuevos métodos, así como para que los servidores que desconozcan estos nuevos métodos puedan responder con un código de error (código 501, véase la tabla 3.3), indicando la lista completa de métodos admitidos en un campo, *Allow*, de la cabecera de la respuesta.

Método INFO. El método INFO se ha definido en [40] para incorporar la posibilidad de enviar mensajes de control durante una sesión. Las peticiones de este tipo no cambian el estado de las llamadas ni los parámetros de la sesión, sino que se emplean para intercambiar cualquier tipo de información de control durante una llamada. Se trata de un método de carácter general,

cuyas cabecera y cuerpo se pueden emplear para enviar información de cualquier tipo, como dígitos DTMF [146], nivel de señal en enlaces inalámbricos o imágenes. Este método se ha definido como un mecanismo abierto a la definición posterior de otros tipos de señalizaciones, incluyendo guías generales para ello.

Este método juega un papel fundamental en la adaptación de la señalización SS7 de la red telefónica a redes basadas en SIP [183], permitiendo emular los mensajes que en la red telefónica se transmiten durante las llamadas.

Métodos SUBSCRIBE, UNSUBSCRIBE Y NOTIFY. El protocolo PINT [117] define ampliaciones tanto para SIP como para SDP. En particular define estos tres nuevos métodos SIP, mediante los cuales es posible solicitar, desde redes IP, la prestación de servicios telefónicos por parte de dispositivos situados en redes tradicionales, es decir, dotar a dispositivos SIP de capacidades de presencia.

Los clientes SIP pueden enviar mensajes SUBSCRIBE a servidores PINT para suscribirse a sesiones de servicios telefónicos. La relación entre los métodos SUBSCRIBE y UNSUBSCRIBE es análoga a la existente entre los métodos INVITE y BYE. Esto es, tanto los agentes de usuario como los servidores PINT pueden enviar mensajes UNSUBSCRIBE con objeto de que cese la participación de un cliente en una sesión de servicios.

Los mensajes NOTIFY se envían durante una sesión de servicios telefónicos para comunicar a un suscriptor que se ha producido un cambio en el estado de la sesión; como puede ser, la finalización del envío de un fax.

Método PRACK. Los agentes de usuario clientes envían peticiones de este tipo a los servidores para confirmar la recepción de respuestas provisionales. Como se comentará en la sección 3.4.7, se ha introducido para aumentar la fiabilidad en los intercambios de peticiones y respuestas.

3.4.6. Respuestas a peticiones

La tabla 3.3 resume las respuestas que pueden generar los servidores SIP ante cualquier petición. Las respuestas pueden ser tanto definitivas como provisionales. Los proxys y servidores de registro, así como los agentes de usuario servidores pueden generar todos los tipos de mensajes. Sin embargo, los servidores de redirección no generan códigos 2xx (éxito) ni 6xx (errores globales).

3.4.7. Mecanismos de fiabilidad

Normalmente, SIP se realiza basando su operación en un protocolo de transporte no fiable, como UDP. SIP define mecanismos de comunicación fiable basados en retransmisión [130]. Como enfoque general, el comportamiento de los servidores SIP se define de modo que descarten las peticiones repetidas (identificables gracias al campo *CSeq*), devolviendo además un mensaje de notificación.

Para los métodos BYE, CANCEL, OPTIONS Y REGISTER, se definen reglas de retransmisión a intervalos crecientes de forma exponencial a partir de un intervalo inicial con un valor recomendado de, al menos, medio segundo. Sin embargo, este procedimiento no es aplicable al método INVITE, puesto que la respuesta ante este tipo de métodos pueda tardar varios intervalos de retransmisión, debido a diversos factores, como pueden ser búsquedas complejas o retardos desde que el agente servidor

Respuesta provisional (1xx)	100	continuar
	180	llamando
	181	la llamada se está retransmitiendo
	182	la llamada está en cola
Éxito (2xx)	200	confirmación
	201	aceptación
Redirección (3xx)	300	varias posibilidades
	301	traslado permanente
	302	traslado provisional
Errores en el cliente (4xx)	400	petición incorrecta
	401	no autorizado
	402	ampliación incorrecta
	403	prohibido
	404	no encontrado
	480	no disponible temporalmente
	482	bucle detectado
	484	dirección incompleta
Errores en el servidor (5xx)	500	error interno del servidor
	501	no realizado
	502	pasarela no válida
	503	servicio no disponible
	505	versión no admitida
Errores globales	600	ocupado
	604	no existe
	606	inaceptable

Cuadro 3.3: Códigos de respuesta comunes del protocolo SIP.

de usuario destinatario notifica la llamada a su usuario hasta que éste esté disponible y proporcione una respuesta.

Sin embargo, los mecanismos de retransmisión comentados sólo se utilizan para las peticiones y las respuestas definitivas, no para las respuestas provisionales (véase la tabla 3.3). Este hecho ocasiona problemas de compatibilidad con la red telefónica. Además, las transacciones de invitación pueden ser bastante prolongadas, llegando incluso a requerirse el envío de solicitudes a servidores intermediarios para que la transacción no se cancele por inactividad. Por estos motivos, en [130] se ha definido el método de ampliación PRACK.

Recientemente se ha desarrollado el protocolo SCTP, descrito en la sección 2.1, que supone una alternativa como protocolo de transporte para SIP. SCTP proporciona, junto con otras funciones, mecanismos de retransmisión más elaborados que los disponibles con UDP. Estudios recientes [132] ponen de manifiesto que la fiabilidad conseguida con SCTP es mayor que con UDP, al tiempo que las prestaciones son mayores que con TCP.

3.5. SAP

La especificación del protocolo SAP [59] establece las reglas que los directorios de sesiones multicast deben seguir en la transmisión periódica de anuncios de sesiones, tales como el límite de ancho de banda, el algoritmo a seguir para el cálculo del intervalo de transmisión, las direcciones y el puerto de destino de los anuncios, las condiciones bajo las cuales se considera que las sesiones anunciadas dejan de estar activas, etc. Puesto que SAP es un protocolo diseñado para el anuncio de sesiones predefinidas, a diferencia de SIP, no contempla ningún proceso de negociación.

Los anuncios SAP, cuya estructura se esquematiza en la figura 3.9, contienen la descripción de sesiones junto con una cabecera de *autenticación*, que debe estar presente salvo en circunstancias excepcionales. El campo indicador del tipo de datos de la descripción puede contener el tipo MIME de la descripción transportada en el anuncio. Si la descripción tiene formato SDP, el uso de este campo es opcional.

La cabecera de autenticación tiene dos funciones: autenticar al creador de la sesión descrita en un anuncio y verificar la misma identidad cuando se solicite la modificación o suspensión de una sesión. SAP versión 2 contempla autenticación PGP [70] y CMS [68].

Aunque admite el uso de formatos distintos a SDP para describir sesiones, se desaconseja en aras de la mayor interoperabilidad posible, mientras que es obligatorio que todo agente SAP manipule correctamente *descripciones en formato SDP*

El *alcance* de los anuncios se restringe al ámbito de la sesión, para lo cual el receptor SAP debe determinar el ámbito multicast en el que se encuentra, probablemente mediante los mecanismos proporcionados por el protocolo MZAP [61].

El *cifrado* de los anuncios SAP, aunque se prevé –indicada por un bit (E) en la cabecera SAP–, no se recomienda, dado que los escenarios en los que se puede necesitar probablemente requieren otros mecanismos de anuncio. De hecho, en [59] no se especifica ningún algoritmo concreto de cifrado, ni mecanismos de generación y distribución de claves. de este modo se evita el desperdicio de ancho de banda que puede suponer transmitir anuncios cifrados cuando algunos receptores no son capaces de descifrarlos. Asimismo, se permite *comprimir* el contenido de los anuncios usando el formato Zlib, descrito en [38], en cuyo caso el bit C de la cabecera debe tener el valor 1.

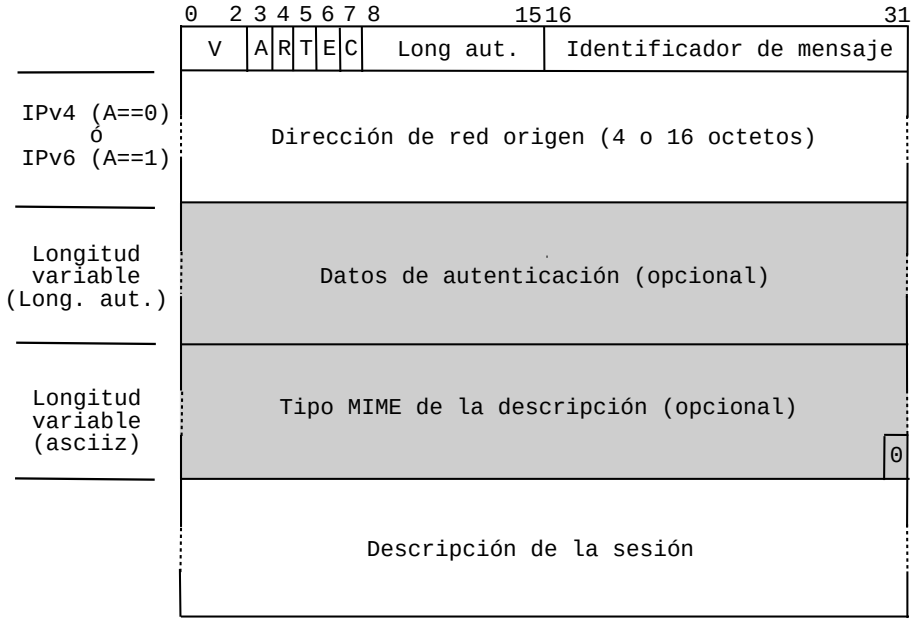


Figura 3.9: Estructura de los anuncios SAP

El bit R de la cabecera es un campo reservado cuyo valor debe ser 0, mientras que el bit T indica si el anuncio corresponde a la creación de una sesión (valor 1), o se trata de una solicitud de eliminación de un anuncio previo (valor 0).

Dado que SAP es un protocolo con muy bajo consumo de ancho de banda –por omisión se asume cuatro kilobits por segundo en cada grupo de anuncios–, el periodo de transmisión de anuncios proporcionado por el algoritmo dado en [59] puede llevar a que cada sesión se anuncie a intervalos del orden de decenas de minutos. Por ello, como mecanismo para que sea posible determinar rápidamente la existencia de sesiones, se recomienda el despliegue de *proxys cache* que mantengan una lista actualizada de toas las sesiones anunciadas.

Los orígenes del protocolo SAP se encuentran en una de las aplicaciones de Mbone, sd [180] –Session Directory–, un directorio de sesiones para anunciar conferencia multimedia. Las primeras versiones experimentales de SAP, SAPv0 y SAPv1, se emplearon en la aplicación sdr [180], similar a sd excepto en interfaz de usuario. Las modificaciones añadidas en la especificación vigente, SAPv2 [59], permiten que las aplicaciones SAPv2 puedan interpretar anuncios especificados en los formatos anteriores, pero, por el contrario, no todos los anuncios SAPv2 se pueden interpretar correctamente en herramientas implementadas para SAPv1. Las herramientas desarrolladas para SAPv0 no pueden interpretar anuncios SAPv2.

3.6. Descripción de Sesiones y Negociación de Capacidades

3.6.1. Descripciones SDP

El protocolo SDP, especificado en [57, 58], es un lenguaje de descripción diseñado exclusivamente para describir sesiones multimedia en tiempo real, siendo útil para invitaciones, anuncios, y cualquier otra forma de inicio de sesiones. En la actualidad, es el protocolo de descripción más extendido para

anunciar así como para negociar las características de las sesiones multimedia en Internet.

En este ámbito, por descripción se entiende proporcionar en formato estándar la información necesaria para que posibles participantes (usuarios y aplicaciones) puedan tener conocimiento de y unirse a una sesión. Puesto que SDP no es más que un formato de descripción, las descripciones SDP se pueden transportar mediante distintos protocolos, como SAP [59], SIP [60], RTSP [147], correo electrónico con ampliaciones MIME [48, 49] o HTTP [45].

Cabe destacar que, aunque SDP ha sido desarrollado por el IETF, durante los últimos años muchas recomendaciones de la ITU-T relacionadas con aplicaciones multimedia y redes IP (las recomendaciones T.38, Q.1970, H.332 y H.248.15, entre otras) lo están adoptando como protocolo de descripción de sesiones. Asimismo, el lenguaje SMIL [8] del W3C utiliza descripciones SDP [67].

Nótese la variedad de situaciones en las que se puede emplear una descripción SDP. Por ejemplo, en el caso de descripciones transmitidas en anuncios SAP los usuarios que reciben los anuncios probablemente se encontrarán en el ámbito (zona de la red) en el cual se desarrolla la sesión y por tanto podrán participar, mientras que en otros casos (descripciones distribuidas mediante HTTP) usuarios que están fuera del ámbito de la sesión pueden acceder a su descripción.

Sin embargo, en la actualidad SDP se emplea además, aunque esta otra función no se distinga de forma explícita [90], para describir las capacidades (de recepción, captura, reproducción, etc) de los sistemas y proporcionar varias alternativas de configuración a elegir. Esto es, SDP se emplea actualmente para desempeñar dos funciones: descripción de sesiones y negociación de capacidades, tarea esta última para la cual no está diseñado.

El diseño original de SDP ha estado motivado e influenciado por su uso en directorios de sesiones en Mbone, dedicados a anunciar conferencias multimedia y comunicar las direcciones y herramientas específicas necesarias para participar en ellas. Sin embargo, SDP es válido como formato de descripción de propósito general. Si bien su definición no contempla mecanismos de negociación de contenidos y/o codificación, las especificaciones de los protocolos SIP, RTSP, SAP y MEGACO designan a SDP como formato de descripción preferente para sus funciones de especificación y negociación de contenidos y formatos multimedia.

Cada descripción SDP proporciona la siguiente información:

- Nombre y propósito de la sesión
- Tipo y formato de los medios que la componen
- Intervalo(s) temporal(es) de desarrollo de la sesión
- Parámetros necesarios para poder recibir e interpretar los datos: direcciones, puertos, formatos, etc.
- Información complementaria relativa a los recursos de red necesarios para participar en la sesión: ancho de banda e información de contacto del responsable de la sesión

Es posible establecer sesiones de *ámbito privado* cifrando la descripción SDP, proceso que depende del mecanismo utilizado para transportar la descripción. También es posible anunciar una sesión en privado, incluyendo entonces las claves y la información acerca del esquema de cifrado en el propio anuncio.

```
1  v=0
2  o=foob 13354567 23344578 IN IP4 128.111.52.10
3  s=Nombre de esta sesión
4  i=Información acerca de esta sesión
5  u=http://www.example.org
6  e=Foo Bar <foo.bar@example.org>
7  p=Foo Bar (34)111-11-11-11
8  t=3113654400 3193000000
9  a=tool:sdr v2.5a8
10 a=type:broadcast
11 m=audio 19660 RTP/AVP 0
12 c=IN IP4 224.2.1.1/127
13 a=ptime:40
14 m=video 61000 RTP/AVP 31
15 c=IN IP4 224.2.2.1/127
```

Figura 3.10: Ejemplo de descripción SDP

A diferencia de otros formatos de representación como ASN.1 [179] o XDR [162], y al igual que en el caso del protocolo SIP, con objeto de aumentar la portabilidad y hacer posible su procesado con *scripts*, la *codificación* de las descripciones SDP es textual. Dado que las descripciones SDP pueden estar contenidas en unidades de tamaño limitado, como anuncios SAP o SIP sobre UDP, la codificación de los campos es muy compacta. Además, las reglas de composición de los campos de las descripciones son muy poco flexibles, de modo que su procesamiento y validación sean sencillos.

SDP da soporte para *mecanismos de clasificación de sesiones automatizables*. Asimismo, en su especificación se recomienda el uso del conjunto de caracteres ISO 10646 en codificación UTF-8 para los campos de contenido libre, aunque para permitir representaciones compactas se admiten otros conjuntos, como el ISO 8859-1. Los nombres de los campos y atributos se codifican en el subconjunto US-ASCII del UTF-8.

Cada descripción SDP está constituida por campos o líneas de texto con la siguiente estructura:

```
<tipo> = <valor>
```

Siendo `<tipo>` un carácter para el cual se distinguen mayúsculas y minúsculas, mientras que el formato de `<valor>` depende de `<tipo>`. Las líneas se agrupan en una sección inicial –que describe globalmente la sesión– y cero o más secciones específicas para cada uno de los flujos de datos que intervienen en la sesión.

La figura 3.10 ilustra la estructura de las descripciones SDP con un ejemplo. Aunque el formato SDP no está diseñado para que los usuarios manipulen e interpreten directamente las descripciones, no es difícil hacerlo. La descripción de ejemplo corresponde a una sesión en la que el protocolo de transporte es RTP, con perfil para sonido y vídeo, sonido en formato PCM de ley μ (tipo de datos 0 en RTP) al puerto 19660, y vídeo en formato H.261 (tipo de datos 31) al puerto 61000. La línea 13 de la figura especifica, mediante el atributo `ptime`, que cada paquete RTP de sonido contiene muestras tomadas durante un intervalo de 40 milisegundos.

La especificación de SDP describe detalladamente la sintaxis y proporciona una gramática en

formato ABNF [33] , aunque, dada su sencillez se pueden construir analizadores basados simplemente en expresiones regulares.

Además de los atributos (especificados en las líneas que comienzan con la letra a) definidos en la especificación inicial de SDP, algunas aplicaciones pueden requerir otros específicos; por ello, se prevé el registro de los de uso común a través de la IANA [73], como es el caso de [89], que registra más de cuarenta atributos específicos para conexiones ATM. Este mismo procedimiento está contemplado para registrar valores frecuentes de algunos campos estándar.

La especificación original del protocolo SDP, versión 0, se definió en el RFC 2327 [57], vigente en la actualidad. El grupo de trabajo MMUSIC [74] del IETF ha venido desarrollando en los últimos meses un borrador [58] en el que se revisa ligeramente el protocolo. La gramática no sufrirá cambios, excepto una aclaración en la sintaxis de las direcciones multicast y la adición de la sintaxis concreta de las direcciones IPv6 [111]. Otros cambios menores son la actualización de la estructura de los URI, teniendo en cuenta el RFC 2732 [63], y ejemplos referentes al uso de SDP en SIP y RTSP.

La modificación prevista que afectará de forma más directa a las aplicaciones multimedia es el cambio de unidades en el valor de ancho de banda, especificado en las líneas que comienzan con la letra b, que pasa a ser de bits por segundo, en lugar de kilobits por segundo, como se especificaba en [57]. Este cambio es imprescindible para poder utilizar correctamente las modificaciones hechas en las últimas revisiones de RTP [144] y [26], que definen un mecanismo para especificar por separado el ancho de banda reservado al conjunto de transmisores y al conjunto de receptores en las sesiones RTP. Ambos valores suelen ser de unos pocos kilobits por segundo o incluso inferiores al kilobit por segundo.

Como ya se ha comentado, la especificación original de SDP no contempla la función de negociación de capacidades, siendo por tanto poco adecuada para conferencias que se puedan iniciar de forma espontánea, como una llamada telefónica. La capacidad de negociar y seleccionar las características de cada sesión multimedia es una función básica de cualquier arquitectura de control, como se destacó en la sección 3.1, por lo que se requiere una solución específica para este problema. Para cubrir esta carencia, al menos provisionalmente, se han definido ampliaciones a SDP, entre las que destacan [3], que describe un mecanismo sencillo de especificación de capacidades, y [23], que amplía SDP con atributos que permiten especificar grupos de medios. Sin embargo, se trata de soluciones simples y limitadas, puesto que debe mantener la compatibilidad con la especificación básica del protocolo. En la siguiente sección se describe la solución propuesta por el IETF.

3.6.2. Descripciones SDPng

SDPng [113, 90], el *protocolo de descripción de sesiones y negociación de capacidades* que se está desarrollando actualmente en el IETF como futuro sustituto de SDP, establece un modelo de descripción de sesiones multimedia con el que se pretende superar las limitaciones de SDP.

Siguiendo el modelo SDPng, una sesión multimedia consta de uno o más *componentes* de sesión, cada uno de los cuales describe un tipo de interacción (conversación, diapositivas, etc.) que se puede llevar a cabo mediante diferentes aplicaciones y posiblemente con diferentes protocolos. Asimismo, se define el concepto de *configuración* como un conjunto de valores concretos para los parámetros de un componente que permite llevar a cabo una variación de un componente. SDPng distingue *configuraciones potenciales* (posibles) y *configuraciones reales* (configuraciones posibles que se ha decidido utilizar para algún componente).

Esta última distinción es la base del proceso de negociación de capacidades, puesto que permite tanto especificar las posibles configuraciones como proponer la configuración deseada por los participantes en sesiones multimedia. Las configuraciones reales de una sesión se seleccionan mediante un proceso de negociación en el que se determina las configuraciones potenciales comunes a los participantes.

Desde el punto de vista sintáctico SDPng establece una base simple para el formato de las descripciones, basada en XML [20, 32], al tiempo que es ampliable de forma modular. SDPng no es compatible con SDP, dado que en el desarrollo de la nueva sintaxis se ha optado por una mayor expresividad. Sin embargo, desde el punto de vista semántico, son ampliamente compatibles, e incluso se han previsto mecanismos de conversión de descripciones SDP a SDPng.

SDPng contempla desde su especificación inicial todas las opciones que se han venido añadiendo a SDP, en forma de ampliaciones al estándar, durante los últimos años, por lo que presenta la estabilidad derivada de los años de experimentación con SDP. Además, características como calidad de servicio o seguridad, para las cuales no se ha propuesto todavía ninguna ampliación para SDP, probablemente sólo se incorporen a SDPng.

Las descripciones SDPng constan de cuatro secciones:

- Definiciones
- Configuraciones, comprendiendo configuraciones potenciales y reales
- Restricciones
- Atributos de sesión

Las secciones de configuraciones y de atributos de sesión son asimilables a las líneas de SDP. La sección de configuraciones de SDPng es parcialmente equivalente a las líneas de SDP que describen los medios y las conexiones de una sesión así como a sus atributos asociados, mientras que la sección de atributos de sesión de SDPng se corresponde con las líneas de parámetros de sesión de SDP. Sin embargo, las otras dos secciones no tienen equivalencia en SDP y permiten especificar descripciones con mayor expresividad.

SDPng admite la definición de nuevas familias de direcciones y protocolos de transporte, formatos multimedia y tipos de contenido de forma similar a SDP, pero de manera más estructurada que simplemente mediante nuevos tipos MIME. En [113] se perfila la utilización de SDPng en los protocolos de control de sesiones multimedia desarrollados por el IETF: SAP, RTSP, SIP y MEGACO.

3.6.3. Tipos MIME

Los tipos MIME [48, 49, 109, 50] se utilizan en los sistemas de conferencia multimedia de dos formas:

- *Indicación del formato de las descripciones de sesiones*. Las descripciones SDP pueden distribuirse mediante correo electrónico o transacciones HTTP; el tipo MIME asignado a estas descripciones es *application/sdp*, al que conviene asociar aplicaciones que interpreten la descripción y permitan lanzar, con los parámetros adecuados, las aplicaciones necesarias para participar en la sesión. Además, tanto los mensajes de señalización del protocolo SIP como los

del protocolo MEGACO, así como los anuncios SAP especifican el formato de descripción de sesiones utilizados mediante tipos MIME.

- *Especificación de formatos multimedia en descripciones de sesiones*. Tanto las descripciones SDP como las SDPng utilizan tipos MIME para identificar los formatos empleados en una sesión multimedia. Nótese que, en el caso de las descripciones SDP, los tipos se pueden especificar de dos formas. Cuando se trata de un formato RTP de sonido o vídeo, se emplea una sintaxis como la de las líneas 11 y 14 de la figura 3.10, es decir, se usa el identificador numérico del formato en el protocolo RTP (véase la sección 2.3.1). En otro caso, la sintaxis es como la de la línea siguiente:

  ```
  m=application 32416 udp wb
  ```

  donde se especifica por separado el tipo MIME superior (*application*) y el subtipo (*wb*).

  Actualmente está en desarrollo la especificación [28] del procedimiento a seguir para registrar formatos RTP como tipos MIME, documento en el que además se registran los formatos definidos en el perfil para transmisión de sonido y vídeo sobre RTP [141, 144]. Ha de tenerse en cuenta que, para un formato de datos, se puede asignar el mismo tipo MIME al formato de transporte en RTP así como a otros modos de transferencia, en cuyo caso es probable que existan parámetros opcionales distintos para cada modo. Sin embargo, en otros casos, como el formato de sonido Ogg Vorbis [194], se prefiere definir un tipo MIME distinto para el formato de transporte sobre RTP.

Capítulo 4

Diseño del sistema

A lo largo de las siguientes secciones se analiza el diseño del sistema a cuyo desarrollo se ha contribuido durante este proyecto. En primer lugar, se expondrán dos principios básicos de diseño que afectan directamente a los sistemas basados en RTP.

Las características específicas de los sistemas en tiempo real, presentadas en la sección 2.2, imponen restricciones que los protocolos de transporte desarrollados para otros tipos de sistemas no tienen en cuenta. Para afrontar este problema, RTP se ha definido como un protocolo de nivel de aplicación, siguiendo además los dos principios básicos de diseño propuestos en [30]: segmentación de unidades de datos en el nivel de aplicación e integración del procesamiento de los diferentes niveles. El primero es un principio de diseño de la arquitectura de los protocolos, mientras que el segundo es un principio de diseño de programación de los protocolos.

Estos principios pretenden dar una base sólida para protocolos que, en comparación con la redes TCP/IP actuales, constituyan la base de redes de mayor capacidad y adaptables a un rango más amplio de tecnologías y de servicios (voz, vídeo, gráficos y texto, en modos de transferencia posiblemente interactivos).

Como analizaremos, estos principios entran en contradicción con el tradicional diseño basado en capas, que no debería considerarse como el único enfoque válido, sino como un enfoque posible que hay que comparar con otros en función de su simplicidad y prestaciones. El nuevo modelo que se deriva de ambos principios, propuesto inicialmente en [30], se ha aplicado no sólo para el diseño del protocolo RTP, sino también en otros casos [19].

4.1. Segmentación de Nivel de Aplicación

En la sección 2.2 se justificó que en las aplicaciones de tiempo real, la función de reordenación de paquetes durante la recepción puede entrar en conflicto con la temporización de la reproducción de los datos. Existen dos alternativas en el comportamiento de las aplicaciones cuando las unidades de datos se reciben desordenadas:

- El protocolo de transporte detiene la entrega de unidades a la aplicación hasta que sea posible la entrega en orden –lo que muchas veces implica esperar a retransmisiones desde el transmisor.
- La aplicación continúa el procesado a pesar de que se hayan perdido o no hayan llegado aún

algunas unidades.

Con protocolos que gestionan de forma inflexible la reordenación de paquetes, como TCP, las aplicaciones en tiempo real quedan sustancialmente limitadas. En general, las aplicaciones deberían poder continuar el procesado de unidades de datos aunque algunas se hayan perdido, existiendo las siguientes alternativas:

- Continuar el procesado sin realizar comprobaciones adicionales, alternativa generalmente válida para vídeo y voz.
- Retransmisión por parte del transmisor, encargándose de esta función la aplicación (lo que permite tanto el uso de un *buffer* de retransmisión como el cálculo de los datos perdidos, según las condiciones.).

Además, el algoritmo aplicado para reordenar las unidades puede presentar diversas variaciones, en función de las necesidades de la aplicación. Sólo la aplicación puede conocer qué grado de desorden es admisible o qué secuencias de unidades son válidas según el tipo y formato de los datos manipulados.

El principio de diseño propuesto para solucionar el problema expuesto es el principio de segmentación de nivel de aplicación; según el cual, los niveles inferiores al de aplicación tratan los datos en unidades especificadas por la aplicación. Estas unidades se denominan *unidades de datos de aplicación (ADU)*, y se deben utilizar para todas las operaciones, tanto de manipulación de datos como de control de la transmisión. La ADU se define como la mínima unidad que la aplicación puede tratar independientemente de si se recibe en orden.

Siguiendo este principio, el transmisor debe incluir junto con cada unidad de datos de aplicación los parámetros necesarios para que el receptor pueda asignarle una posición o lugar en el flujo recibido aún cuando otras unidades se pierdan en el proceso de transmisión. Asimismo, la sintaxis de los datos debe permitir su procesado aún cuando se reciban desordenadas.

El protocolo de transporte RTP sigue este principio, utilizando matasellos para identificar el lugar de cada unidad de datos de aplicación dentro de un flujo de datos, y añadiendo en las guías de diseño de modos de empaquetado de formatos de datos la restricción de que cada unidad se pueda procesar aun cuando se reciba desordenada.

4.2. Procesamiento Integrado de Niveles

Generalmente [30], la función de mayor importancia de toda arquitectura de protocolos es la transferencia de datos entre aplicaciones. Dejando a un lado operaciones de control de alto nivel, como inicio de sesiones o localización de servicios, y centrándonos en las operaciones de transferencia de datos[1], las funciones realizadas por los protocolos se pueden clasificar en dos: *manipulación de datos* y *control de la transferencia*.

Las funciones de manipulación de datos más comunes en las aplicaciones de red son:

[1] Que normalmente se desarrollan de forma independiente respecto a las operaciones de transferencia de datos.

- Transformación entre formatos de presentación y de transferencia. Es decir, construcción de unidades de transferencia a partir de los datos utilizados por las aplicaciones, y viceversa.
- Cifrado de datos.
- Transferencia de datos entre la aplicación y el subsistema de E/S del sistema operativo.
- Transferencia de y hacia la red.
- Almacenamiento de datos transmitidos, de cara a su posible retransmisión.
- Detección y/o corrección de errores.

Nótese que la realización de todas la operaciones supone algún tipo de lectura o escritura de datos entre diferentes niveles o componentes del sistema, lo que, en algunos casos, requiere transferencias de zonas de memoria, al incorporar nuevos bloques de datos a las unidades.

Las funciones de control de la transferencia más comunes son:

- Multiplexión del servicio de transporte.
- Detección de problemas en la transmisión, como desorden o pérdidas de paquetes.
- Envío de acuses de recibo, función que, por su complejidad y amplio rango de variedades, merece ser mencionada aparte de otros métodos de detección de problemas en la transmisión.
- Control de congestión y de flujo.
- Asignación de matasellos que sirvan de base para protocolos de tiempo real.
- Indicación de los límites de las unidades de datos, de modo que los receptores puedan separar las unidades enviadas por el transmisor. Esta función es específica de aquellos protocolos basados en unidades discretas.

El orden en que es posible realizar las operaciones viene condicionado por las operaciones de control necesarias antes de que se puedan aplicar otras operaciones de manipulación.

De entre las funciones de control de la transferencia enumeradas, se puede distinguir un conjunto de funciones de control muy acopladas a otras operaciones de manipulación de datos, mientras que otras se pueden realizar con mayor grado de independencia. Compárese la función de asignación de matasellos, que inevitablemente va asociada a la transmisión de una unidad de datos, con la función de control de congestión, que, en general, se realiza de forma independiente.

Con la finalidad de aumentar la modularidad del sistema, uno de los objetivos de diseño debe ser reducir el número de operaciones de control directamente acopladas a operaciones de manipulación de datos.

Si analizamos las implementaciones de protocolos TCP/IP disponibles en núcleos de sistemas operativos de libre distribución, se puede comprobar [30, 115] que el coste computacional de las operaciones de control de la transferencia acopladas a las de manipulación es muy inferior a la de estas últimas[2]. La conclusión que se extrae de este análisis es que el aspecto donde se deben concentrar

[2]Normalmente, las operaciones de control se reducen a verificar números de secuencia o enviar un acuse de recibo, mientras que las operaciones de manipulación involucran la transferencia de bloques de datos de miles de octetos de longitud.

las mejoras de prestaciones necesarias para que se pueda mejorar la calidad de reproducción de las aplicaciones multimedia es la manipulación de datos.

Asimismo, la tendencia mostrada por las arquitecturas de ordenadores hacia sistemas RISC, superescalares y multiprocesador, hacen aun mayor la ventaja de que las lecturas/escrituras y las operaciones (copias, cifrado, cálculo de sumas, etc) realizadas sobre los bloques de datos se lleven a cabo en el mismo nivel, mientras la probabilidad de que los datos permanezcan en los niveles superiores de la jerarquía de memoria del sistema es mayor[3].

En este sentido, algunos experimentos realizados [30] muestran que la conversión entre formatos de presentación y de transporte puede tener efectos muy significativos[4], haciendo que las prestaciones del sistema queden divididas por 4. Del mismo modo, la realización de varias copias para cada bloque de datos [115] puede llegar a limitar sensiblemente el nivel de calidad de los formatos multimedia.

En [30] se introdujo como solución a este problema un principio de diseño de protocolos diferente al basado en una jerarquía de capas: *procesado integrado de niveles*. Este principio establece que las arquitecturas de los protocolos deben ser tales que al escribir aplicaciones basadas en ellas sea posible realizar todas las operaciones de procesado de datos (tanto manipulación como operaciones de control) en uno o dos bucles que integren todas las operaciones. Por tanto, la interacción de las operaciones de procesado no debe ser incompatible con su integración.

Una arquitectura que presente estas características *no elimina necesariamente la separación de funciones entre niveles*, sino que haciendo más flexible la interacción entre los niveles, amplia las posibilidades de implementación de aplicaciones. De hecho, este principio se ha venido siguiendo parcialmente desde hace más de una década en el núcleo BSD [30], con técnicas de optimización que posteriormente se han aplicado asimismo en los otros núcleos de libre distribución existentes en la actualidad.

El enfoque derivado de este principio es particularmente más eficiente y flexible que la jerarquización en capas en los niveles de transporte y superiores, puesto que los datos se intercambian entre los extremos de una conexión, de modo que no se pierde la posibilidad de que elementos intermedios de la red traten los datos de forma transparente. Como inconveniente, puede llevar –si no se sigue una metodología adecuada– a sistemas más complejos y difíciles de mantener, así como a un gran número de implementaciones dispares y particulares, lo que se debe paliar con principios de diseño genéricos.

Por ejemplo, desacoplando los detalles sintácticos de cada nivel, todos los niveles pueden utilizar los mismos datos, dotándoles de diferente semántica. De este modo, donde se tenía una jerarquía para aislar detalles sintácticos, este objetivo se logra sin la pérdida de flexibilidad y eficiencia impuesta por una jerarquía estricta.

Es más, como se discute en [83], la aplicación de una jerarquía estricta al software de redes dificulta la aplicación de principios de diseño orientado a objetos y orientado a aspectos, puesto que, entre otros problemas:

- Acopla las funciones de recepción y transmisión.
- Las aplicaciones no pueden controlar parámetros cuyo valor más adecuado se conoce con frecuencia sólo en el nivel de aplicación, como el periodo de retransmisión de paquetes o el tiempo máximo de espera a un acuse de recibo[5].

[3] Por ejemplo, realizando copias y cifrado en el mismo bucle en lugar de en bucles separados.
[4] En particular cuando involucran conversiones al formato ASN.1.
[5] Estos parámetros degradan significativamente las prestaciones de TCP en sistemas con un elevado intervalo de retorno

- Funciones como la gestión de calidad de servicio quedan dispersas entre diferentes niveles.
- Cada nivel queda acoplado a, utiliza y depende de los servicios del sistema (hilos, sincronización, etc) que lo ejecuta de forma independiente al resto de niveles.

Al contrario de lo que cabría esperar, cada capa se puede convertir en un componente de gran tamaño y difícil de reutilizar, mezclando fragmentos de código que podrían ser independientes. Esto lleva a aumentar la complejidad de las implementaciones de protocolos y dificultar su ampliación. A continuación se comentan los problemas citados:

- *Acoplamiento de funciones independientes*. Dentro de cada nivel se mezclan funciones que podrían realizarse de forma independiente, como las operaciones de transmisión y recepción. Basta considerar el caso de un transmisor que gestiona *buffers* de retransmisión y un receptor que responde con acuses de recibo. En un sistema como este, el transmisor y el receptor implementan operaciones (retransmisión frente a envío de acuses de recibo, respectivamente) que podrían separarse, siendo así componentes reutilizables y adecuados para sistemas que sólo envían o reciben.
- *Limitaciones en la comunicación entre niveles*. Algunos parámetros que se deberían poder establecer desde niveles superiores, como por ejemplo el intervalo de retransmisión de paquetes (que depende de las condiciones de la red y las necesidades de la aplicación) o el tratamiento que se debe dar a los paquetes desordenados o con errores, están aislados en niveles inferiores, sin que sea posible su modificación. Del mismo modo estos parámetros están acoplados con otras funciones, como puede ser el control de errores, que pueden no interesar a los niveles superiores.
- *Dispersión de aspectos dependientes del sistema*. En los sistemas que siguen una jerarquía estricta, aspectos como el número de hilos de ejecución que llevan a cabo las funciones del protocolo quedan dispersos entre los diferentes niveles, de modo que la modificación de estos aspectos mediante parámetros es más complicada.
- *Dispersión de funciones globales de las pilas de protocolos entre los niveles*. La introducción de servicios adicionales, como gestión de calidad de servicio, en una pila de protocolos jerarquizada puede requerir modificaciones en la interfaz y el código de todas las capas. Esto es, algunas funciones requieren una estructura vertical, en lugar de la estructura puramente horizontal impuesta por una jerarquía de capas.

4.3. Arquitectura Global del Sistema

El sistema en el cual se integra el trabajo de implementación realizado durante este proyecto se compone de varios paquetes de software libre: un conjunto de bibliotecas que implementan protocolos de transporte y control de sesiones multimedia de nivel de aplicación, así como una biblioteca base de aplicaciones de red.

Con este sistema se pretende proporcionar una base sólida y completa para la realización de aplicaciones de la arquitectura de sistemas multimedia del IETF. Dado el amplio espectro de aplicaciones

de acuses de recibo.

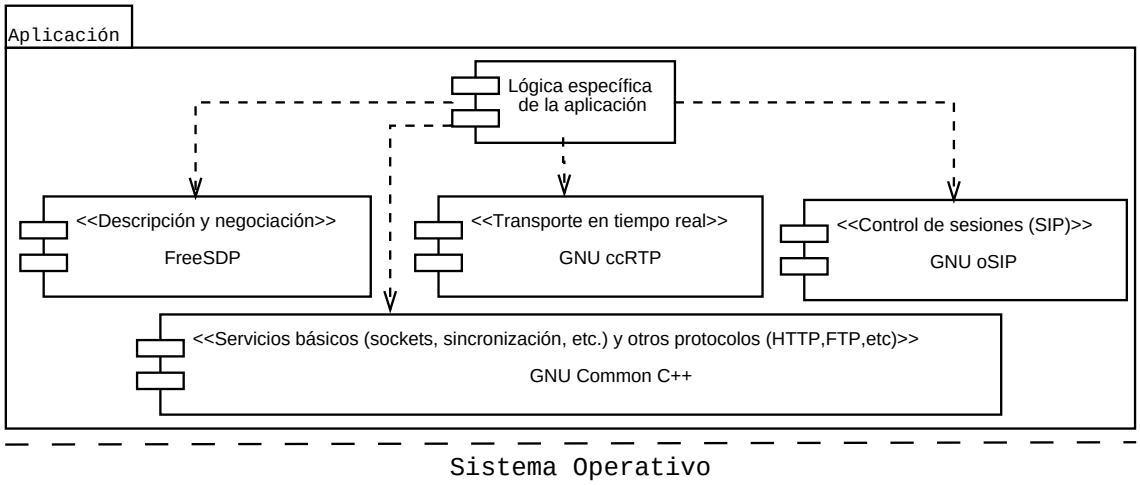


Figura 4.1: Arquitectura global del sistema

a las que se debe adaptar, desde aplicaciones básicas de voz sobre IP a aplicaciones de convergencia entre redes de conmutación de paquetes y redes de conmutación de circuitos, pasando por directorios de sesiones y sistemas de conferencia multicast, el sistema debe ser altamente *modular*. Por el mismo motivo, debe ser *portable*, sin dejar de tener en cuenta entornos y sistemas operativos alejados de los de propósito general. Asimismo, dado que la arquitectura del IETF es altamente escalable, el sistema debe ser *escalable*, tanto en cuanto al coste computacional de sus algoritmos como en cuanto al uso de recursos del sistema operativo. Por último, el diseño debe tener en cuenta que, como se detallará en el capítulo 5, durante la implementación del sistema se pondrá especial énfasis en su *robustez y seguridad*.

La primera decisión de diseño que se ha tomado ha sido la selección del lenguaje de programación del sistema. Se ha optado por usar C++ [173, 43], aunque existen otros lenguajes adecuados a los problemas que se pretende solucionar, como Eiffel o Ada. La decisión ha venido motivada por la mayor extensión y portabilidad de C++ así como por la posibilidad de integrar componentes programados en C, lo que permite incorporar al sistema componentes aptos para sistemas muy restringidos.

En la figura 4.1 se muestra un esquema de la arquitectura de las aplicaciones basadas en el sistema propuesto. El componente de menor nivel del sistema, GNU Common C++ [175], es una biblioteca de abstracción de servicios del sistema y otros servicios y protocolos útiles para aplicaciones de red. Como biblioteca base, proporciona estos servicios tanto para los demás componentes del sistema como para las aplicaciones. Los otros tres componentes, GNU ccRTP [174], GNU oSIP [107] y FreeSDP [108], proporcionan, respectivamente, servicios de transporte en tiempo real sobre RTP, manipulación y gestión de peticiones y transacciones SIP, y manipulación de descripciones SDP. Aunque la figura esquematiza una aplicación que hace uso de todos los componentes, cualquiera de ellos se puede utilizar de manera aislada en otros sistemas.

En particular, aunque GNU ccRTP se basa ampliamente en servicios de GNU Common C++, estas dependencias están concentradas de forma que todas las funciones independientes de servicios del sistema tales como sockets [168], hilos [169] o sincronización [1] se pueden reutilizar sin requerir el uso de GNU Common C++.

Asimismo, puesto que, hasta la fecha, las implementaciones de SDP y SIP no han planteado problemas de diseño del grado de complejidad que sí plantea el protocolo RTP, GNU ccRTP se ha

implementado en C++, mientras que FreeSDP y GNU oSIP están implementadas en C.

En cuanto a la escalabilidad del sistema, todos los componentes están preparados para diferentes modelos de ejecución basados en un número variable de hilos.

Las siguientes secciones describen el diseño de cada uno de los componentes. La descripción se acompaña de diagramas UML de interacción y de clases simplificados (sin mostrar los métodos ni los datos miembro de las clases y ocultando las clases de menor importancia y aquellas que encapsulan únicamente detalles de implementación). Se describirá brevemente la interfaz de programación de cada una de las bibliotecas, proporcionándose referencias a los respectivos manuales.

4.4. GNU Common C++

GNU Common C++ [175] es una biblioteca escrita en C++ que proporciona abstracciones para servicios del sistema, como sincronización, procesos, hilos, sockets, carga dinámica de objetos, temporización, acceso sincronizado a archivos o entrada/salida serie, así como servicios y protocolos orientados a aplicaciones de red, como los protocolos HTTP, FTP o SSL, tratamiento de documentos XML o manipulación de URLs. Common C++ incorpora además clases base para funciones genéricas, como gestión de archivos de configuración y mecanismos de persistencia[6].

Estos elementos se integran de forma que constituyen un modelo de desarrollo de aplicaciones, con un esquema de excepciones propio y un subsistema de plantillas que recoge algunos de los patrones de programación más comunes [101, 102, 1, 99], como listas, conjuntos, objetos sincronizados o contadores de referencia. En general, las clases de Common C++ están diseñadas de manera que pueden utilizarse directamente para las funciones más comunes, al tiempo que admiten ampliaciones y variaciones mediante herencia o especialización.

Common C++ está diseñada para usarse en aplicaciones con múltiples hilos de ejecución, proporcionando un modelo neutral y completo para crear, gestionar y destruir hilos de ejecución (mediante la clase `Thread`, cuya estructura básica es similar a la de la clase homónima de Java, disponiendo de un método `run` equivalente). Asimismo, todas las clases del sistema están diseñadas para funcionar correctamente en aplicaciones multihilo. Algunas de estas clases se han integrado en Common C++ específicamente para dar soporte a aplicaciones multihilo, entre ellas destacan las siguientes:

- `Semaphore`, abstracción de semáforos de sincronización.
- `ThreadLock`, que proporciona bloqueos de lectura/escritura.
- `Conditional`, abstracción de las variables de condición utilizadas para implementar monitores.
- `AtomicCounter`, que implementa un contador
- `Buffer` y `FixedBuffer`, clases base para gestionar bloques de memoria de acceso sincronizado.
- `TimerPort`, abstracción para la planificación temporal de sucesos.

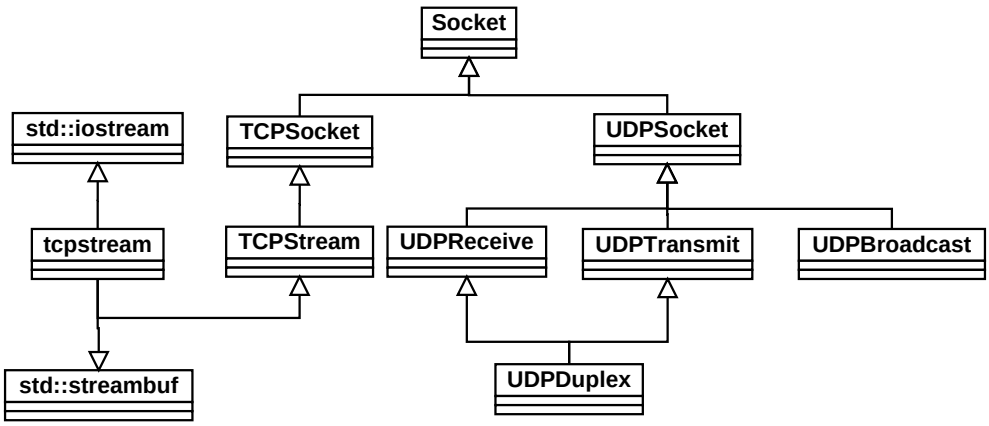


Figura 4.2: Diagrama de clases de la jerarquía de sockets de Common C++

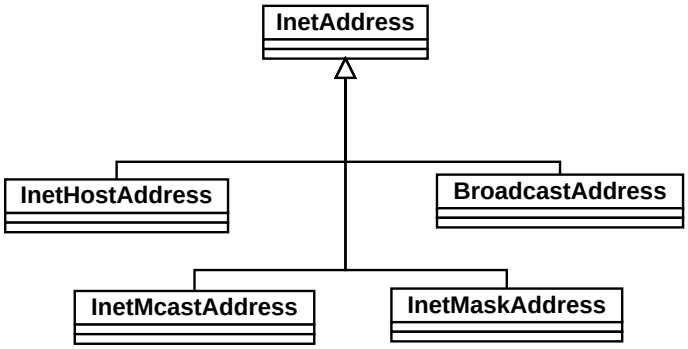


Figura 4.3: Diagrama de clases de la jerarquía de direcciones de Internet de Common C++

Gran parte de las abstracciones de Common C++ se integran en el modelo de flujos de entrada/salida de C++. En particular, en la figura 4.2, que muestra un esquema simplificado de la jerarquía de clases de sockets, se puede observar la clase `tcpstream`, que define un flujo de entrada/salida asociado a una conexión TCP. En la figura aparece asimismo la jerarquía de las clases relacionadas con los sockets UDP, que, siguiendo las líneas generales de diseño de Common C++, ofrece varios niveles de especialización.

Common C++ define, mediante las clases `SocketService` y `SocketPort`, un modelo para agrupar objetos de la jerarquía de sockets de Common C++, de manera que sus conexiones sean procesadas por un mismo hilo de ejecución. Este modelo incorpora la posibilidad de definir mediante herencia métodos que se deben ejecutar ante ciertos sucesos, como recepción de datos o errores en la transmisión, así como métodos de ejecución periódica.

Relacionada con la jerarquía de sockets, la jerarquía de direcciones de Internet de Common C++ (figura 4.3), además de aplicar polimorfismo y tipado estricto a las direcciones de Internet, emplea un mecanismo de validación genérico basado en una jerarquía paralela de objetos función [1, 99, 173].

De esta forma, Common C++ ofrece abstracciones y servicios útiles tanto en la implementación de protocolos como RTP o SIP, como en la implementación de la mayoría de las aplicaciones basadas en estos protocolos, que frecuentemente dependen de tales servicios en la misma medida que de-

[6] Al igual que para todos los componentes del sistema, una descripción más exhaustiva que la presentada en esta memoria se puede encontrar en el manual y el manual de referencia distribuidos junto con Common C++, también disponibles en formato HTML en su web

penden de los protocolos de transporte y control. Además, Common C++ constituye una sólida base para la realización de protocolos y aplicaciones escalables, proporcionando elementos para el uso y asignación flexible de hilos de ejecución, así como mecanismos de gestión de memoria adaptables a sistemas multiprocesador, aspecto que se detallará en el capítulo 5.

Junto con los servicios del sistema y de red básicos comentados, Common C++ también ofrece abstracciones relacionadas con algunos de los protocolos de nivel de aplicación más extendidos en Internet, como HTTP, FTP o SSL. Estas abstracciones definen flujos de entrada/salida de C++ (de manera similar a la clase `tcpstream`), siendo asimismo ampliables mediante herencia. De esta forma, se completa el modelo que Common C++ establece para el desarrollo de aplicaciones de red fácilmente integrables en entornos web.

4.5. GNU ccRTP

GNU ccRTP proporciona servicios de transporte sobre RTP así como un modelo de desarrollo de aplicaciones basadas en este protocolo. En las secciones anteriores se ha expuesto el enfoque y los principios de diseño seguidos en el desarrollo del protocolo RTP. Siguiendo estos principios, RTP define en el nivel de aplicación, además de un formato de paquete y operaciones de manipulación de éstos, operaciones de control de la transferencia, algunas acopladas a la manipulación de los paquetes de datos, y otras –fundamentalmente las funciones de RTCP– independientes.

Actualmente existe un número suficiente de implementaciones de RTP ([136, 180, 82, 112, 135, 185, 145], entre otras), tanto en forma de bibliotecas como en forma de aplicaciones con implementaciones específicas, como para suponer que el desarrollo de implementaciones de RTP es un problema solucionado satisfactoriamente. Sin embargo, estas implementaciones presentan, *sin excepción conocida*, problemas. Basta comprobar el número de implementaciones dispares que ha sido necesario emplear, no intencionalmente, para completar la matriz de implementación de RTP [5], necesaria para iniciar el proceso de paso de estado *proposed standard* a estado *draft standard*.

El análisis de esta matriz revela que las implementaciones existentes, incluyendo tanto las de distribución más o menos libre como las propietarias, llevan a la práctica el protocolo sólo parcialmente, careciendo en muchos casos de funciones básicas para la correcta interacción entre sistemas distintos[7]. Es cierto que algunas carencias son puntuales y se pueden superar con parches que no afecten más que a unos pocos archivos fuentes; sin embargo, las más importantes nos llevan a las siguientes conclusiones sobre las bibliotecas y aplicaciones de RTP existentes:

- Muchas de ellas ni siquiera implementan RTCP o lo hacen parcialmente, considerando como opcional este elemento, cuando es una parte esencial del protocolo de la que dependen funciones tan básicas como la sincronización temporal entre los participantes o la asociación de diferentes fuentes de sincronización pertenecientes a un mismo participante (como por ejemplo, asociar los flujos de imagen y sonido procedentes del mismo participante en una conferencia multimedia).

- En todos los casos conocidos, carecen de los mecanismos de control de congestión del flujo de datos requeridos por las últimas revisiones del estándar, lo que supone una grave limitación

[7] Como sincronización entre sesiones RTP, detección de colisiones o uso y formato estándar de algunos tipos básicos de paquetes RTCP.

de cara al despliegue de sistemas basados en RTP a gran escala. Dado que el diseño de estas implementaciones no contempla la incorporación de mecanismos de control de congestión o el uso de los nuevos protocolos comentados en la sección 2.1, su incorporación supone una reestructuración casi completa de la biblioteca o aplicación[8] [9].

- El problema anterior se agrava si consideramos que en la mayoría de los casos estas implementaciones carecen de o implementan sólo parcialmente los mecanismos de regulación de la transmisión de paquetes de control, establecidos en la primera especificación de RTP y ampliados en las últimas revisiones.

- En general, la actualización de las implementaciones realizadas siguiendo la primera versión del estándar [143, 141] o revisiones provisionales a la última revisión [144, 142] requeriría una reestructuración y cambios sustanciales, a pesar de que los cambios en el estándar no afectan a las funciones básicas especificadas inicialmente, sino que conllevan la modificación de algunos algoritmos aislados (como el cálculo del periodo de envío de paquetes RTCP, o la adición de un tiempo de espera previo al envío de paquetes RTCP de tipo BYE), la relajación de algunas restricciones y la incorporación de pequeñas ampliaciones puntuales [144, apéndice B] y [142, sección 9].

- Como conclusión general, entre las aplicaciones existentes, aunque algunas se utilizan actualmente con éxito [10] y ponen de manifiesto la validez y madurez de RTP, el grado de conformidad con el estándar de RTP [143, 141] no es el que cabría esperar. De hecho, son inevitables los problemas de interoperabilidad entre diferentes aplicaciones.

Considerar que las aplicaciones y bibliotecas comentadas están basadas y siguen el estándar RTP sería tan discutible como considerar que una aplicación que se limita a utilizar un socket en crudo para enviar paquetes con cabeceras TCP válidas[11], pero obviando algunos o muchos de los algoritmos de control de la transmisión de TCP, fuese una aplicación basada en TCP. Del mismo modo que la ausencia de algunos de los mecanismos de control de TCP desarrollados durante la última década produjo problemas de pérdida de prestaciones y congestión en el pasado, como se comentó en el epígrafe 2.2, si el uso de estas implementaciones se extendiese durante los próximos años tanto como cabe esperar, además de problemas de interacción entre ellas, surgirían situaciones de congestión.

Las limitaciones funcionales comentadas van asociadas y están condicionadas[12] por la falta de modularidad y genericidad en el diseño de las pilas RTP. Quizás el hecho de que RTP suele requerir adaptaciones para cada entorno ha facilitado que las implementaciones disponibles no sigan un enfoque genérico, impidiendo generalmente su reutilización. Este hecho sería justificable en aplicaciones específicas desarrolladas para entornos concretos, pero en ningún caso en aplicaciones de uso común o bibliotecas desarrolladas como base para un amplio rango de tipos de aplicaciones.

Estos problemas de diseño se ponen de manifiesto cuando se intenta ampliar o actualizar alguna de las implementaciones, o realizar componentes de mayor nivel basados en ellas, surgiendo los problemas de mantenimiento derivados de un diseño que no recoge todas las características del sistema.

[8] Nótese que, desde su definición inicial, RTP no se ha asociado a ningún protocolo de transporte o red concreto; sin embargo, siempre se ha optado por basar las implementaciones en UDP sin prever futuros cambios o ampliaciones.

[9] Asimismo, la implementación de nuevos perfiles de RTP, se dificulta en gran medida.

[10] Cabe destacar las conocidas aplicaciones de Mbone [180, 112].

[11] O incluso con algunas irregularidades.

[12] Aparte de otros condicionantes ligados a prácticas de dudosa ética.

RTP se ha diseñado como un modelo de protocolos de nivel de aplicación que sigue los principios de procesamiento integrado de niveles y segmentación de nivel de aplicación. Este diseño requiere que el diseño de programación de sus implementaciones identifique y defina interfaces modulares para cada bloque de funciones tanto de control de la transferencia como de manipulación de datos. En el caso de RTP, el enfoque de diseño que se ha seguido para aplicaciones basadas en otros protocolos de nivel de aplicación como FTP, HTTP o incluso SIP, no es adecuado, ya que las interacciones y posibles combinaciones y variaciones de las operaciones de control de la transferencia y manipulación de datos de RTP no se pueden reducir a una biblioteca de manipulación de mensajes y gestión de máquinas de estados.

En este proyecto se presenta como una posible solución GNU ccRTP [174, 123], que pretende ser un modelo de desarrollo de aplicaciones basadas en RTP, proporcionando toda la flexibilidad del modelo de protocolos al tiempo que encapsula en módulos independientes las operaciones de control de la transferencia y manipulación de datos. Para ello, sigue los siguientes principios generales de diseño:

- *Varios niveles de interfaz*. La aplicación de los principios de procesamiento integrado de niveles y segmentación de nivel de aplicación es difícil de compatibilizar con la definición de una interfaz que establezca un único nivel. Por el contrario, para ofrecer la flexibilidad que se busca con ambos principios sin que el sistema degenere en una mezcla confusa de aspectos pertenecientes a distintos niveles de detalle, GNU ccRTP ofrece una interfaz organizada en varios niveles o capas. Las aplicaciones que sólo requieran las funciones más comunes de RTP pueden así utilizar únicamente la capa más externa, mientras que aplicaciones especializadas pueden acceder mediante herencia de clases, a una segunda o incluso una tercera interfaz.

- *Genericidad respecto a servicios del sistema*. Aunque GNU ccRTP utiliza los servicios del sistema (sockets, hilos, etc) a través de las abstracciones proporcionadas por GNU Common C++, su implementación es genérica, mediante el uso de plantillas de C++, con respecto a los sockets o puntos de acceso al servicio de transporte subyacente, así como con respecto al número y coordinación de los hilos de ejecución del sistema [13].

- En todo el sistema se mantiene la *separación de elementos específicos de la recepción de datos e información de control y elementos específicos de su transmisión*. De este modo se aumenta la modularidad del sistema y es posible componer objetos que sirvan de base a aplicaciones que sólo actúen como receptores o transmisores, sin que se incorporen los elementos que no se van a utilizar. Este enfoque da lugar a la aparición de jerarquías de clases derivadas de clases base que encapsulan las operaciones comunes a los elementos de transmisión y recepción.

- Uso de métodos plantilla [100, 1, 102] como mecanismo básico para permitir el *procesamiento integrado de niveles*, de modo que redefiniendo métodos virtuales en clases derivadas de las clases de GNU ccRTP sea posible redefinir y/o complementar las operaciones de manipulación de datos y control de la transferencia.

- *Representación de los segmentos de nivel de aplicación (o unidades de datos de la aplicación) mediante objetos*. La interfaz de GNU ccRTP permite acceder a las unidades de la aplicación mediante objetos, en contraste con la práctica común de representar las unidades de datos mediante bloques de datos en crudo. Los objetos de tipo unidad de datos de aplicación encapsulan

[13] En el capítulo 5 se detallarán algunos usos de esta característica.

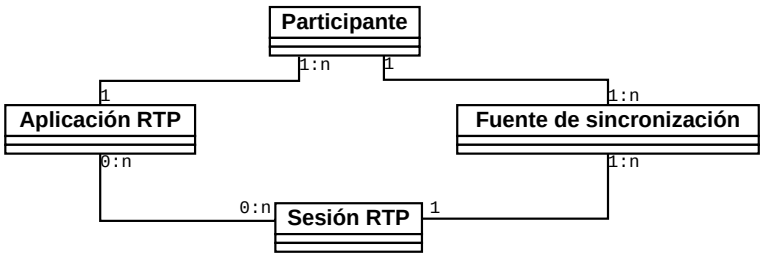


Figura 4.4: Relaciones entre participantes, fuentes de sincronización y sesiones en una pila RTP

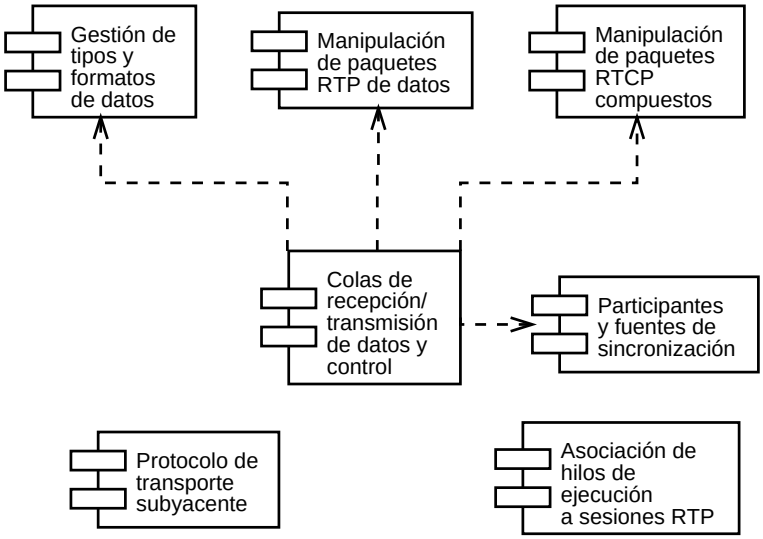


Figura 4.5: Componentes de una pila RTP

funciones del modelo RTP, como asociación de una unidad de datos a una fuente de sincronización y a un participante en una sesión.

Asimismo, siguiendo las líneas generales de diseño expuestas en las secciones previas, todos los componentes de ccRTP admiten la interacción de múltiples hilos de ejecución.

Para representar las relaciones entre fuentes de sincronización, sesiones RTP y participantes en una aplicación RTP, se ha elaborado el modelo esquematizado en la figura 4.4. En este modelo, cada fuente de sincronización en una sesión RTP está asociada a un participante en una sesión multimedia, lo que es posible gracias a la función de identificación de participantes de RTCP. Puesto que la relación entre participantes en aplicaciones RTP y sesiones RTP es indirecta (a través de las fuentes de sincronización), se contempla el caso infrecuente en que un mismo participante tenga asociadas varias fuentes de sincronización en una misma sesión RTP (como, por ejemplo, varios flujos procedentes de cámaras instaladas en la misma localización). Además, las sesiones RTP y los participantes están asociados a una aplicación RTP; la utilidad de estas dos últimas relaciones se comentará más adelante.

Durante el diseño de ccRTP, se ha considerado que, en general, una pila RTP se puede descomponer a grandes rasgos en los módulos mostrados en la figura 4.5.

- *Gestión de tipos y formatos de datos*. Define objetos para manipular, desde el punto de vista del protocolo RTP, los formatos de datos utilizados por las aplicaciones. Los objetos de este

módulo encapsulan las propiedades de los formatos que son de interés para el nivel RTP: tipo o identificador numérico de datos y frecuencia del reloj utilizado para calcular los matasellos de los paquetes.

- *Manipulación de paquetes RTP de datos*. Encapsula las operaciones de manipulación de paquetes de datos RTP, así como algunas funciones de control de la transferencia acopladas, como la validación de los paquetes. Este módulo encapsula por tanto el formato de los paquetes RTP de datos.

- *Manipulación de paquetes RTCP compuestos*. Análogo al módulo anterior, para paquetes RTCP; define abstracciones para la gestión, composición y descomposición de los paquetes RTCP compuestos así como las operaciones de control acopladas.

- *Gestión de participantes*. Encapsula el modelo de aplicaciones RTP, participantes y fuentes de sincronización.

- *Colas de recepción/transmisión de datos y control*. La cola que proporciona el servicio de transporte RTP completo, recepción y transmisión de datos y control, se obtiene mediante especializaciones de clases cola específicas para recepción y transmisión de paquetes de datos así como para gestión de paquetes de control. En ccRTP, cada cola da servicio a una sesión RTP. Estas colas se definen de manera tal que el protocolo de transporte subyacente utilizado y el modelo de hilos de ejecución seguido se especifican como parámetros de plantillas. Los objetos utilizados para instanciar estas plantillas se definen en los dos módulos siguientes:

- *Conexión con el protocolo de transporte subyacente*. Contiene un conjunto ampliable de objetos de conexión con protocolos de transporte. Todos estos elementos presentan una interfaz común usada en las colas de recepción/transmisión.

- *Asociación de hilos de ejecución a sesiones RTP*. Contiene un conjunto asimismo ampliable de objetos que asocian, mediante especialización de clases, un modelo de ejecución concreto a una cola de recepción/transmisión.

En el manual de referencia de ccRTP [123] se puede encontrar una descripción exhaustiva de la biblioteca.

ccRTP define un modelo de tipos y formatos de datos, esquematizado en la figura 4.6, que permite todos los posibles usos de los tipos de datos considerados en el estándar de RTP. Nótese la diferencia entre tipos y formatos de datos. Los primeros no son más que identificadores numéricos, mientras que los segundos se registran a través de la IANA [28], y asocian a un tipo de datos RTP un tipo MIME así como parámetros obligatorios y opcionales, además de otras aclaraciones sobre su uso (aplicaciones que los emplean, problemas de interoperabilidad, estándar que los define, etc). El modelo se compone de los siguientes elementos:

- El tipo entero de 7 bits `PayloadType`, para los identificadores numéricos de tipos de datos.

- El tipo enumerado `StaticPayloadType`, que define los códigos de tipos de datos asignados de forma estática hasta la fecha (véase [142].

- La clase `StaticPayloadFormat`, que asocia a un tipo de datos los parámetros necesarios para el correcto funcionamiento de la pila RTP (frecuencia de reloj).

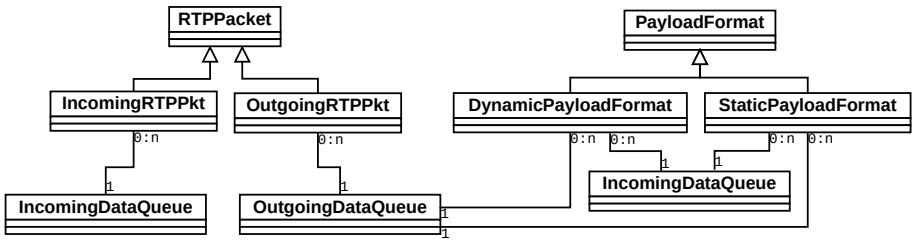


Figura 4.6: Módulos de manipulación de paquetes RTP y tipos y formatos de datos

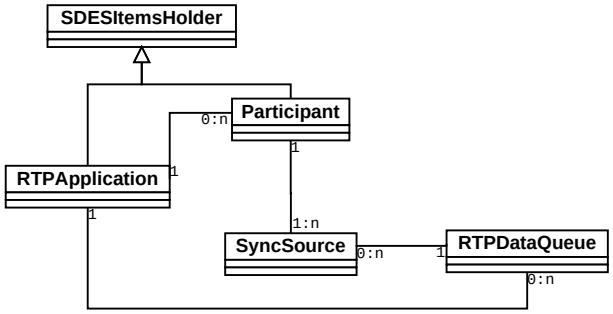


Figura 4.7: Módulo de gestión de participantes

- La clase `DynamicPayloadFormat`, para representar formatos negociados de forma dinámica mediante protocolos de control. Los objetos de esta clase asocian identificadores de tipo de datos a frecuencias de reloj. Tal y como se especifica en el estándar, los tipos asignados de forma estática pueden utilizarse para definir nuevos formatos dinámicos.

En la figura 4.6 se presenta un esquema del módulo de manipulación de paquetes de datos, que proporciona las operaciones antes mencionadas de forma separada para la gestión de paquetes entrantes y salientes, manteniendo por tanto independientes las operaciones relacionadas con funciones de transmisión y recepción.

El módulo de manipulación de paquetes compuestos RTCP encapsula el tratamiento de los paquetes RTCP en la clase `RTCPCompoundHandler`, que contiene y se relaciona con otras clases de menor nivel. Esta clase proporciona métodos de composición y descomposición de paquetes RTCP compuestos, entre los cuales se encuentran métodos que permiten definir y acceder a los campos y cabeceras de ampliación contemplados en la especificación de RTP.

La figura 4.7 esquematiza el módulo de gestión de participantes. En él, se define la clase `Participant`, que representa participantes remotos, y la clase `RTPApplication`, que representa al participante local. Asimismo, se define la clase `SyncSource`, utilizada para representar a las fuentes de sincronización RTP. Esta última clase proporciona acceso a los datos recibidos, las estadísticas acumuladas a lo largo de las sesiones y las estadísticas que se puede haber recibido mediante paquetes RTCP. Cada objeto de tipo `SyncSource` tiene asociado un objeto `Participant` (siempre y cuando los paquetes RTCP necesarios para determinar la asociación se hayan recibido), que permite acceder a la información de control de la sesión RTP. La función de la clase `RTPApplication` se detalla en la descripción del módulo de colas.

El módulo de colas de recepción/transmisión de datos y control constituye el núcleo de ccRTP. Las clases de este módulo definen una cola de paquetes RTP de datos entrantes y una cola de paquetes

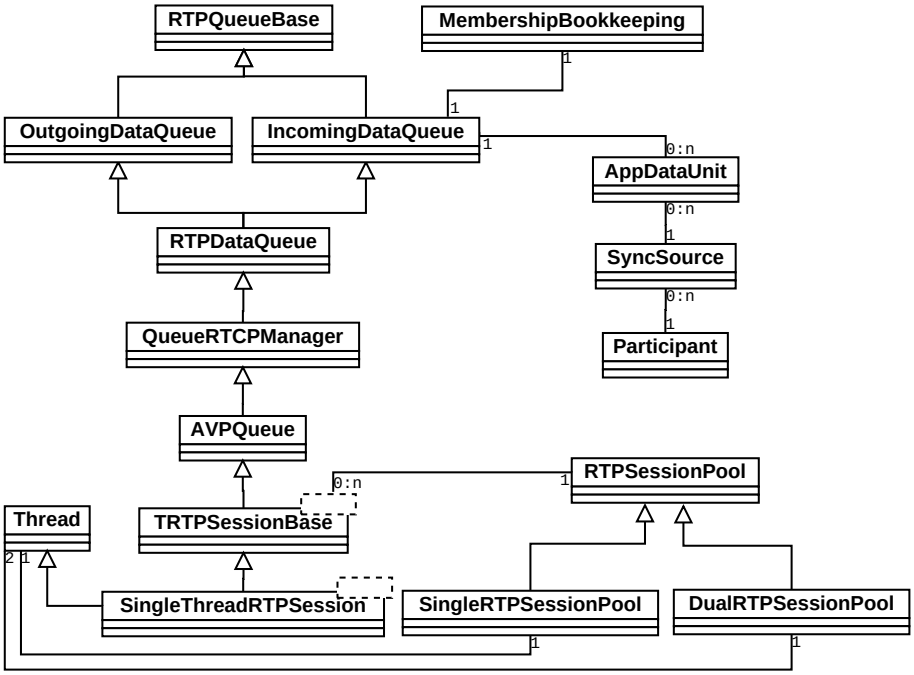


Figura 4.8: Diagrama de clases simplificado de ccRTP

RTP de datos salientes. Ambas colas, junto con las clases de gestión de paquetes RTCP, son los componentes de mayor importancia de las sesiones RTP en el modelo ccRTP.

Al implementar el nivel de transporte RTP tomando como base sendas colas de transmisión y recepción, es posible integrar en la pila RTP funciones de control de la transferencia no acopladas a las funciones de manipulación de datos, así como funciones que, sin pertenecer estrictamente al nivel RTP, aparecen generalmente asociadas a éste en las aplicaciones que utilizan RTP como protocolo de transporte.

Ambas colas constituyen un modelo de aplicación del principio de procesamiento integrado de niveles al protocolo RTP, realizando funciones como control de congestión, gestión de *buffers* de entrada/salida, reordenación de unidades de datos, y otras funciones que, estando generalmente encapsuladas en la capa de transporte, en RTP forman parte del nivel de aplicación.

La figura 4.8 muestra el diagrama de clases del módulo de colas así como de parte de los módulos de asignación de hilos de ejecución y gestión de participantes. En ella se aprecia que las funciones de transmisión y recepción están definidas por clases independientes (`OutgoingDataQueue` y `IncomingDataQueue`, respectivamente) que heredan las características comunes de una clase base virtual, `RTPQueueBase`.

Tanto la cola de transmisión como la de recepción se han diseñado de modo que todos aquellos algoritmos y parámetros que son modificables dentro del modelo RTP se puedan modificar y redefinir mediante técnicas de programación aplicables en C++, como métodos plantilla, herencia o especialización, instanciación de plantillas, delegación de métodos y otros patrones [1, 101, 102, 51, 100, 173] comunes. Las funciones desempeñadas por las colas con:

- *Cola de transmisión* (clase `OutgoingDataQueue`). Con ccRTP las unidades de datos de la aplicación no se transmiten directamente, sino que se entregan a la pila RTP para que sean insertados en una cola de paquetes salientes. Como caso particular, se puede configurar la cola

de paquetes salientes de manera que las unidades se transmitan inmediatamente. Sin embargo, la introducción de este paso intermedio permite incorporar funciones de control de la transferencia a la pila RTP, en lugar de quedar delegadas en la aplicación.

La función básica de la cola de paquetes salientes es proporcionar un mecanismo de transmisión de unidades de datos de aplicaciones RTP suficientemente general como para dar base a la implementación de:

- Algoritmos de inserción de retrasos artificiales (para aumentar la regularidad en el periodo de transmisión).
- Fragmentación de unidades de datos (definiendo un tamaño máximo, modificable, de las unidades transportadas).
- Control de congestión (en función del tipo de servicio de la red) de nivel de aplicación.
- Gestión de listas de destinatarios.
- Gestión de situaciones de sobrecarga de la aplicación (suspendiendo la transmisión de paquetes si no se realiza en un intervalo).
- Cálculo de estadísticas de la transmisión.

Estos algoritmos son redefinibles bien mediante la selección de valores de algunos parámetros, o bien mediante especialización de la clase y redefinición de métodos plantilla. La interfaz pública de esta clase define métodos básicos para la inserción de unidades de datos en la cola de paquetes salientes y la modificación de los parámetros de uso más extendido, como la lista de destinatarios o el tamaño del segmento de transporte.

- *Cola de recepción* (clase `IncomingDataQueue`). Proporciona servicios de recepción de paquetes RTP de datos. La existencia de esta cola como paso intermedio entre la recepción de unidades de datos y su entrega a la aplicación permite aplicar el modelo de participantes y fuentes de sincronización comentado anteriormente, siendo la cola de recepción la encargada de asociar cada unidad de datos recibida a una fuente de sincronización y al objeto participante correspondiente.

  La interfaz pública de la clase `IncomingDataQueue` proporciona los métodos necesarios para extraer unidades de datos de la cola de paquetes entrantes, permitiendo aplicar filtros por diversos parámetros, como matasellos o fuente de sincronización origen de la unidad.

  El acceso a las unidades de datos recibidas se realiza mediante la clase `AppDataUnit`, que es la abstracción de la unidad de datos de aplicación de los sistemas con segmentación de nivel de aplicación aplicada a los paquetes entrantes en ccRTP. Esta clase encapsula las relaciones con participantes, fuentes de sincronización y sesiones RTP.

  Son funciones de la cola de entrada:

  - Validación de paquetes RTP.
  - Construcción y gestión del conjunto de fuentes de sincronización existentes en cada sesión.
  - Asociación de unidades de datos con fuentes de sincronización y participantes.

- Comprobación de la unicidad de los identificadores de fuentes de sincronización y detección de bucles en la red RTP (que puedan haberse introducido como consecuencia de la incorrecta configuración de elementos intermedios de nivel RTP).
- Gestión de situaciones de sobrecarga de la aplicación (descartando unidades de datos tras un cierto intervalo).
- Reordenación de paquetes de manera controlable y parametrizable por parte de la aplicación.
- Amortiguación de los flujos de entrada (implementando un *buffer* de reordenación).
- Cálculo de estadísticas de recepción.

Nótese que las colas descritas hasta ahora no proporcionan los servicios del protocolo RTCP, por lo que son útiles como base de implementaciones específicas desarrolladas para entornos en los que el uso de RTCP no es posible (como en aplicaciones que utilizan enlaces unidireccionales).

Los servicios RTCP se añaden a la jerarquía de clases de colas, mediante herencia, con la clase `QueueRTCPManager`, que, basándose en el módulo de gestión de paquetes RTCP, proporciona funciones como sincronización entre diferentes flujos, identificación de participantes y recepción/transmisión de paquetes RTCP compuestos siguiendo las restricciones temporales y de uso de ancho de banda especificadas por RTP. La clase `QueueRTCPManager` incorpora mecanismos redefinibles de procesado genérico de paquetes RTCP, encargándose tanto de la extracción de estadísticas acerca de las fuentes de sincronización, como de la generación de estadísticas para los paquetes RTCP salientes. Esta clase integra asimismo las operaciones de temporización del envío de paquetes RTCP.

Como se destacó anteriormente, el módulo de gestión de participantes proporciona los elementos necesarios para que cada sesión RTP se pueda asociar a una aplicación RTP (un objeto de la clase `RTPApplication`). De este modo, cuando se dispone de los servicios de RTCP (en particular, del servicio de identificación de participantes), una misma aplicación o proceso basado en ccRTP puede implementar simultáneamente varias aplicaciones RTP; es decir, ser participante en varias aplicaciones RTP. Por tanto, se tienen varios espacios de identificadores canónicos, al igual que dentro de cada aplicación RTP existen varios espacios de identificadores de fuente de sincronización (uno para cada sesión RTP).

La clase `AVPQueue` implementa, mediante especialización de `QueueRTCPManager` una cola de recepción y transmisión de paquetes RTP y RTCP siguiendo el perfil para sonido y vídeo [141, 142]. Esto es, define una clase base para sesiones RTP añadiendo la restricciones específicas que el perfil para sonido y vídeo impone sobre la gestión de paquetes RTCP.

La plantilla `TRTPSessionBase` tiene como parámetros el tipo concreto de cola y los objetos que proporcionan el servicio de transporte del protocolo subyacente (para datos y control por separado). Esta plantilla define una sesión RTP para el perfil de sonido y vídeo, sin asignar ningún modelo de ejecución. Así es posible definir otras clases que incorporen este aspecto con plena libertad.

Para que el módulo de colas proporcione funciones concretas, requiere clases de acceso al servicio de transporte subyacente y a los servicios de hilos del sistema. Las primeras, proporcionadas por el módulo de conexión con el protocolo de transporte subyacente, se especifican como parámetros de la plantilla `TRTPSessionBase`. Las segundas se concretan en clases derivadas de `TRTPSessionBase` o clases relacionadas con ella.

En el diagrama de clases de la figura 4.8 se muestran las dos opciones de asociación de hilos de ejecución proporcionadas actualmente por ccRTP:

- La plantilla `SingleThreadRTPSession`, que asigna un hilo de ejecución para servir tanto la conexión de datos como la de control de una sesión RTP.

- La jerarquía de clases derivada de la clase `RTPSessionPool`, que asocia hilos de ejecución a conjuntos de sesiones RTP. Entre las clases de esta jerarquía, se encuentran `SingleRTPSessionPool` y `DualRTPSessionPool`, que asignan un único hilo de ejecución, en el primer caso; y un hilo de ejecución para el servicio de datos y otro para el servicio de control, en el segundo caso.

No obstante, todas las funciones de una sesión RTP están contenidas en la plantilla `TRTPSessionBase`, que puede utilizarse independientemente de los servicios de sockets e hilos contemplados en ccRTP.

En ccRTP, la clase sesión RTP (`RTPSession`) se define como el caso básico de sesión RTP[14] (una sesión RTP según el perfil de sonido y vídeo, basada en sockets UDP sobre IP versión 4 con un hilo de ejecución que sirve las conexiones de datos y control). Este caso se obtiene instanciando la plantilla `SingleThreadRTPSession` (derivada de `TRTPSessionBase` y la clase de GNU Common C++ `Thread`) con las clases `DualRTPUDPIPv4Channel` y `AVPQueue`. `DualRTPUDPIPv4Socket` se utiliza para proporcionar la conexión de transporte, tanto para datos como para control, sobre UDP/IPv4, mientras que `AVPQueue` aporta una cola de servicio de datos y control que sigue el perfil para sonido y vídeo (AVP). El modelo de hilos se aplica redefiniendo el método `Thread::run()` en la plantilla `SingleThreadRTPSession` de modo que ejecute los métodos de servicio de datos y control definidos en este caso por la clase `AVPQueue`.

El diseño de ccRTP, aunque incorpora toda la complejidad del modelo RTP, permite que el uso básico de la clase `RTPSession` (uso de su interfaz pública) no requiera un conocimiento detallado del diseño de las clases superiores de la jerarquía. Los diferentes niveles definidos en la jerarquía de clases que da base a las sesiones RTP establecen diferentes niveles de interfaz [1, 173, 51] a los que sólo es necesario acceder conforme sea necesario redefinir el comportamiento de los correspondientes niveles del sistema. Considérese como ejemplo la clase `AVPQueue`, que, para redefinir las reglas de uso del ancho de banda reservado a paquetes RTCP, únicamente requiere el acceso a algunos de los métodos protegidos de la clase `QueueRTCPManager`.

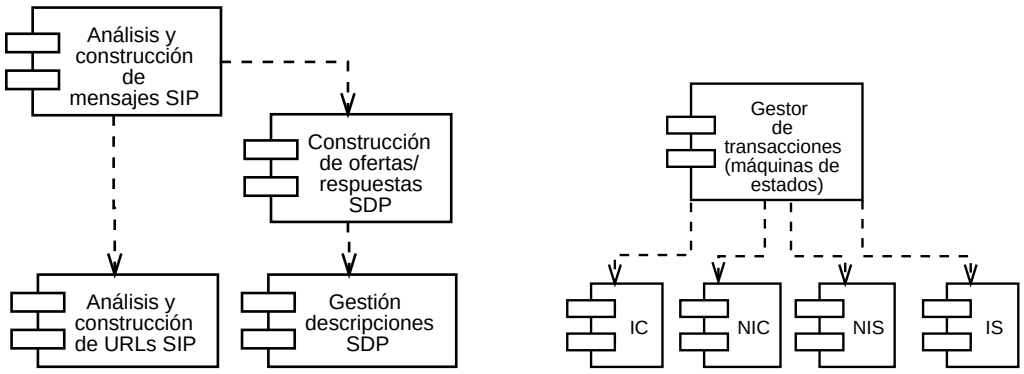
Finalmente, y retomando el problema de la dispersión de dependencias de aspectos del sistema [83], nótese cómo las dependencia de los servicios relacionados con hilos y sockets están concentradas en los módulos de conexión con el protocolo de transporte subyacente y de asociación de hilos de ejecución a sesiones RTP, siguiendo un enfoque similar al de la programación orientada a aspectos. Así, todos los servicios proporcionados por los cuatro módulos restantes son reutilizables independientemente de los dos aspectos comentados.

4.6. GNU oSIP

GNU oSIP [107] es una biblioteca escrita en C para la manipulación de mensajes y la gestión de transacciones SIP. Del mismo modo que el diseño del protocolo SIP tiene como objetivo reducir la carga de los servidores SIP, oSIP se ha diseñado para que sea válida tanto para la realización de agentes de usuario como para servidores SIP.

En general, una aplicación basada en SIP requiere el desarrollo de cuatro bloques de funciones:

[14] El que normalmente se tiene en cuenta en la mayoría de las implementaciones actuales.



(a) Análisis y construcción de mensajes (b) Transacciones

Figura 4.9: Módulos de oSIP

1. Servicios básicos de transporte de mensajes.
2. Manipulación de mensajes SIP.
3. Gestión de transacciones, en particular de los temporizadores de las transacciones relacionados con las retransmisiones e intervalos de expiración de transacciones.
4. Gestión del estado de las sesiones SIP.

oSIP cubre los bloques 2 y 3. Dado que el bloque 1 no se implementa en oSIP, no se establece ninguna dependencia de los servicios de sockets del sistema (que puede proporcionar Common C++). Respecto al bloque 4, su realización no es necesaria para implementar los servidores sin estados, que representan la mayoría de los servidores desplegados hasta la fecha[15].

oSIP está dividida en los dos componentes esquematizados en la figura 4.9: el analizador y constructor de mensajes SIP, y el gestor de transacciones. Estos componentes establecen un modelo de programación de aplicaciones basadas en SIP según el cual oSIP gestiona las máquinas de estados asociadas a las transacciones SIP y proporciona métodos para manipular los mensajes SIP.

Por tanto, el desarrollo de una aplicación basada en oSIP requiere, en lo referente al control de sesiones, definir un conjunto de métodos encargados de la transmisión/recepción de los mensajes así como la lógica que interrelacione las transacciones SIP de cada llamada.

A continuación se describe el diseño de los dos componentes de oSIP:

- *Analizador y constructor de mensajes SIP*. Cumple las funciones de análisis y construcción de peticiones y respuestas SIP. Contempla la gestión de mensajes con varios adjuntos, y proporciona métodos de manipulación básica de descripciones SDP.

  Los mensajes SIP se manipulan mediante objetos que representan los componentes de los mensajes SIP (véase la sección 3.4.3), a los que se accede mediante métodos de consulta y modificación.

[15] En el capítulo 5 se analizará el problema de la incorporación de las funciones de gestión de estados de sesiones.

El analizador sintáctico de mensajes SIP que encapsula este módulo es permisivo, es decir, se opta por no realizar las comprobaciones innecesarias para descomponer los mensajes en los elementos a los que puede acceder la aplicación. Dado que el analizador no certifica por completo la validez de los mensajes, algunos de los valores proporcionados por los métodos de consulta sobre mensajes recibidos pueden no ser válidos. Por ejemplo, se puede admitir un URL no válido en la cabecera *Via:*, sobre la que únicamente se comprueba el primer valor de la lista de URLs.

De este modo, se minimiza el número de operaciones realizadas por el analizador básico, aumentando así el número de transacciones que es posible gestionar a igualdad de recursos. Este diseño permite que el rendimiento de las aplicaciones basadas en SIP sea, además de mayor de lo que sería si se validase por completo los mensajes, más estable respecto a la complejidad de las cabeceras de los mensajes. No obstante, oSIP proporciona los métodos base para implementar ampliaciones con objeto de validar todos los campos de las cabeceras de los mensajes.

El mismo enfoque de diseño se aplica para la construcción de mensajes. Los métodos de modificación y de consulta de mensajes proporcionados por oSIP permiten establecer los valores de los campos de la cabecera de los mensajes SIP y los elementos de los campos estándar. Asimismo, se prevé la posibilidad de construir y obtener campos de ampliación de forma transparente a la biblioteca.

Esto es, mientras el valor de los campos desconocidos se extrae como una cadena de texto, el valor de los campos conocidos se extrae como un conjunto de parámetros dependiente de la cabecera (por ejemplo, el código numérico de las respuestas). Del mismo modo, un campo se puede añadir a un mensaje mediante métodos específicos para campos conocidos o mediante un método genérico que toma como parámetro una cadena de texto. oSIP gestiona de forma especializada todos los métodos básicos y de ampliación descritos en la sección 3.4, a excepción del método PRACK.

Este módulo se apoya en dos módulos auxiliares:

- *Módulo de negociación según el modelo oferta/respuesta con SDP [129]*. Proporciona métodos de generación automática de respuestas a partir de ofertas. Las respuestas se generan teniendo en cuenta parámetros de configuración, previamente establecidos, como los tipos de medios y formatos de codificación admitidos o los puertos válidos para conexiones de datos.
- *Módulo de gestión de URLs SIP*, que contiene métodos específicos para manipular los URL SIP, en particular los parámetros comunes en estos URL, como *transport* o *phone*, véase la sección 3.3.1.

Ambos módulos son accesibles para las aplicaciones basadas en oSIP, sirviendo así de base para realizar ampliaciones o especializaciones de la biblioteca.

- *Gestor de transacciones*. Este módulo encapsula las máquinas de estados correspondientes a los cuatro tipos de transacciones incluidos en la última revisión de SIP [133, secciones 17.1 y 17.2], que definen los posibles estados y los sucesos que dan lugar a transiciones entre estados. Los cuatro tipos de transacciones son los siguientes:

  1. Transacciones de tipo INVITE (iniciadas por una petición INVITE) en los UAC (IC en la figura 4.9).

2. Transacciones de tipo INVITE en los UAS (IS).
3. Transacciones de tipo no INVITE en los UAC (NIC).
4. Transacciones de tipo no INVITE en los UAS (NIS).

Las transacciones de clientes (1 y 3) se inician localmente, mientras que las de servidores (2 y 4) se inician en una máquina remota. Para cada tipo de transacciones se definen cinco estados entre los que se producen transiciones debido a sucesos de varios tipos: inicio de la transacción, transmisión/recepción de respuestas provisionales, recepción de mensajes repetidos, retransmisión de mensajes previamente transmitidos tras expirar el tiempo de espera a respuestas (en el caso en que el protocolo de transporte sea no fiable), y terminación de la transacción.

oSIP permite asociar un método a cada uno de estos sucesos para cada tipo de transacción. Aparte de algunos métodos globales (como error de recepción), estos métodos se definen sólo para los tipos de transacciones que proceda; por ejemplo, los correspondientes a respuestas provisionales solo se definen para transacciones en clientes

A cada transacción se le asocia una cola de sucesos en espera de ser atendidos, estableciéndose por tanto un *modelo de programación basado en una secuencia de sucesos atendidos por métodos redefinibles*. Los sucesos de cada transacción se procesan en secuencia. Para iniciar transacciones, oSIP proporciona métodos que permiten construir las peticiones SIP, así como métodos que permiten crear, a partir del mensaje de petición construido, una nueva transacción y, con ello, una cola de sucesos. Del mismo modo, en el caso de los servidores, una transacción se puede crear a partir de una petición SIP recibido.

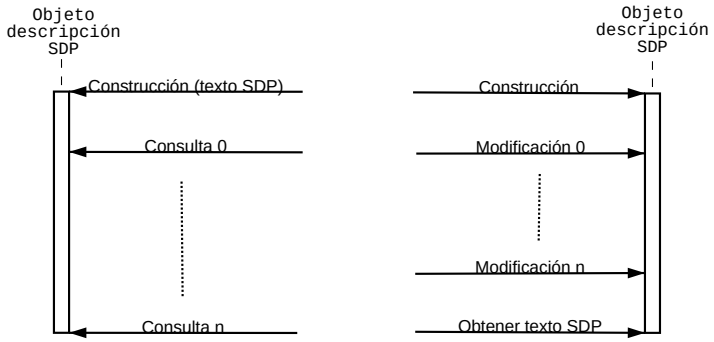
Nótese que el gestor de transacciones es independiente del modelo de ejecución utilizado. Es decir, proporciona métodos para procesar las colas de sucesos asociadas a las transacciones, pero éstos han de ser ejecutados por algún hilo de ejecución definido en la aplicación. oSIP define un método para procesar el siguiente suceso de cualquier tipo de transacción, así como otros para procesar el siguiente suceso de un tipo particular de transacción o de una transacción en concreto. Por tanto, es posible cualquier esquema de asociación de hilos de ejecución a transacciones.

4.7. FreeSDP

La complejidad del protocolo SDP y, como consecuencia la de los sistemas que lo implementan es notablemente menor a la de RTP. Por ello, FreeSDP se ha programado en C; sin embargo, se ha seguido un diseño orientado a objetos, con la particularidad de que el único tipo de objetos destacable del diseño es el *tipo descripción SDP*.

Puesto que el protocolo SDP no es más que un formato de descripción, el tratamiento que requiere es simple y encaja en los patrones de diseño tradicionales. El diseño que se propone con FreeSDP define dos módulos independientes: análisis sintáctico y extracción de la información contenida en las descripciones en formato SDP, y construcción de descripciones en formato SDP a partir de los parámetros de la sesión. A continuación se detalla el diseño de ambos módulos, esquematizados, respectivamente, en las figuras 4.10(a) y 4.10(b):

- *Análisis sintáctico y extracción de información*. Este módulo se basa en un constructor de objetos descripción que, tomando como parámetro una descripción SDP textual, construye un



(a) Análisis sintáctico y extracción de información

(b) Generación en sintaxis SDP

Figura 4.10: Procesado de descripciones SDP con FreeSDP

objeto de tipo descripción tras comprobar la sintaxis. De este objeto se puede extraer toda la información contenida en la descripción original mediante métodos de consulta.

- *Construcción de descripciones SDP*. Engloba un constructor de objetos descripción, en un conjunto de métodos de modificación, y en un método de volcado de la información en una descripción SDP textual. El constructor de este módulo construye un objeto descripción a partir de la información imprescindible para una descripción SDP: versión del protocolo, nombre de la sesión y otros parámetros hasta un total de 10. La interfaz de este módulo define además un conjunto de métodos de modificación que permite añadir nuevas propiedades a la descripción. Estos métodos mantienen en todo momento el objeto descripción en un estado que sigue el estándar. Finalmente, un último método proporciona una descripción SDP textual a partir de un objeto descripción.

Ambos módulos dependen de un conjunto de rutinas de implementación y de tipos comunes (como tipos de redes, tipos de protocolos de transporte, tipos de modificadores del parámetro ancho de banda y otros parámetros de las descripciones SDP).

Como se comentó en la sección 3.6, SDP presenta limitaciones como protocolo de negociación de capacidades. Por este motivo, en el IETF se ha adoptado la decisión de no seguir definiendo nuevas ampliaciones, en la medida de los posible, una vez que se termine la revisión del estándar actualmente en desarrollo [58] y la definición de las ampliaciones esenciales para incorporar los mecanismos de negociación de capacidades imprescindibles para el despliegue de sistemas multimedia. Este hecho, unido a que el mecanismo de ampliación previsto por SDP es simplemente la definición de nuevos atributos y/o valores de atributos, hace posible que se pueda añadir cualquiera de las ampliaciones propuestas sin requerir cambios en la estructura de la biblioteca. En la sección 5.4 se enumerarán las ampliaciones a SDP actualmente implementadas en FreeSDP.

Capítulo 5

Implementación

Durante el desarrollo del sistema presentado en este proyecto se ha seguido un conjunto de normas y prácticas previamente definidas como resultado de la experiencia acumulada lo largo de años de desarrollo de software libre. El presente capítulo expone estas normas y prácticas, aplicadas a todos los componentes del sistema, así como los detalles de implementación más relevantes de cada uno de estos componentes.

Teniendo en cuenta que los lenguajes de programación utilizados son C y C++, se ha seguido un conjunto de recomendaciones [173, 101, 102, 177, 182, 43] con objeto de garantizar la modularidad y reusabilidad de los componentes. En cuanto a la portabilidad del sistema, se han aplicado técnicas que vienen siendo utilizadas con éxito en numerosos sistemas de software libre [182, capítulos 15 y 16] y [164].

Todos los componentes del sistema comparten el mismo conjunto de estándares de codificación, compilación, construcción de bibliotecas y empaquetado de fuentes. Asimismo, utilizan los mismos formatos y herramientas de generación de documentación.

La organización del árbol de archivos fuente y el esquema de empaquetado de los componentes se ha realizado siguiendo los estándares del proyecto GNU [165, 164, 182], utilizando para ello un grupo de herramientas [182] que simplifican la aplicación de estos estándares así como otras tareas: *Autoconf* [95, 154, 157], *Automake* [96] y *Libtool* [110]. GNU *Autoconf* proporciona un entorno para la definición de comprobaciones durante la configuración del paquete antes de su compilación, determinando las características de compilación y ejecución del sistema para el que se construye el paquete. GNU *Automake* simplifica y automatiza la creación de reglas de compilación de los paquetes, proporcionando como resultado plantillas de *makefiles* [160] estándar y portables. Junto con *Autoconf* automatiza, además de la compilación y enlazado, tareas como la creación de distribuciones de fuentes o aplicación de juegos de pruebas. Por último, GNU *Libtool* proporciona las funciones necesarias para generar bibliotecas tanto estáticas como compartidas en diferentes plataformas [110], encapsulando las características específicas de cada una de ellas tras una interfaz común.

Estas herramientas, junto con otras auxiliares [118], proporcionan la infraestructura necesaria para que el sistema se pueda construir correctamente en diversas plataformas y con distintos compiladores, dando lugar tanto a bibliotecas estáticas como a bibliotecas compartidas o DDLs. Entre otras, el sistema se ha probado con éxito en plataformas tales como distribuciones de GNU/Linux, variantes del sistema BSD, Hurd, Win32, Solaris o HP-UX, utilizando tanto GCC como los respectivos compiladores propietarios. Dadas las características específicas del sistema Win32, se proporcionan

además *makefiles* para las plataformas Cygwin y MinGW32, el compilador Borland C++, y archivos de proyecto para Visual C++.

Asimismo, se han empleado herramientas de auditoría de seguridad, como *Flawfinder* [187] y *RATS* [153], siguiendo, además de las técnicas expuestas en las referencias dadas anteriormente, algunas de las recomendaciones dadas en [188, 189, 186].

En cuanto a la documentación del sistema, se ha utilizado *Doxygen* [62] como herramienta de generación automática de manuales de referencia. Esta herramienta, válida para diversos lenguajes (C, C++, Java e IDL, entre otros) permite generar, a partir de bloques de comentarios y marcas en el código fuente, manuales de referencia en formatos man, HTML, LaTeX y otros, incluyendo diagramas de clases y permitiendo enlazar manuales de diferentes paquetes. Los manuales de Common C++, ccRTP y FreeSDP se distribuyen asimismo junto con los fuentes y se pueden generar en diversos formatos electrónicos a partir del texto en formato TeXInfo [29].

5.1. GNU Common C++

Common C++ define en general clases base sencillas, permitiendo por tanto que las bibliotecas y aplicaciones que la utilizan sean eficientes tanto en uso de memoria como volumen de operaciones. Aunque aprovecha muchas de las características del último estándar de C++ [173], utiliza mecanismos que permiten compilar y utilizar la biblioteca aun en compiladores simples y entornos restringidos. En este sentido, Common C++ utiliza espacios de nombres y algunas de las cabeceras estándar definidas recientemente; sin embargo, puesto que estos elementos no siempre presentes en todos los sistemas, es posible utilizar la biblioteca sin ellos.

Actualmente, Common C++ consta de dos bibliotecas que se construyen de forma independiente: *ccgnu2*, que constituye el núcleo de Common C++ (clases relacionadas con procesos, hilos, sincronización, entrada/salida, temporización y sockets), y *ccext2*, que incluye las clases de mayor nivel, como las relativas a protocolos de Internet, persistencia o manipulación de fechas.

Puesto que Common C++ define un modelo de aplicaciones amplio, con numerosas opciones de compilación dependientes de la plataforma, se distribuye junto con un script, *ccgnu2-config*, que proporciona las opciones adecuadas de compilación y enlazado de aplicaciones que utilizan Common C++.

Entre los elementos más importantes de Common C++, destaca su modelo neutral de procesos, hilos de ejecución y mecanismos de sincronización, que proporciona todos los servicios requeridos por un amplio rango de aplicaciones y es válido para todas las plataformas a las que se ha adaptado Common C++. No obstante, mediante clases de ampliación, proporciona acceso a servicios específicos, como las señales de los sistemas POSIX o los eventos de los sistemas Win32. Algunos de estos servicios se emulan asimismo para las plataformas en las que no están disponibles, mientras que aquellos cuya emulación tendría limitaciones sólo están disponibles en los sistemas que los proporcionan directamente.

Common C++ presta asimismo servicios de gestión de memoria específicos para sistemas multiprocesador, utilizando opcionalmente la biblioteca Hoard [12], una implementación del servicio de reserva dinámica de memoria que es altamente escalable y permite que el sistemas aproveche las posibilidades de las arquitecturas multiprocesador de memoria compartida.

El modelo de sockets y direcciones de Internet de Common C++ admite actualmente el uso de

IPv4, proporcionando clases base para el uso de sockets TCP y UDP, tanto en modos de transferencia unicast como multicast. Este modelo es, sin embargo, suficientemente general como para admitir, mediante herencia y delegación [100, 173] ampliaciones que incorporen el protocolo IPv6.

Asimismo, como trabajo futuro, se considera la posibilidad de incorporar clases para los protocolos de transporte SCTP y UDP Lite. En el caso de SCTP, la interfaz de programación se está definiendo actualmente [172] en base a las conocidas interfaces para UDP y TCP, por lo que su incorporación al sistema de flujos de entrada/salida de Common C++ es factible tan pronto como las implementaciones de este protocolo en los sistemas operativos estén más extendidas que en la actualidad. En el caso de UDP Lite, puesto que la única modificación con respecto a UDP que su uso supone en el nivel de aplicación es la definición de un nuevo código de protocolo de redes IP –que comparte comportamiento e interfaz de programación con UDP–, tan pronto como UDP Lite se implemente en sistemas operativos de uso general, puede ser fácilmente incorporado a Common C++.

5.2. GNU ccRTP

Dado que ccRTP se ha diseñado como una biblioteca genérica para RTP, implementa todos los algoritmos definidos en el estándar, siguiendo la revisión actual [144], por lo que es válida tanto para aplicaciones de conferencia de propósito general como para traductores o mezcladores.

La implementación de ccRTP viene determinada por el hecho de que pretende ser un modelo de desarrollo de aplicaciones que no imponga ningún diseño particular en las aplicaciones que la utilicen. Por ello, busca la descomposición ortogonal del mayor número de elementos, de manera que éstos se puedan combinar para dar lugar a diferentes variantes del modelo RTP. Como consecuencia de este enfoque, aunque las aplicaciones pueden acceder a todos los módulos de la biblioteca mediante la cabecera `cc++/rtp.h`, se definen varias cabeceras específicas para módulos que pueden utilzarse de manera independiented, como `cc++/rtpbase` (que define los elementos de los módulos de gestión de tipos y formatos de datos), `cc++/rtpqueue` (que define los elementos del módulo de colas de recepción/transmisión) o `cc++/rtpext.h` (que define diversas clases de ampliación de la biblioteca).

Con el objetivo descomponer la pila RTP en el mayor número posible de componentes reutilizables, ccRTP combina diferentes técnicas de programación orientada a objetos, programación genérica y programación orientada a aspectos. En los diferentes módulos de ccRTP se emplean las siguientes técnicas:

- Mecanismos de herencia, delegación y métodos plantilla [100, 99, 173], para permitir la redefinición y ampliación de métodos, así como para independizar los módulos de servicios prestados por otros módulos.
- Genericidad mediante plantillas de C++ [1, 173, 43], para generar pilas con protocolos de transporte/red y comportamiento de la cola de recepción/transmisión instanciables.
- Combinación de plantillas y herencia de clases [1] para proporcionar plantillas que permitan la definición de objetos sesión RTP mediante la combinación de diversos aspectos (modelo de ejecución, variedad de cola de transmisión/recepción y tipo de protocolo de transporte subyacente).

Por una parte, como se observa parcialmente en la figura 4.8 (página 71), las clases de mayor complejidad se definen a partir de varias clases base mediante herencia múltiple. En la figura se muestra cómo la clase `IncomingDataQueue` deriva de las clases `RTPQueueBase` y `MembershipControl`, estando esta última encargada de la gestión de la tabla de fuentes de sincronización activas en la sesión. Este mismo esquema se repite en otros casos, como la clase `OutgoingDataQueue`, que deriva, además de `RTPQueueBase`, de la clase `DestinationListHandler`, encargada de la gestión de la lista de destinatarios de unidades de datos. Asimismo, a lo largo de toda la jerarquía de clases del módulo de colas de recepción/transmisión, los métodos proporcionados por otros módulos (en particular los relacionados con el uso del protocolo de transporte subyacente) se definen como métodos virtuales puros que forman parte de métodos plantilla.

Por otra parte, el uso de plantillas permite definir tipos genéricos, como la plantilla `TRTPSessionBase`, mediante un mecanismo flexible de polimorfismo en tiempo de compilación [1], lo que ahorra la pérdida de eficiencia que conllevan otros mecanismos de polimorfismo, al igual que en el caso de la biblioteca estándar de plantillas de C++.

Siguiendo el modelo de datos presentado en la figura 4.4 (página 68), ccRTP permite recorrer la lista de participantes en un aplicación, así como la lista de fuentes de sincronización activas en una sesión RTP. Estas operaciones se realizan a través de sendos iteradores, definidos de forma similar a los iteradores de tipo `const_iterator` de la biblioteca estándar de plantillas de C++ [6, 88]. Por ello, los objetos `RTPApplication` pueden considerarse contenedores de objetos *Participant*, y, del mismo modo, los objetos *RTPQueue* (y, por extensión, todos los tipos de sesiones RTP que se definan a partir de esta clase), se pueden ver como contenedores de objetos *SyncSource*.

El elemento de mayor complejidad en ccRTP es la la cola de paquetes entrantes de cada sesión RTP (clase `IncomingDataQueue`, que gestiona unidades de datos que deben ser asociadas a las diferentes fuentes de sincronización presentes en la sesión. La estructura de la implementación actual de esta cola se muestra en la figura 5.1.

En la figura se muestra asimismo la estructura de la lista de paquetes salientes, actualmente implementada en la clase `OutgoingDataQueue`, y de estructura más simple. Se trata de una cola donde se introducen las unidades de datos proporcionadas por la aplicación, sobre las que se aplican las operaciones resumidas en el apartado de diseño antes de su transmisión.

La cola de paquetes entrantes de ccRTP está formada realmente por un número variable de colas; una cola global que contiene todas las unidades recibidas en espera de ser extraídas y procesadas, y un conjunto de colas individuales para cada fuente de sincronización distinta. Por tanto, cada unidad de datos recibida se incluye en dos colas; una específica para la fuente de sincronización asociada, y otra global. Esta estructura proporciona una base eficiente para proporcionar servicios de extracción de unidades de datos en función de parámetros como matasellos o fuente de sincronización asociada.

La clase `MembershipBookkeeping` gestiona una tabla de objetos fuente de sincronización, implementada como una tabla de dispersión, y define un conjunto de métodos para añadir, eliminar y buscar fuentes en la tabla. Así, cuando se produce la recepción de una unidad de datos, además de otras operaciones asociadas a la recepción, la unidad se asocia con el objeto fuente de sincronización correspondiente.

Tanto la cola de paquetes entrantes como los enlaces entre paquetes y fuentes de sincronización se gestionan mediante clases y estructuras base que se pueden manipular en la clase `IncomingDataQueue` y sus clases derivadas, o reutilizar en la implementación de clases de colas de paquetes entrantes alternativas.

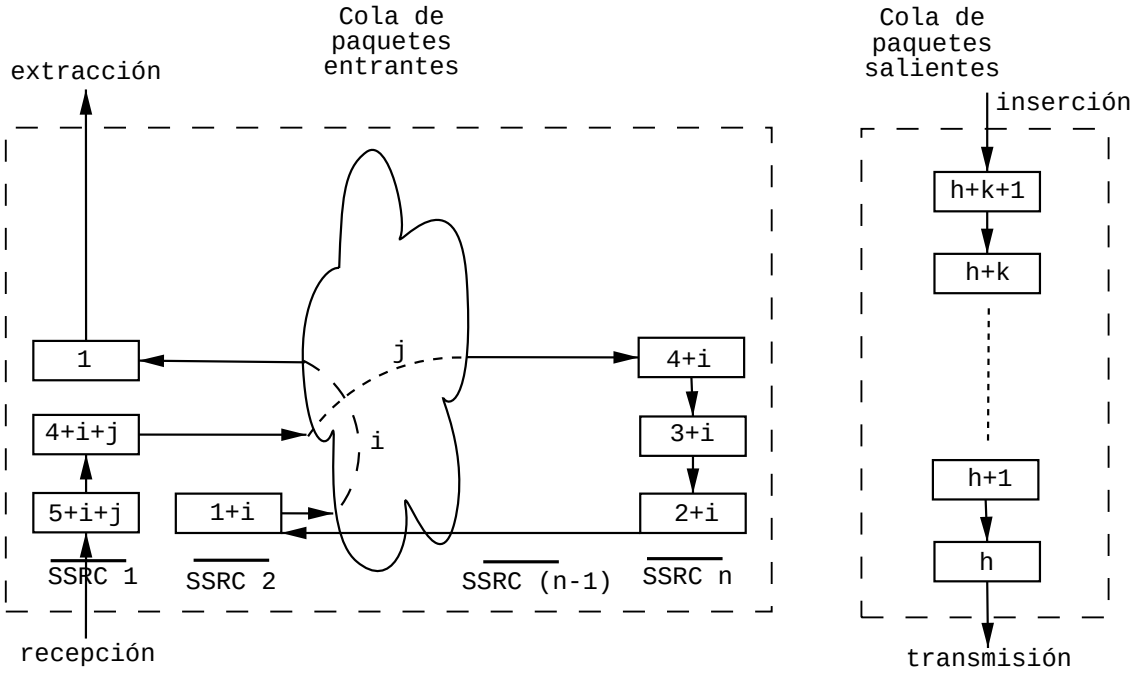


Figura 5.1: Esquema de la lista de paquetes de ccRTP

Entre otras funciones cuya realización fuera de la pila RTP supondría un considerable aumento de complejidad y repetición de código entre diferentes aplicaciones, la cola de paquetes entrantes realiza operaciones para reordenar los paquetes sobre las colas que gestiona. Junto con la reordenación, y atendiendo a parámetros temporales seleccionables mediante métodos públicos, la cola de paquetes entrantes implementa asimismo el mecanismo de amortiguación en el receptor que se describió en la sección 2.2.2. Las estadísticas proporcionadas por las clases que implementan el protocolo RTCP permiten que las aplicaciones realicen los cálculos necesarios para estimar el valor adecuado de los parámetros temporales que se debe fijar para el algoritmo de amortiguación. Esta función se podría añadir asimismo en la propia pila RTP mediante especialización o herencia.

Dado que las dos listas descritas y la tabla de fuentes de sincronización son elementos que influyen decisivamente en el rendimiento de la biblioteca, se ha optado por implementarlas de manera específica mediante estructuras de bajo nivel.

Por otra parte, para proporcionar un modelo completo que se pueda utilizar en aplicaciones basadas en RTP sin requerir ninguna biblioteca adicional, ccRTP utiliza ampliamente los servicios ofrecidos por Common C++, tanto en cuanto a hilos de ejecución, sincronización, temporización y servicios de transporte. Estos servicios se utilizan en los módulos de asociación de hilos de ejecución y de conexión con el protocolo de transporte subyacente. Asimismo, algunas clases de Common C++ que no se utilizan en ccRTP son útiles para la implementación de aplicaciones basadas en RTP. En particular, las clases `Buffer` y `FixedBuffer` facilitan la gestión de los bloques de memoria reservados para las unidades de datos que se han de insertar en la cola de salida.

El módulo de asignación de hilos de ejecución define una clase base particularmente útil para que aplicaciones que pueden llegar a gestionar un elevado número de sesiones RTP no requieran el uso de un número igual de hilos de ejecución. Esta clase es `RTPSessionPool`, que permite asociar un hilo de ejecución a varias sesiones RTP, para las que gestiona las operaciones de transmisión y recepción de forma circular periódicamente y cada vez que se produce un suceso que requiere atención en alguna

de las sesiones (recepción de paquetes, ya sean de datos o de control).

En relación con este aspecto, las clases mostradas en la jerarquía de la figura 4.8 se han implementado de manera que sus métodos se puedan ejecutar desde un número variable de hilos de ejecución mediante la clase `ThreadLock` de Common C++, utilizando sus operaciones de bloqueo de lectura/escritura, aplicando métodos [138] que permiten que las clases sean reutilizables sin degradar sus prestaciones.

Por otra parte, un detalle de implementación de ccRTP de especial impacto en sus prestaciones es que reduce al mínimo el número de copias de bloques de datos, factor que tiene una influencia decisiva [115] en el rendimiento de aplicaciones que utilizan vídeo de alta calidad o gestionan un número elevado de sesiones RTP.

En la sección 2.3.4 se analizó la complejidad adicional que introducían los traductores y mezcladores de nivel RTP. La posible aparición de estos elementos intermedios en las redes RTP implica la obligatoriedad de los algoritmos de detección de colisiones y bucles en todas las aplicaciones basadas en RTP. ccRTP no sólo implementa estos algoritmos, sino que proporciona funciones especialmente útiles para la implementación de traductores y mezcladores de nivel RTP.

Así, un traductor RTP se puede implementar con ccRTP mediante el uso dos objetos de tipo sesión RTP, realizando éstos todas las operaciones de sincronización, introducción de retardos y reordenación de unidades de datos que puede incorporar un traductor RTP.

En cuanto a la posibilidad de *implementar los nuevos perfiles de RTP* descritos en la sección 2.3.3, ccRTP ofrece un modelo fácilmente adaptable. En el caso del perfil para RTP seguro, su implementación se puede realizar, como se sugiere en la definición del perfil [10], como si SRTP fuese una capa adicional situada entre el nivel de transporte y el nivel RTP; es decir, mediante clases específicas definidas en el módulo de conexión con el protocolo de transporte que se utilizarían para instanciar la plantilla `TRTPSessionBase`. Asimismo, es posible aplicar un enfoque más próximo al principio de procesamiento integrado de niveles, incorporando el perfil mediante clases que especialicen la clase `AVPQueue`.

El perfil para sonido y vídeo con realimentación basada en RTCP se puede implementar en ccRTP también mediante herencia, añadiendo los nuevos tipos de paquetes RTCP como especialización de la clase `QueueRTCPManager`, y el mecanismo de retransmisión de paquetes como especialización de la clase `OutgoingDataQueue`.

No obstante, las dos ampliaciones a corto plazo de mayor importancia que se consideran actualmente para ccRTP están directamente relacionadas con la cola de paquetes entrantes: añadir mecanismos de gestión de la tabla de fuentes de sincronización escalables a grupos de gran tamaño, y añadir una versión de la cola de paquetes entrantes específica para sistemas con sólo dos fuentes de sincronización por sesión.

En primer lugar, el mantenimiento de una tabla de fuentes de sincronización en cada sesión RTP –lo cual es necesario para implementar las funciones del protocolo RTCP– supone una limitación de escalabilidad de la biblioteca que puede tener graves consecuencias en sesiones con un número de participantes del orden de cientos de miles o millones, en particular si se pretende utilizar en dispositivos de memoria limitada. Como solución a este problema se ha propuesto un algoritmo de muestreo de las fuentes de sincronización en sesiones RTP [80]. La implementación de este algoritmo y su incorporación a la jerarquía de clases es factible en tanto que es posible definir una clase alternativa a `IncomingDataQueue` que, en lugar de utilizar `MembershipBookkeeping` para el mantenimiento de la tabla de fuentes de sincronización, utilice una nueva clase que encapsule el algoritmo de

muestreo.

Por último, se considera la posibilidad de añadir otra clase alternativa a `IncomingDataQueue`, específica para aquellas aplicaciones que se comunican con una única fuente de sincronización remota. Una implementación específica para este caso particular probablemente proporcione mejoras de prestaciones apreciables, puesto que se simplifica notablemente la gestión de fuentes de sincronización (entre otras modificaciones, la lista de paquetes de entrada queda reducida a una lista simple, no es necesaria la tabla de fuentes de sincronización, y el algoritmo de detección de colisiones de identificadores SSRC queda reducido a una simple comparación).

5.3. GNU oSIP

oSIP está implementada íntegramente en C, si bien, como se ha detallado anteriormente, su diseño es modular y orientado a objetos. Los dos módulos de SIP se corresponden con dos bibliotecas que se construyen de manera independiente: `fsmtl`, para el módulo de gestión de transacciones, y `osip`, para el módulo de análisis y construcción de mensajes. Los métodos y objetos de ambos módulos se definen, respectivamente, en las cabeceras `osip/fsm.h` y `osip/smsg.h`, incluidas por la cabecera global `osip/osip.h`.

La implementación del analizador sintáctico y el constructor de mensajes SIP se ha realizado en C, sin utilizar herramientas específicas para analizadores léxico-sintácticos completos, que, aunque pueden ser útiles para lenguajes o formatos de sintaxis compleja, en el caso de SIP requerirían más recursos y no simplificarían la implementación. De este modo, es posible manipular los mensajes con el número mínimo de operaciones necesarias para analizar los campos de interés según el diseño expuesto.

Las máquinas de estados de las transacciones SIP se definen asimismo, dada su relativa simplicidad, sin utilizar herramientas específicas. Por otra parte, las colas de sucesos de SIP se implementan con colas de C que el módulo de gestión de transacciones asocia a cada transacción.

Aunque oSIP no impone ningún modelo concreto de asignación de hilos de ejecución ni de sincronización, proporciona, como opción de configuración, los elementos necesarios para, sin requerir ninguna otra biblioteca, asociar un hilo de ejecución separado que sirva todas las transacciones, o bien asociar un hilo de ejecución separado para cada una de las transacciones en curso.

oSIP ha sido sometida con éxito a los tests de tortura definidos por el IETF para las implementaciones de SIP [79], y a pruebas con otras implementaciones independientes en los últimos eventos SIPit. No obstante, como futura ampliación, cuyo desarrollo se ha iniciado actualmente, se considera añadir un módulo adicional de comprobación sintáctica más completo, de modo que se proporcione una solución adecuada para aplicaciones que requieran un mayor nivel de seguridad. La implementación de oSIP facilita la incorporación de este módulo como elemento adicional o incluso la sustitución del módulo de análisis sintáctico básico, puesto que el comportamiento del analizador se define mediante una tabla de métodos que asocia a cada campo SIP un método encargado de su análisis.

Finalmente, oSIP carece actualmente de una interfaz que permita gestionar fácilmente los estados de las sesiones SIP. Aunque se argumentó en el apartado de diseño que, en el caso de muchos servidores, esta función es innecesaria, sí existen otros tipos de aplicaciones basadas en SIP para las cuales creemos posible desarrollar modelos reutilizables y suficientemente genéricos para la gestión de los estados de llamadas. Este problema se plantea como trabajo futuro de especial interés.

5.4. FreeSDP

FreeSDP se ha implementado íntegramente en C. Dado que el diseño presentado en el capítulo anterior se puede llevar a la práctica en este lenguaje sin que sea necesario recurrir a estructuras o mecanismos propios de los lenguajes orientados a objetos, se ha optado por utilizar C y reducir las dependencias a una biblioteca estándar de C [41]. De este modo, sin afectar significativamente al diseño de la biblioteca, se maximiza su portabilidad y se minimizan los recursos necesarios.

FreeSDP se distribuye junto con un manual introductorio [108], así como con un exhaustivo manual de referencia. Asimismo, con la distribución fuente se proporciona un conjunto de pequeñas aplicaciones de prueba.

La biblioteca proporciona dos cabeceras: `freesdp/parser.h` y `freesdp/formatter.h`, correspondientes, respectivamente, a los módulos analizador y constructor de descripciones. Ambas cabeceras están incluidas en una cabecera global, `freesdp/freesdp.h`. Del mismo modo, la biblioteca se obtiene enlazando dos bibliotecas separadas, una para el módulo analizador, y otra para el módulo constructor. Ambas dependen de una tercera biblioteca que contiene los elementos comunes a los dos módulos. Así se facilita su empaquetado por separado.

Tanto el analizador como el constructor de descripciones se han implementado mediante los mecanismos habituales de C, en lugar de utilizar herramientas específicas de análisis léxico, que ofrecen pocas ventajas respecto a la opción elegida, aparte de facilitar la integración con herramientas de análisis sintáctico, lo que hemos considerado innecesario para la sintaxis SDP. El resultado es un sistema más compacto y rápido pero no más difícil de mantener.

Los dos módulos de la biblioteca siguen la sintaxis definida por la última revisión de SDP [58, apéndice A], especificada en formato ABNF [33], y basada asimismo en definiciones de los RFC 1630 [13] (sintaxis de los URI), 2732 [63] (sintaxis de los URI específicos para IPv6) y 2822 [125] (sintaxis de las direcciones de correo-e). Además, se tienen en cuenta algunos errores menores encontrados en la sintaxis del borrador actual de SDP.

La implementación del módulo de análisis sintáctico es permisiva; es decir, admite errores comunes en las implementaciones de SDP -como terminadores de línea incorrectos, espacios adicionales, errores en mayúsculas/minúsculas- que no impiden interpretar correctamente y extraer la información contenida en las descripciones. No obstante, los errores graves en las descripciones, como la falta de líneas obligatorias, detienen el proceso de análisis sintáctico, si bien, los objetos resultantes de un proceso de análisis fallido proporcionan cuanta información haya sido posible obtener antes de producirse el error.

Por el contrario, el constructor de descripciones aplica de manera estricta la sintaxis estándar, utilizando además las variaciones que, a tenor de las discusiones en la lista de correo del grupo MMUSIC del IETF, son admitidas por la mayoría de las implementaciones existentes. En particular, utilizando un amplio conjunto de tipos enumerados para representar los diferentes grupos de atributos y valores de atributos, el constructor aplica el uso de mayúsculas/minúsculas más común.

FreeSDP proporciona métodos y tipos específicos para obtener y modificar los elementos que forman parte de los atributos conocidos por la biblioteca, mientras que los atributos no reconocidos por FreeSDP se pueden obtener mediante un método genérico que proporciona el valor de estos atributos en forma de cadena de texto.

Actualmente, FreeSDP soporta de forma específica la ampliación que define modificadores de an-

cho de banda para sesiones RTP [26] y la ampliación que define el atributo *rtcp* [69], ambas relevantes para el uso de RTP de acuerdo con las modificaciones incorporadas en su última revisión [144].

Capítulo 6

Conclusiones y Trabajo Futuro

En este proyecto se ha realizado un estudio de los sistemas de sesiones multimedia en Internet, prestándose especial atención a los problemas actuales y los últimos desarrollos que pretenden darles solución.

Se han desarrollado varios componentes de un sistema software que establece un modelo de programación de aplicaciones de sesión multimedia. El sistema obtenido es libre, modular, portable, escalable y seguro. Asimismo, abarca las funciones de transporte en tiempo real, control y descripción de sesiones así como negociación de capacidades. En su desarrollo se han aplicado principios de diseño específicos de los sistemas de tiempo real y se han seguido técnicas de programación modular, orientada a objetos, programación genérica y programación orientada a aspectos.

En particular, en el campo del transporte en tiempo real se ha desarrollado un sistema, GNU ccRTP, que, además de dar solución a problemas actuales, constituye, gracias a su modularidad, una plataforma de experimentación en las siguientes materias:

- Control de congestión en RTP y sistemas de transporte en tiempo real en general.
- Compensación de las fluctuaciones en el periodo de recepción de unidades de datos.
- Control de errores.

Se ha comentado ya a lo largo de los correspondientes capítulos las ampliaciones y mejoras previstas a corto plazo para cada uno de los componentes del sistema presentado en este proyecto. Cabe destacar por último, como objetivos próximos pero de mayor envergadura, el desarrollo de proyectos ya iniciados, como el teléfono/agente de mensajería instantánea universal del proyecto GNU, así como la mayor integración y despliegue del sistema en plataformas abiertas de convergencia entre redes de conmutación de circuitos y redes de conmutación de paquetes, como GNU Bayonne o Asterisk.

Finalmente, habiendo publicado los resultados de este proyecto como *software libre*, siguiendo un modelo de desarrollo plenamente abierto, esperamos contribuir al desarrollo de implementaciones libres de protocolos abiertos que proporcionen una base sólida para el desarrollo de soluciones libres para sistemas de telecomunicaciones.

GNU Free Documentation License

Version 1.1, March 2000

Preamble

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

6.1. Applicability and Definitions

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

6.2. Verbatim Copying

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

6.3. Copying in Quantity

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

6.4. Modifications

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
- State on the Title page the name of the publisher of the Modified Version, as the publisher.
- Preserve all the copyright notices of the Document.
- Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- Include an unaltered copy of this License.
- Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.
- Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties – for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

6.5. Combining Documents

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6.6. Collections of Documents

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

6.7. Aggregation With Independent Works

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

6.8. Translation

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

6.9. Termination

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

6.10. Future Revisions of This License

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License .ºr any later version.ªpplies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. A copy of the license is included in the section entitled “GNU Free Documentation License”.

If you have no Invariant Sections, write “with no Invariant Sections” instead of saying which ones are invariant. If you have no Front-Cover Texts, write “no Front-Cover Texts” instead of “Front-Cover Texts being LIST”; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Acrónimos

ABNF	Augmented Backus-Naur Form
ADU	Application Data Unit
ANSI	American National Standards Institute
ASN	Abstract Syntax Notation
ATM	Asynchronous Transfer Mode
BNF	Backus-Naur Form
CGI	Common Gateway Interface
DCCP	Datagram Congestion Control Protocol
FTP	File Transfer Protocol
GNU	GNU's Not UNIX
HTTP	HyperText Transfer Protocol
IANA	Internet Assigned Numbers Authority
IETF	Internet Ingeneering Task Force
IP	Internet Protocol
IPv4	Internet Protocol, version 4
IPv6	Internet Protocol, version 6
ITU-T	International Telecommunication Union - Telecommunication Standarization Sector
IVR	Interactive Voice Response
MEGACO	Media Gateway Control Protocol
MIME	Multipurpose Internet Mail Extensions
MMUSIC	Multiparty Multimedia Session Control
NTP	Network Time Protocol
PGP	Pretty Good Privacy
PER	Packet Encoding Rules
Perl	Practical Extraction and Report Language (Pathologically Eclectic Rubbish Lister)
RADIUS	Remote Authentication Dial In User Service
RFC	Request For Comments
RR	Receiver Report
RSVP	Resource ReSerVation Protocol
RTCP	RTP Control Protocol
RTP	Real-Time Streaming Protocol
SAP	Session Announcement Protocol
SCTP	Stream Control Transmission Protocol
SDES	Source DEScription
SDP	Session Description Protocol

SIP	Session Initiation Protocol
SIPit	SIP Interoperability Test
SMIL	Synchronized Multimedia Integration Language
SR	Sender Report
SS7	System Signaling No. 7
SSL	Secure Sockets Layer
SSRC	Synchronization SouRCe
STL	Standart Template Library
TCP	Transport Control Protocol
TRIP	Telephony Routing over IP
UA	User Agent
UAC	User Agent Client
UAS	User Agent Server
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
XDR	eXternal Data Representation
XML	eXtensible Markup Language

Bibliografía

[1] Andrei Alexandrescu. *Modern C++ Design. Generic Programming and Design Patterns Applied*. Addison Wesley Professional, primera edición, 2001.

[2] Mark Allman, Vern Paxson y W. Richard Stevens. *TCP Congestion Control*. RFC 2581, Internet Engineering Task Force, Network Working Group, Abril 1999. Estado: Proposed Standard.

[3] Flemming Andreasen. *SDP Simple Capability Declaration*. Borrador del IETF, Internet Engineering Task Force, Multiparty Multimedia Session Control (mmusic) Working Group, Febrero 2002. Trabajo en desarrollo bajo el nombre draft-ietf-mmusic-sdpng-05.txt.

[4] Mauricio Arango, Andrew Dugan, Isaac Elliott, Christian Huitema y Scott Pickett. *Media Gateway Control Protocol (MGCP) Version 1.0*. RFC 2705, Internet Engineering Task Force, Network Working Group, Octubre 1999. Categoría: Informational.

[5] Audio-Video Transport Working Group. *RTP/RTCP Implementation Report*. Informe técnico, Internet Engineering Steering Group, Septiembre 2002. http://www.iesg.org/IESG/Implementations/RTP-RTCP-Implementation.txt.

[6] Matt Austern. *The standard librarian: Defining iterators and const iterators*. *C/C++ Users Journal*, página 74, Enero 2001. http://www.cuj.com/experts/1901/austern.htm?topic=experts.

[7] Daniel O. Awduche, Angela Chiu, Anwar Elwalid, Indra Widjaja y XiPeng Xiao. *Overview and Principles of Internet Traffic Engineering*. RFC 3272, Internet Engineering Task Force, Network Working Group, Mayo 2002. Categoría: Informational.

[8] Jeff Ayars, Dick Bulterman, Aaron Cohen, Ken Day, Erik Hodge, Philipp Hoschka, Eric Hyche, Muriel Jourdan, Michelle Kim, Kenichi Kubota *et al.* *Synchronized Multimedia Integration Language (SMIL 2.0)*. Informe técnico, World Wide Web Consortium, 2001.

[9] Fred Baker, Abel Weinrib, Bob Braden, Lixia Zhang, Scott Bradner, Allyn Romanow, Michael O'Dell y Allison Mankin. *Resource ReSerVation Protocol (RSVP). Version 1 Applicability Statement. Some Guidelines on Deployment*. RFC 2208, Internet Engineering Task Force, Network Working Group, Septiembre 1997. Categoría: Informational.

[10] Mark Baugher, Rolf Blom, Elisabetta Carrara, David A. McGrew, Mats Naslund, Karl Norrman y David Oran. *The Secure Real-time Transport Protocol*. Borrador del IETF, Internet Engineering Task Force, Network Working Group, Audio Video Transport Working Group, Junio 2002. Trabajo en desarrollo, bajo el nombre draft-ietf-avt-srtp-05.txt. Válido hasta Diciembre de 2002.

[11] Mark Baugher, Bill Strahm y Irina Suconick. *Real-Time Transport Protocol Management Information Base*. RFC 2959, Internet Engineering Task Force, Network Working Group, Octubre 2000. Estado: Proposed Standard.

[12] Emery D. Berger, Kathryn S. McKinley, Robert D. Blumofe y Paul R. Wilson. *Hoard: A scalable memory allocator for multithreaded applications*. En *Architectural Support for Programming Languages and Operating Systems*. Noviembre 2000. Disponible en http://www.hoard.org.

[13] Tim Berners-Lee. *Universal Resource Identifiers in WWW. A Unifying Syntax for the Expression of Names and Addresses of Objects on the Network as used in the World-Wide Web*. RFC 1630, Internet Engineering Task Force, Network Working Group, Junio 1994. Categoría: Informational.

[14] Steven Blake, David L. Black, Mark A. Carlson, Elwyn Davies, Zheng Wang y Walter Weiss. *An Architecture for Differentiated Services*. RFC 2475, Internet Engineering Task Force, Network Working Group, Diciembre 1998. Categoría: Informational.

[15] Bob Braden and David D. Clark and Jon Crowcroft and Bruce Davie and Steve Deering and Deborah Estrin and Sally Floyd and Van Jacobson and Greg Minshall and Craig Partridge and Larry Peterson and K. K. Ramakrishnan and Scott Shenker and John Wroclawski and Lixia Zhang. *Recommendations on Queue Management and Congestion Avoidance in the Internet*. RFC 2309, Internet Engineering Task Force, Network Working Group, Abril 1998. Categoría: Informational.

[16] Bob Braden. *T/TCP - TCP for Transactions - Concepts*. RFC 1379, Internet Engineering Task Force, Network Working Group, Noviembre 1992. Categoría: Informational.

[17] Bob Braden. *T/TCP - TCP Extensions for Transactions Functional Specification*. RFC 1644, Internet Engineering Task Force, Network Working Group, Julio 1994. Categoría: Experimental.

[18] Bob Braden, Lixia Zhang, Steve Berson, Shai Herzog y Sugih Jamin. *Resource ReSerVation Protocol (RSVP). Version 1 Functional Specification*. RFC 2206, Internet Engineering Task Force, Network Working Group, Septiembre 1997. Estado: Proposed Standard.

[19] Torsten Braun y Christophe Diot. *Protocol Implementation Using Integrated Layer Processing*. En *ACM SIGCOMM'95*, páginas 151–161. Mayo 1995. http://www.iam.unibe.ch/ braun/lit/.

[20] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen y Eve Maler. *Extensible Markup Language (XML) 1.0 (Second Edition)*. Informe técnico, World Wide Web Consortium, 2000.

[21] Pat R. Calhoun, John Loughney, Jari Arkko, Erik Guttman y Glen Zorn. *Diameter Base Protocol*. Borrador del IETF, Internet Engineering Task Force, AAA Working Group, Julio 2002. Trabajo en desarrollo, bajo el nombre draft-ietf-aaa-diameter-12.txt. Válido hasta Octubre de 2002.

[22] B. Callaghan, B. Pawlowski y P. Staubach. *NFS: Network File System Protocol Specification*. RFC 1813, Internet Engineering Task Force, Network Working Group, Junio 1995. Categoría: Informational. Obsoleto en favor de [156].

[23] Gonzalo Camarillo, Jan Holler, Goran AP Eriksson y Henning Schulzrinne. *Grouping of media lines in SDP*. Borrador del IETF, Internet Engineering Task Force, Multiparty Multimedia Session Control (mmusic) Working Group, Febrero 2002. Trabajo en desarrollo bajo el nombre draft-ietf-mmusic-fid-06.txt. Válido hasta Agosto de 2002.

[24] Philip Carden. *Building Voice over IP*. *Network Computing*, 2000.

[25] Carl Rigney and Allan C. Rubens and William Allen Simpson and Steve Willens. *Remote Authentication Dial In User Service (RADIUS)*. RFC 2865, Internet Engineering Task Force, Network Working Group, Junio 2000. Estado: Draft Standard.

[26] Stephen L. Casner. *SDP Bandwidth Modifiers for RTCP Bandwidth*. Borrador del IETF, Internet Engineering Task Force, Network Working Group, Audio Video Transport Working Group, Noviembre 2001. Trabajo en desarrollo, bajo el nombre draft-ietf-avt-rtcp-bw-05.txt. Válido hasta Mayo de 2002.

[27] Stephen L. Casner y Marshall Eubanks. *Re: [AVT] RTCP Question*. http://www.ietf.org/mail-archive/working-groups/avt/current/msg00323.html, sep 2001. Conversación mantenida a través de la lista de distribución avt@ietf.org, del grupo de trabajo AVT del IETF.

[28] Stephen L. Casner y Philipp Hoschka. *MIME Type Registration of RTP Payload Formats*. Borrador del IETF, Internet Engineering Task Force, Network Working Group, Audio Video Transport Working Group, Noviembre 2001. Trabajo en desarrollo, bajo el nombre draft-ietf-avt-rtp-mime-06.txt. Válido hasta Mayo de 2002.

[29] Robert J. Chassell y Richard M. Stallman. *GNU Texinfo 4.0b*. Free Software Foundation, 59 - Temple Place - Suite 330. Boston, MA 02111-1307, USA, 4.0b edición, Mayo 2001. Disponible en http://www.gnu.org/software/texinfo.

[30] D. G. Clark y D. L. Tennenhouse. *Architectural Considerations for a New Generation of Protocols*. En *ACM SIGCOMM Computer Communication Review*, volumen 20, páginas 200–208. Telemedia, Networks and Systems Group. Massachusetts Institute of Technology, Septiembre 1990. http://tns-www.lcs.mit.edu/publications/acm90/paper.html.

[31] Lode Coene. *Stream Control Transmission Protocol Applicability Statement*. RFC 2960, Internet Engineering Task Force, Network Working Group, Abril 2002. Categoría: Informational.

[32] Robin Cover. *The XML Cover Pages. Extensible Markup Language (XML)*. http://www.oasis-open.org/cover/xml.html, Febrero 2002. Organization for the Advancement of Structured Information Standards.

[33] David H. Crocker y Paul Overell. *Augmented BNF for Syntax Specifications: ABNF*. RFC 2234, Internet Engineering Task Force, Network Working Group, Noviembre 1997. Estado: Proposed Standard.

[34] Bill Croft y John Gilmore. *Bootstrap Protocol (BOOTP)*. RFC 951, Internet Engineering Task Force, Network Working Group, Septiembre 1985. Estado: Draft Standard.

[35] Jon Crowcroft, Mark Handley y Ian Wakeman. *Internetworking Multimedia*. Series in Networking. Morgan Kaufmann Publishers, Septiembre 1999. Se puede consultar en http://www.cs.ucl.ac.uk/staff/jon/mmbook/book/book.html.

[36] Fernando Cuervo, Nancy Greene, Christian Huitema, Abdallah Rayhan, Brian Rosen y John Segers. *Megaco Protocol Version 1.0*. RFC 3015, Internet Engineering Task Force, Network Working Group, Noviembre 2000. Estado: Proposed Standard.

[37] Ismail Dalgic y Hanlin Fang. *Comparison of H.323 and SIP for IP Telephony Signaling*. En *Proceedings of Photonics East*. Septiembre 1999. http://www.cs.columbia.edu/ hgs/papers/others/Dalg9909_Comparision.pdf.

[38] Peter L. Deutsch y Jean-Loup Gailly. *ZLIB Compressed Data Format Specification version 3.3*. RFC 1832, Internet Engineering Task Force, Network Working Group, Mayo 1996. Estado: Draft Standard.

[39] Peter L. Deutsch, Rickard Schoultz, Patrik Faltstrom y Chris Weider. *Architecture of the WHOIS++ service*. RFC 1835, Internet Engineering Task Force, Network Working Group, Agosto 1995. Estado: Proposed Standard.

[40] Steve Donovan. *The SIP INFO Method*. RFC 2976, Internet Engineering Task Force, Network Working Group, Octubre 2000. Estado: Proposed Standard.

[41] Ulrich Drepper, Andreas Jaeger, Paul Eggert *et al.* *The GNU C Library Reference Manual*. Free Software Foundation, 59 - Temple Place - Suite 330. Boston, MA 02111-1307, USA, 2.2.5 edición, Junio 2002. Disponible en http://www.gnu.org/manual/.

[42] Ralph Droms. *Dynamic Host Configuration Protocol*. RFC 2131, Internet Engineering Task Force, Network Working Group, Marzo 1997. Estado: Draft Standard.

[43] Margaret A. Ellis y Bjarne Stroustrup. *The Annotated C++ Reference Manual*. Professional Computing Series. AT & T Bell Telephone Laboratories, Incorporated. Addison-Wesley Publishing Company, segunda edición, Octubre 1992.

[44] Gerard Fernando, Vivek Goyal, Don Hoffman y M. Reha Civanlar. *RTP Payload Format for MPEG1/MPEG2 Video*. RFC 2250, Internet Engineering Task Force, Network Working Group, Audio-Video Transport Working Group, Enero 1998. Estado: Proposed Standard.

[45] Roy T. Fielding, James Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen, Larry Masinter y Paul J. Leach Tim Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2616, Internet Engineering Task Force, Network Working Group, Junio 1999. Estado: Proposed Standard.

[46] Sally Floyd y Kevin Fall. *Promoting the Use of End-to-End Congestion Control in the Internet*. En *IEEE/ACM Transactions on Networking*, volumen 7, páginas 458–472. Agosto 1999. http://www.icir.org/floyd/end2end-paper.html.

[47] Sally Floyd, Mark Handley, Jitendra Padhye y Jörg Widmer. *Equation-Based Congestion Control for Unicast Applications*. En *ACM SIGCOMM Conference on Applications, Technologies, Architectures and Protocols for Computer Communication*, páginas 43–56. Estocolmo, Suecia, Agosto 2000. http://www.acm.org/pubs/citations/proceedings/comm/347337/p43-floyd/.

[48] Ned Freed y Nathaniel S. Borenstein. *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*. RFC 2045, Internet Engineering Task Force, Network Working Group, Noviembre 1996. Estado: Draft Standard.

[49] Ned Freed y Nathaniel S. Borenstein. *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*. RFC 2046, Internet Engineering Task Force, Network Working Group, Noviembre 1996. Estado: Draft Standard.

[50] Ned Freed, John Klensin y Jon Postel. *Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures*. RFC 2048, Internet Engineering Task Force, Network Working Group, Noviembre 1996. Categoría: Best Current Practice.

[51] Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides. *Design Patterns. Elements of Reusable Object-Oriented Software*. Professional Computing Series. Addison Wesley, segunda edición, 1994.

[52] Joan Gargano y Ken Weiss. *Whois and Network Information Lookup Service. Whois++*. RFC 1834, Internet Engineering Task Force, Network Working Group, Agosto 1995. Categoría: Informational.

[53] Nancy Greene, Michael A. Ramalho y Brian Rosen. *Media Gateway Control Protocol Architecture and Requirements*. RFC 2805, Internet Engineering Task Force, Network Working Group, Abril 2000. Estado: Proposed Standard.

[54] Arnt Gulbrandsen, Paul Vixie y Levon Esibov. *A DNS RR for specifying the location of services (DNS SRV)*. RFC 2782, Internet Engineering Task Force, Network Working Group, Febrero 2000. Estado: Proposed Standard.

[55] Fred Halsall. *Data Communications, Computer Networks and Open Systems*. Electonic Systems Engineering Series. Addison Wesley, cuarta edición, 1996.

[56] Mark Handley, Jon Crowcroft, Carster Bormann y Jörg Ott. *Very Large Conferences on the Internet: the Internet Multimedia Conferencing Architecture*. *Computer Networks (Amsterdam, Netherlands))*, 31, n. 3:páginas 191–204, Diciembre 1999. http://www.aciri.org/mjh/cnis.ps.gz.

[57] Mark Handley y Van Jacobson. *SDP: Session Description Protocol*. RFC 2327, Internet Engineering Task Force, Network Working Group, Abril 1998. Estado: Proposed Standard.

[58] Mark Handley, Van Jacobson y Colin Perkins. *SDP: Session Description Protocol*. Borrador del IETF, Internet Engineering Task Force, Network Working Group, Mayo 2002. Trabajo en desarrollo bajo el nombre draft-ietf-mmusic-sdp-new-10.txt. Válido hasta Noviembre de 2002.

[59] Mark Handley, Colin Perkins y Edmund Whelan. *Session Announcement Protocol*. RFC 2974, Internet Engineering Task Force, Network Working Group, Octubre 2000. Categoría: Experimental.

[60] Mark Handley, Henning Schulzrinne, Eve Schooler y Jonathan Rosenberg. *SIP: Session Initiation Protocol*. RFC 2543, Internet Engineering Task Force, Network Working Group, Marzo 1999. Estado: Proposed Standard.

[61] Mark Handley, David Thaler y Roger Kermode. *Multicast-Scope Zone Announcement Protocol (MZAP)*. RFC 2776, Internet Engineering Task Force, Network Working Group, Febrero 2000. Estado: Proposed Standard.

[62] Dimitry Van Heesch. *Doxygen Manual*. dimitry@stack.nl, version 1.2.16 edición, Mayo 2002. Se puede obtener, en diversos formatos, junto con la distribución fuente de Doxygen en http://www.doxygen.org.

[63] Robert M. Hinden, Brian E. Carpenter y Larry Masinter. *Format for Literal IPv6 Addresses in URL's*. RFC 2732, Internet Engineering Task Force, Network Working Group, Diciembre 1999. Estado: Proposed Standard.

[64] Paul Hoffman y Scott Bradner. *Defining the IETF*. RFC 3233, Internet Engineering Task Force, Network Working Group, Febrero 2002. Categoría: Best Current Practice.

[65] Paul E. Hoffman, Larry Masinter y Jamie Zawinski. *The mailto URL scheme*. RFC 2368, Internet Engineering Task Force, Network Working Group, Julio 1998. Estado: Proposed Standard.

[66] Gerard J. Holzmann. *Design and Validation of Computer Protocols*. Software Series. Prentice Hall, Lucent Technologies, Bell Laboratories, Incorporated. Murray Hill, New Jersey 07974., Octubre 1991. http://cm.bell-labs.com/cm/cs/who/gerard/popd.html.

[67] Philipp Hoschka. *Integrating SDP Functionality Into SMIL*. Rfc, Internet Engineering Task Force, Agosto 1998. Borrador del IETF bajo el nombre draft-hoschka-smilsdp-01.txt. Valido hasta el 1 de Febrero de 1999. http://www.w3.org/AudioVideo/1998/08/draft-hoschka-smilsdp-01.

[68] Russell Housley. *Cryptographic Message Syntax*. RFC 2630, Internet Engineering Task Force, Network Working Group, Junio 1999. Estado: Proposed Standard.

[69] Christian Huitema. *RTCP attribute in SDP*. Borrador del IETF, Internet Engineering Task Force, Network Working Group, Febrero 2002. Trabajo en desarrollo bajo el nombre draft-ietf-mmusic-comedia-02.txt. Válido hasta Agosto de 2002.

[70] John W.Ñoerenberg II, Jon Callas, Lutz Donnerhacke, Hal Finney y Rodney Thayer. *OpenPGP Message Format*. RFC 2440, Internet Engineering Task Force, Network Working Group, Noviembre 1998. Estado: Proposed Standard.

[71] Internet Assigned Numbers Authority. *RTP Parameters*. http://www.iana.org/assignments/rtp-parameters, Octubre 2001.

[72] Internet Assigned Numbers Authority. *Session Initiation Protocol (SIP) Parameters, SIP Event Types Namespaces y SIP Table of Mappings from service field values to transport protocols*. http://www.iana.org/assignments/sip-parameters, Octubre 2001.

[73] Internet Assigned Numbers Authority. *Session Description Protocol (SDP) Parameters*. http://www.iana.org/assignments/sdp-parameters, Enero 2002.

[74] Internet Engineering Task Force, Transport Area. *Multiparty Multimedia Session Control (mmusic) Charter*. http://www.ietf.org/html.charters/mmusic-charter.html, Septiembre 2002.

[75] Van Jacobson. *Compressing TCP/IP Headers for Low-Speed Serial Links*. RFC 1144, Internet Engineering Task Force, Network Working Group, Febrero 1990. Estado: Proposed Standard.

[76] Wenyu Jiang, Jonathan Lennox, Henning Schulzrinne y Kundan Singh. *Towards Junking the PBX: Deploying IP Telephony*. En *NOSSDAV 2001*. Network and Operating System Support for Digital Audio and Video, Junio 2001. http://www.cs.columbia.edu/ hgs/papers/Jian0106_Junking.pdf.

[77] Alan Johnston, Steve Donovan, Robert Sparks, Chris Cunningham y Kevin Summers. *Session Initiation Protocol Basic Call Flow Examples*. Borrador del IETF, Internet Engineering Task Force, SIPPING Working Group, Agosto 2002. Trabajo en desarrollo, bajo el nombre draft-ietf-sipping-basic-call-flows. Válido hasta Febrero de 2003.

[78] Alan Johnston, Steve Donovan, Robert Sparks, Chris Cunningham y Kevin Summers. *Session Initiation Protocol PSTN Call Flows*. Borrador del IETF, Internet Engineering Task Force, SIPPING Working Group, Agosto 2002. Trabajo en desarrollo, bajo el nombre draft-ietf-sipping-pstn-call-flows. Válido hasta Febrero de 2003.

[79] Alan Johnston, Jonathan Rosenberg y Henning Schulzrinne and. *Session Initiation Protocol PSTN Call Flows*. Borrador del IETF, Internet Engineering Task Force, SIPPING Working Group, Agosto 2002. Trabajo en desarrollo, bajo el nombre draft-ietf-sipping-torture-tests-00.txt. Válido hasta Febrero de 2003.

[80] Jonathan Rosenberg and Henning Schulzrinne. *Sampling of the Group Membership in RTP*. RFC 2762, Internet Engineering Task Force, Network Working Group, Febrero 2000. Categoría: Experimental.

[81] Jonathan Rosenberg and Hussein F. Salama and Matt Squire. *Telephony Routing over IP (TRIP)*. RFC 3219, Internet Engineering Task Force, Network Working Group, Enero 2002. Estado: Proposed Standard.

[82] Jori Liesenborgs. *JRTPLIB*. http://lumumba.luc.ac.be/jori/jrtplib/jrtplib.html, Septiembre 2002. Limburgs Universitair Centrum.

[83] Matthias Jung y Ernst W. Biersack. *How layering protocol software violates separation of concerns*. En *ECOOP Workshop on Aspects and Dimensions of Concerns*. jun 2000. http://trese.cs.utwente.nl/Workshops/adc2000/papers/Jung.pdf.

[84] Brian Kantor y Phil Lapsley. *Network News Transfer Protocol. A Proposed Standard for the Stream-Based Transmission of News*. RFC 977, Internet Engineering Task Force, Network Working Group, Febrero 1986. Estado: Proposed Standard.

[85] Stephen Kent y Randall Atkinson. *Security Architecture for the Internet Protocol*. RFC 2401, Internet Engineering Task Force, Network Working Group, Noviembre 1998. Estado: Proposed Standard.

[86] Yoshihiro Kikuchi, Yoshinori Matsui, Toshiyuki Nomura, Shigeru Fukunaga y Hideaki Kimata. *RTP Payload Format for MPEG-4 Audio/Visual Streams*. RFC 3016, Internet Engineering Task Force, Network Working Group, Noviembre 2000. Estado: Proposed Standard.

[87] Katsushi Kobayashi, Akimichi Ogawa, Stephen L. Casner y Carsten Bormann. *RTP Payload Format for 12-bit DAT Audio and 20- and 24-bit Linear Sampled Audio*. RFC 3190, Internet Engineering Task Force, Network Working Group, Audio-Video Transport Working Group, Enero 2002. Estado: Proposed Standard.

[88] Klaus Kreft y Angelica Langer. *Iterators in the Standard C++ Library*. *C++ Report*, Agosto 1999. http://home.camelot.de/langer/Articles/IteratorsIStdlib/cppr9612_kreft.html.

[89] Rajesh Kumar y Mohamed Mostafa. *Conventions for the use of the Session Description Protocol (SDP) for ATM Bearer Connections*. RFC 3108, Internet Engineering Task Force, Network Working Group, Mayo 2001. Estado: Proposed Standard.

[90] Dirk Kutscher, Joerg Ott y Carsten Bormann. *Session Description and Capability Negotiation*. Borrador del IETF, Internet Engineering Task Force, Multiparty Multimedia Session Control (mmusic) Working Group, Julio 2002. Trabajo en desarrollo bajo el nombre draft-ietf-mmusic-sdpng-05.txt. Válido hasta Diciembre de 2002.

[91] Lars-Ake Larzon. *A Lighter UDP*. Master's thesis, Division of Computer Communications. Departmento of Computer Science and Electrical Engineering. Lulea University of Technology, Diciembre 1999.

[92] Lars-Ake Larzon, Mikael Degermark y Stephen Pink. *The UDP Lite Protocol*. Borrador del IETF, Internet Engineering Task Force, Transport Area Working Group, Enero 2002. Trabajo en desarrollo, bajo el nombre draft-ietf-tsvwg-udp-lite-00.txt. Válido hasta Julio de 2002.

[93] Jonathan Lennox y Henning Schulzrinne. *Call Processing Language Framework and Requirements*. Informe técnico, Internet Engineering Task Force, Network Working Group, IP Telephony Working Group, 2000.

[94] Çaglan M. Aras, James F. Kurose, Douglas S. Reeves y Henning Schulzrinne. *Real-Time Communications in Packet-Switched Networks*. En *IEEE Proceedings*, 82, páginas 122–139. Enero 1994. Disponible en http://www.cs.columbia.edu/ hgs/papers/others/Aras9401_Real.ps.gz.

[95] David J. MacKenzie, Akim Demaille *et al.* *GNU Autoconf 2.53*. Free Software Foundation, 59 - Temple Place - Suite 330. Boston, MA 02111-1307, USA, 2.53 edición, Marzo 2002. Disponible en http://www.gnu.org/software/autoconf/.

[96] David J. MacKenzie, Tom Tromey, Akim Demaille *et al.* *GNU Automake 1.6.3*. Free Software Foundation, 59 - Temple Place - Suite 330. Boston, MA 02111-1307, USA, 1.6.3 edición, Julio 2002. Disponible en http://www.gnu.org/software/automake.

[97] Rohan Mahy, Ben Campbell, Alan Johnston, Daniel G. Petrie, Jonathan Rosenberg y Robert J. Sparks. *A Multi-party Application Framework for SIP*. Borrador del IETF, Internet Engineering Task Force, SIPPING Working Group, Junio 2002. Trabajo en desarrollo, bajo el nombre draft-ietf-sipping-cc-framework. Válido hasta Diciembre de 2002.

[98] Allison Mankin, Allyn Romanow, Scott Bradner y Vern Paxson. *IETF Criteria for Evaluating Reliable Multicast Transport and Application Protocols*. RFC 2357, Internet Engineering Task Force, Network Working Group, Junio 1998. Categoría: Informational.

[99] Robert C. Martin. *Advanced Principles, Patterns, and Process of Object Oriented Software Development*. Prentice Hall, primera edición, Septiembre 2001.

[100] Robert C. Martin. *Engineering Notebook: Template Method & Strategy – Inheritance vs. Delegation*. *C/C++ Users Journal*, Agosto 2001. http://www.cuj.com/experts/1908/martin.htm?topic=experts.

[101] Scott D. Meyers. *Effective C++, 50 Specific Ways to Improve Your Programs and Designs*. Professional Computing Series. Brian W. Kernigham. Addison-Wesley, segunda edición, 1998.

[102] Scott D. Meyers. *More Effective C++, 35 New Ways to Improve Your Programs and Designs*. Professional Computing Series. Brian W. Kernigham. Addison-Wesley, segunda edición, 1998.

[103] David L. Millis. *Network Time Protocol (version 3) Specification and Implementation*. RFC 1315, Internet Engineering Task Force, Network Working Group, Marzo 1992. Estado: Draft Standard.

[104] Paul Mockapetris. *Domain Names - Concepts and Facilities*. RFC 1034, Internet Engineering Task Force, Network Working Group, Noviembre 1987. Estado: Standard.

[105] Paul Mockapetris. *Domain Names - Implementation and Specification*. RFC 1035, Internet Engineering Task Force, Network Working Group, Noviembre 1987. Estado: Standard.

[106] Jack Moffitt. *RTP Payload Format for Vorbis Encoded Audio*. Borrador del IETF, Internet Engineering Task Force, Network Working Group, Febrero 2001. Trabajo en desarrollo, bajo el nombre draft-moffitt-vorbis-rtp-00.txt. Disponible en http://www.xiph.org/ogg/vorbis/doc/draft-moffitt-vorbis-rtp-00.txt.

[107] Aymeric Moizard *et al*. *The GNU oSIP Library*. http://www.gnu.org/software/osip/, Septiembre 2002.

[108] Federico Montesino Pouzols. *FreeSDP Manual. A Free SDP Parser and Formatter*, 0.1 edición, Septiembre 2002. Distribuido junto con FreeSDP. http://savannah.gnu.org/projects/freesdp.

[109] Keith Moore. *MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text*. RFC 2046, Internet Engineering Task Force, Network Working Group, Noviembre 1996. Estado: Draft Standard.

[110] Alexandre Oliva, Thomas Tanner, Gary V. Vaughan, Ossama Othman, Robert Boehne y Gordon Matzigkeit. *GNU Libtool*. Free Software Foundation, 59 - Temple Place - Suite 330. Boston, MA 02111-1307, USA, 1.4.2 edición, Febrero 2002. Disponible en http://www.gnu.org/software/libtool/.

[111] Sean Olson, Gonzalo Camarillo y Adam Roach. *Support for IPv6 in SDP*. Borrador del IETF, Internet Engineering Task Force, Network Working Group, Febrero 2002. Trabajo en desarrollo bajo el nombre draft-ietf-mmusic-sdp-ipv6-03.txt. Válido hasta Agosto de 2002.

[112] Open Mash Consortium. University of California. *Open Mash*. http://www.openmash.org, Septiembre 2002.

[113] Joerg Ott y Colin Perkins. *SDPng Transition*. Borrador del IETF, Internet Engineering Task Force, Multiparty Multimedia Session Control (mmusic) Working Group, Julio 2002. Trabajo en desarrollo bajo el nombre draft-ietf-mmusic-sdpng-trans-01.txt. Válido hasta Enero de 2003.

[114] Jörg Ott, Stephan Wenger, Shigeru Fukunaga, Noriyuki Sato, Koichi Yano, Akihiro Miyazaki, Koichi Hata, Rolf Hakenberg y Carsten Burmeister. *Extended RTP Profile for RTCP-based Feedback (RTP/AVPF)*. Borrador del IETF, Internet Engineering Task Force, Network Working Group, Audio Video Transport Working Group, Junio 2002. Trabajo en desarrollo, bajo el nombre draft-ietf-avt-rtcp-feedback-03.txt. Válido hasta Diciembre de 2002.

[115] Colin Perkins, Ladan Gharai, Tom Lehman y Allison Mankin. *Experiments with Delivery of HDTV over IP Networks*. En *12th International Packet Video Workshop*. Abril 2002. http://www.east.isi.edu/ ladan/.

[116] Colin Perkins, Orion Hodson y Vicky Hardman. *A Survey of Packet Loss Recovery Techniques for Streaming Audio*. *IEEE Network Magazine*, 1998.

[117] Scott Petrack y Lawrence Conroy. *The PINT Service Protocol: Extensions to SIP and SDP for IP Access to Telephone Call Services*. RFC 2848, Internet Engineering Task Force, Network Working Group, Junio 2000. Estado: Proposed Standard.

[118] Ken Pizzini *et al.* *GNU sed Manual*. Free Software Foundation, 59 - Temple Place - Suite 330. Boston, MA 02111-1307, USA, 3.79 edición, Abril 2000. Disponible en http://www.gnu.org/software/sed/.

[119] Jon Postel. *Internet Protocol*. RFC 791, Information Sciences Institute. University of Southern California, Septiembre 1981. Estado: Standard.

[120] Jon Postel. *Transmission Control Protocol*. RFC 793, Information Sciences Institute. University of Southern California, Septiembre 1981. Estado: Standard.

[121] Jon Postel y Joyce Reynolds. *File Transfer Protocol (FTP)*. RFC 959, Internet Engineering Task Force, Network Working Group, Octubre 1985. Estado: Standard.

[122] Jon Postel y Joyce Reynolds. *Simple Mail Transfer Protocol*. RFC 2821, Internet Engineering Task Force, Network Working Group, Abril 2001. Estado: Proposed Standard.

[123] Federico Montesino Pouzols. *GNU ccRTP Manual*, 1.0pre0 edición, Septiembre 2002. Distribuido junto con GNU ccRTP. http://www.gnu.org/software/ccrtp/.

[124] Protocol Engineering Laboratory. University of Delaware. *Innovative Transport Layer Protocols*. http://www.eecis.udel.edu/ amer/PEL/poc/index.html, Septiembre 2002. Contiene documentación y enlaces sobre SCTP y protocolos de transporte en general.

[125] Peter W. Resnick *et al.* *Internet Message Format*. RFC 2822, Internet Engineering Task Force, Network Working Group, Abril 2001. Estado: Proposed Standard.

[126] Adam Roach. *Session Initiation Protocol (SIP)-Specific Event Notification*. RFC 3265, Internet Engineering Task Force, Network Working Group, Junio 2002. Estado: Proposed Standard.

[127] Jonathan Rosenberg, Jonathan Lennox y Henning Schulzrinne. *Programming Internet Telephony Services*. Computer Science Technical Report CUCS-010-99, Columbia University, Marzo 1999. http://www.cs.columbia.edu/ library/TR-repository/reports/reports-1999/cucs-010-99.pdf.

[128] Jonathan Rosenberg y Henning Schulzrinne. *Timer Reconsideration for Enhanced RTP Scalability*. En *INFOCOM*, páginas 233–241. 1998. Disponible en http://www.cs.columbia.edu/ hgs/papers/Rose9803_Timer.ps.gz.

[129] Jonathan Rosenberg y Henning Schulzrinne. *An Offer/Answer Model with the Session Description Protocol (SDP)*. RFC 3264, Internet Engineering Task Force, Network Working Group, Junio 2002. Estado: Proposed Standard.

[130] Jonathan Rosenberg y Henning Schulzrinne. *Reliability of Provisional Responses in the Session Initiation Protocol (SIP)*. RFC 3262, Internet Engineering Task Force, Network Working Group, Junio 2002. Estado: Proposed Standard.

[131] Jonathan Rosenberg y Henning Schulzrinne. *Session Initiation Protocol (SIP): Locating SIP Servers*. RFC 3263, Internet Engineering Task Force, Network Working Group, Junio 2002. Estado: Proposed Standard.

[132] Jonathan Rosenberg, Henning Schulzrinne y Gonzalo Camarillo. *The Stream Control Transmission Protocol as a Transport for the Session Initiation Protocol*. Borrador del IETF, Internet Engineering Task Force, Network Working Group, SIP – Session Initiation Protocol Working Group, Junio 2002. Trabajo en desarrollo, bajo el nombre draft-ietf-sip-sctp-03. Válido hasta Diciembre de 2002.

[133] Jonathan Rosenberg, Henning Schulzrinne, Gonzalo Camarillo, Alan Johnston, Jon Peterson, Robert Sparks, Mark Handley y Eve Schooler. *SIP: Session Initiation Protocol*. RFC 3261, Internet Engineering Task Force, Network Working Group, Junio 2002. Estado: Proposed Standard.

[134] Jonathan Rosenberg y Richard Shockey. *The Session Initiation Protocol (SIP): A Key Component IP Telephony Signaling*. *Computer Telephony*, 2000.

[135] Ross Finlayson. *LIVE.COM Streaming Media*. http://www.live.com/liveMedia/, Septiembre 2002.

[136] Daniel Rubinstein, Jonathan Lennox, Jonathan Rosenberg y Henning Schulzrinne. *Bell Labs/Columbia/UMass RTP Library Internal Function Descriptions* . Informe técnico, University of Massachusetts at Amherst, Department of Computer Science, 1999.

[137] Sally Floyd. *Congestion Control Principles*. RFC 2914, Internet Engineering Task Force, Network Working Group, Septiembre 2000. Categoría: Best Current Practice.

[138] Douglas C. Schmidt. *Strategized Locking, Thread-safe Decorator, and Scoped Locking: Patterns and Idioms for Simplifying Multi-threaded C++ Components*. *C++ Report*, 11, n. 9, Septiembre 1999. http://www.cs.wustl.edu/ schmidt/PDF/locking-patterns.pdf.

[139] Henning Schulzrinne. *A Comprehensive Multimedia Control Architecture for the Internet*. En *NOSSDAV 1997*. Network and Operating System Support for Digital Audio and Video, Mayo 1997. http://www.cs.columbia.edu/ hgs/papers/Schu9705_Comprehensive.ps.gz.

[140] Henning Schulzrinne. *Some Frequently Asked Questions about RTP*. http://www.cs.columbia.edu/ hgs/rtp/faq.html, Febrero 2002.

[141] Henning Schulzrinne y Stephen L. Casner. *RTP Profile for Audio and Video Conferences with Minimal Control*. RFC 1890, Internet Engineering Task Force, Network Working Group, Audio-Video Transport Working Group, Enero 1996. Estado: Proposed Standard.

[142] Henning Schulzrinne y Stephen L. Casner. *RTP Profile for Audio and Video Conferences with Minimal Control*. Borrador del IETF, Internet Engineering Task Force, Network Working Group, Audio Video Transport Working Group, Noviembre 2001. Trabajo en desarrollo, bajo el nombre draft-ietf-avt-profile-new-12.txt. Válido hasta Mayo de 2002.

[143] Henning Schulzrinne, Stephen L. Casner, Ron Frederick y Van Jacobson. *RTP: A Transport Protocol for Real-Time Applicacions*. RFC 1889, Internet Engineering Task Force, Network Working Group, Audio-Video Transport Working Group, Enero 1996. Estado: Proposed Standard.

[144] Henning Schulzrinne, Stephen L. Casner, Ron Frederick y Van Jacobson. *RTP: A Transport Protocol for Real-Time Applications*. Borrador del IETF, Internet Engineering Task Force, Network Working Group, Audio Video Transport Working Group, Noviembre 2001. Trabajo en desarrollo, bajo el nombre draft-ietf-avt-rtp-new-11.txt. Válido hasta Mayo de 2002.

[145] Henning Schulzrinne, Ping Pan, Dorgham Sisalem, Steve Casner *et al.* *Rtp tools 1.17*. Internet Real-Time Laboratory. Columbia University, Abril 2001. http://cs.columbia.edu/IRT/software/rtptools/.

[146] Henning Schulzrinne y Scott Petrack. *RTP Payload for DTMF Digits, Telephony Tones and Telephony Signals*. Borrador del IETF, Internet Engineering Task Force, Network Working Group, Audio Video Transport Working Group, Mayo 2002. Trabajo en desarrollo, bajo el nombre draft-ietf-avt-rfc2833bis-00.txt. Válido hasta Diciembre de 2002.

[147] Henning Schulzrinne, Anup Rao y Robert Lanphier. *Real Time Streaming Protocol (RTSP)*. RFC 2326, Internet Engineering Task Force, Network Working Group, Abril 1998. Estado: Proposed Standard.

[148] Henning Schulzrinne y Jonathan Rosenberg. *A Comparison of SIP and H.323 for Internet Telephony*. En *NOSSDAV 1998*. Network and Operating System Support for Digital Audio and Video, 1998. http://www.cs.columbia.edu/ hgs/papers/Schu9807_Comparison.ps.gz.

[149] Henning Schulzrinne y Jonathan Rosenberg. *The Session Initiation Protocol: Providing Advanced Telephony Services Accross Internet*. *Bell Labs Technical Journal*, 1998.

[150] Henning Schulzrinne y Jonathan Rosenberg. *Internet Telephony: Architecture and Protocols – an IETF Perspective*. *Computer Networks and ISDN Systems*, Febrero 1999. http://www.cs.columbia.edu/ hgs/papers/Schu9902_Internet.ps.

[151] Henning Schulzrinne, Igor Slepchin y Jonathan Rosenberg. *SIP Frequently Asked Questions (FAQ)*. http://www.cs.columbia.edu/ hgs/sip/faq/cache/1.html, Febrero 2002.

[152] Henning Schulzrinne y Elin Wedlund. *Application-Layer Mobility using SIP*. *Mobile Computing and Communications Review*, 4, n. 3, Julio 2000. Disponible en http://www.cs.columbia.edu/ hgs/papers/Schu0007_Application.pdf.

[153] Secure Software Solutions. *RATS - Rough Auditing Tool for Security*. http://www.securesw.com/rats/rats.php, Septiembre 2002.

[154] René Seindal y François Pinard. *GNU m4 Macro Processor*. Free Software Foundation, 59 - Temple Place - Suite 330. Boston, MA 02111-1307, USA, 1.4 edición, Mayo 2002. Disponible en http://www.gnu.org/software/m4/.

[155] Taruni Seth, Albert Broscius, Christian Huitema y Huai-An P. Lin. *Performance Requirements for Signaling in Internet Telephony*. Informe técnico, Internet Engineering Task Force, Noviembre 1998. Borrador del IETF bajo el nombre draft-seth-sigtran-req-00.txt. Valido hasta el 16 de Mayo de 1999. http://www.cs.columbia.edu/ hgs/sip/drafts/draft-seth-sigtran-req-00.txt.

[156] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler y D.Ñoveck. *NFS version 4 Protocol*. RFC 3010, Internet Engineering Task Force, Network Working Group, Diciembre 2000. Estado: Proposed Standard.

[157] Peter Simons *et al.* *Autoconf Macro Archive*, Septiembre 2002. http://www.gnu.org/software/ac-archive/.

[158] Dorgham Sisalem y Henning Schulzrinne. *The Loss-Delay Adjustment Algorithm: A TCP-friendly Adaptation Scheme*. En *Network and Operating System Support for Digital Audio and Video (NOSSDAV)*. Cambridge, Reino Unido, Julio 1998. http://www.cs.columbia.edu/IRT/adaptive/.

[159] Dorgham Sisalem y Henning Schulzrinne. *The Direct Adjustment Algorithm: A TCP-friendly Adaptation Scheme*. En *Quality of Future Internet Services Workshop*. Berlín, Alemania, Septiembre 2000. http://www.cs.columbia.edu/IRT/adaptive/.

[160] Paul D. Smith, Richard M. Stallman y Roland McGrath. *GNU Make Manual*. Free Software Foundation, 59 - Temple Place - Suite 330. Boston, MA 02111-1307, USA, 3.79.1 edición, Junio 2000. Disponible en http://www.gnu.org/software/make/.

[161] Karen R. Sollins. *The TFTP Protocol (Revision 2)*. RFC 1350, Internet Engineering Task Force, Network Working Group, Septiembre 1992. Estado: Standard.

[162] Raj Srinivasan. *XDR: External Data Representation Standard*. RFC 1832, Internet Engineering Task Force, Network Working Group, Agosto 1995. Estado: Draft Standard.

[163] William Stallings. *Data and Computer Communications*. Prentice Hall, Inc., quinta edición, 1997.

[164] Richard M. Stallman *et al.* *GNU Coding Standards*, Agosto 2002. Disponible en http://www.gnu.org/prep/standards_toc.html.

[165] Richard M. Stallman *et al. Information for Maintainers of GNU Software*, Septiembre 2002. Disponible en http://www.gnu.org/prep/maintain_toc.html.

[166] W. Richard Stevens. *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley, 1994.

[167] W. Richard Stevens. *TCP/IP Illustrated, Volume 3: TCP for Transactions, HTTP, NNTP, and the UNIX Domain Protocols*. Addison-Wesley, 1996.

[168] W. Richard Stevens. *UNIX Network Programming, Volume 1: Networking APIs: Sockets and XTI*. Prentice-Hall, Second Edition edición, 1998.

[169] W. Richard Stevens. *UNIX Network Programming, Volume 2: Interprocess Communications*. Prentice-Hall, Second Edition edición, 1999.

[170] W. Richard Stevens y Gary R. Wright. *TCP/IP Illustrated, Volume 2: The Implementation*. Addison-Wesley, 1995.

[171] Randall R. Stewart, Qiaobing Xie, Ken Morneault, Chip Sharp, Hanns Juergen Schwarzbauer, Tom Taylor, Ian Rytina, Malleswar Kalla, Lixia Zhang y Vern Paxson. *Stream Control Transmission Protocol*. RFC 2960, Internet Engineering Task Force, Network Working Group, Octubre 2000. Estado: Proposed Standard.

[172] Randall R. Stewart, Qiaobing Xie, La Monte H.P. Yarroll, Jonathan Wood, Kacheong Poon y Ken Fujita. *Sockets API Extensions for Stream Control Transmission Protocol*. Borrador del IETF, Internet Engineering Task Force, Transport Area Working Group, Mayo 2002. Trabajo en desarrollo, bajo el nombre draft-ietf-tsvwg-sctpsocket-04.txt. Válido hasta Noviembre de 2002.

[173] Bjarne Stroustrup. *The C++ Programming Language*. AT & T Bell Telephone Laboratories, Incorporated. Addison-Wesley Publishing Company, tercera edición, Junio 1997.

[174] David Sugar y Federico Montesino Pouzols. *GNU ccRTP*. http://www.gnu.org/software/ccrtp/, Septiembre 2002.

[175] David Sugar y Federico Montesino Pouzols. *GNU Common C++ 2 Manual*, 1.0pre0 edición, Septiembre 2002. Distribuido junto con GNU Common C++ 2.http://www.gnu.org/software/commoncpp/.

[176] Sun Microsystems, Inc. *NFS: Network File System Protocol Specification*. RFC 1094, Internet Engineering Task Force, Network Working Group, Marzo 1989. Categoría: Informational. Obsoleto en favor de [156].

[177] Taligent, Inc. *Well-mannered object-oriented design in C+. Taligent's Guide to Designing Programs*. Addison-Wesley Publishing Company, Junio 1995. http://pcroot.cern.ch/TaligentDocs/TaligentOnline/DocumentRoot/1.0/Docs/books/WM/.

[178] Andrew S. Tanenbaum. *Computer Networks*. Prentice Hall PTR, tercera edición, 1996.

[179] Telecommunication Standarization Sector of ITU. *Information Technology – Abstrac Syntax Notation One (ASN.1): Specification of basic notatio*. Recomendación de la ITU-T X.680, International Telecommunication Union, Diciembre 1997. Disponible en

http://www.itu.int/ITU-T/studygroups/com17/languages/, donde también se pueden encontrar correcciones y ampliaciones de la especificación.

[180] UCL Network and Multimedia Research Group. *Mbone Conferencing Applications*. http://www-mice.cs.ucl.ac.uk/multimedia/software/, Septiembre 2002.

[181] Antti Vaha-Sipila. *URLs for Telephone Calls*. RFC 2806, Internet Engineering Task Force, Network Working Group, Abril 2000. Estado: Proposed Standard.

[182] Gary V. Vaughan, Ben Elliston, Tom Tromey y Ian Lance Taylor. *GNU Autoconf, Automake and Libtool*. New Riders Publishing, primera edición, Noviembre 2001. Disponible en http://www.andamooka.org/autobook/.

[183] Aparna Vemuri y Jon Peterson. *SIP for Telephones (SIP-T): Context and Architectures*. Borrador del IETF, Internet Engineering Task Force, SIPPING Working Group, Junio 2002. Trabajo en desarrollo, bajo el nombre draft-ietf-sipping-sipt-04.txt. Válido hasta Noviembre de 2002.

[184] Mark Wahl, Tim Howes y Steve Kille. *Lightweight Directory Access Protocol (v3)*. RFC 2251, Internet Engineering Task Force, Network Working Group, Diciembre 1997. Estado: Proposed Standard.

[185] Wenqing Jin and Kim Le and others. *Real Time Transport Protocol Library. Release 1.4.0*. http://www.vovida.org, Mayo 2002. Vovida Networks.

[186] David A. Wheeler. *Write It Secure: Format Strings and Locale Filtering*. *e-Security Journal*, Octubre 2000. http://www.dwheeler.com/essays/write_it_secure_1.html.

[187] David A. Wheeler. *Flawfinder*. http://www.dwheeler.com/flawfinder/, Septiembre 2002.

[188] David A. Wheeler. *Programming Secure Applications for Unix-linke Systems*. En *Free and Open Source Developers' European Meeting (FOSDEM) conference. Brussels, Belgium*. Febrero 2002. http://www.dwheeler.com/secure-programs/.

[189] David A. Wheeler. *Secure Programming for Linux and Unix HOWTO*. http://www.dwheeler.com/secure-programs/, Septiembre 2002.

[190] Jörg Widmer. *Equation Based Congestion Control*. Master's thesis, University of Mannheim / AT&T Center for Internet Research at ICSI (ACIRI), Febrero 2000. http://www.informatik.uni-mannheim.de/informatik/pi4/publications/html/widmer.html.

[191] Jörg Widmer y Mark Handley. *Extending Equation-based Congestion Control to Multicast Applications*. En *ACM SIGCOMM Conference on Applications, Technologies, Architectures and Protocols for Computer Communication*. San Diego, California, Agosto 2001. http://www.informatik.uni-mannheim.de/informatik/pi4/publications/html/widmer.html.

[192] Scott Williamson, Mark Kosters, David Blacka, Jasdin Singh y Koert Zeilstra. *Referral Whois (RWhois) Protocol V1.5*. RFC 2167, Internet Engineering Task Force, Network Working Group, Junio 1997. Categoría: Informational.

[193] Xiphophorus. *Ogg Vorbis: open, free audio*. http://www.vorbis.com, Septiembre 2002.

[194] Xiph.Org. *Xiph.Org: home. Building a new era of Open multimedia*. http://www.xiph.org, Septiembre 2002.

[195] Wengyik Yeong, Tim Howes y Steve Kille. *Lightweight Directory Access Protocol*. RFC 1777, Internet Engineering Task Force, Network Working Group, Marzo 1995. Estado: Draft Standard.

[196] Christian Zahl. *Aufbau und Konfiguration von Internet-Multimedia-Konferenzen über Verbindungen niedriger Bandbreite*. Diplomarbeit, Fachgebiet Telekommunikationsnetze, Institut für Offene Kommunikationsysteme, Fachbereich Informatik (FB 13), Technische Universität Berlin, Febrero 1997. Disponible en http://www.cs.columbia.edu/ hgs/papers/Zahl9702_Aufbau.ps.gz.

Índice alfabético