4 Implementación del cliente con QoS.

Tras el análisis realizado a los clientes de VoIP, se eligió twinkle por su capacidad de añadir nuevos hilos de ejecución al programa principal, todo ello gracias a que dicho cliente fue diseñado y concebido como una aplicación multihilo.

La ventaja de desarrollar un cliente de software libre es disponer del código fuente permitiendo esto la localización de los ficheros claves en la comunicación por VoIP. Estos ficheros son aquellos que se encargan de la comunicación con el sistema de audio del equipo y con la parte de la red dedicada a la recepción del flujo RTP.

Para la localización de estos ficheros se recomienda seguir los pasos del código fuente durante realización de una comunicación. La forma más sencilla de encontrar esos ficheros es observar la sucesión de acontecimientos llevados a cabo por el protocolo SIP al realizar una llamada de VoIP. Tras negociar el códec a utilizar y recibir su aceptación, comenzará el proceso de formación de los paquetes RTP, primero se capturarán las tramas de voz, a continuación se codificarán y por último serán transmitidas al extremo remoto.

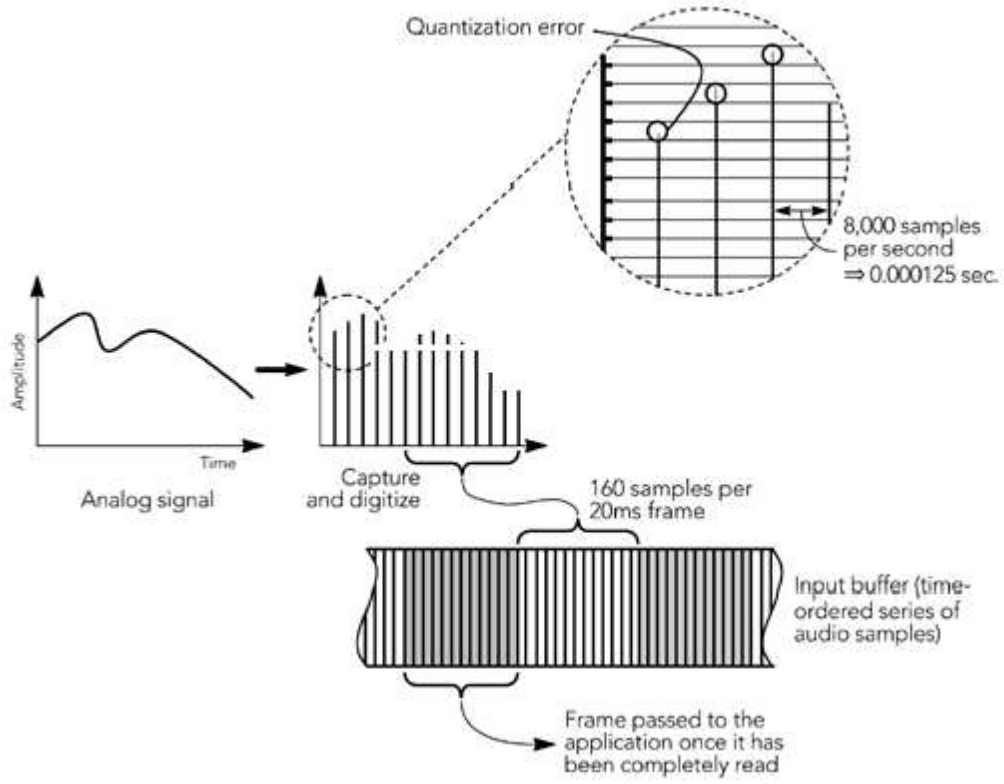


Ilustración 17: Captura, digitalización y formación de tramas de voz.

Estas tramas de voz de duración variable (30ms en el caso de usar códec G.723.1) se almacenan en un buffer a la espera de ser recogidas, empaquetadas y enviadas por la red.

Un cliente de VoIP tradicional introduce cada trama de voz en un solo paquete RTP, de manera que las funciones del cliente que se encargan de crear los paquetes RTP deben de ser modificadas y adaptadas para nuestros propósitos.

Es por ello que a continuación y en los siguientes apartados se va a explicar cuales son los cambios necesarios a desarrollar para obtener un cliente con soporte para la calidad de servicio.

4.1 Cambios a realizar en el cliente.

El primero de los cambios a realizar en el cliente de VoIP, es añadir las modificaciones sobre las funciones de red para que soporten el sistema de empaquetado de tramas basado en el Nffp (número de tramas de voz por paquete), es decir, el sistema de creación de los paquetes RTP. Siempre se debe de tener en cuenta la RFC 3550, que proporciona las reglas para realizar dicho empaquetado, de forma que permita mantener la compatibilidad entre clientes que implementen la misma RFC.

Antes de comenzar a ver cuales son los cambios que se van a realizar, es menester localizar los ficheros del código fuente que van a ser modificados, por lo tanto se va a mostrar a continuación una lista con dichos ficheros:

- src/audio/audio_tx.cpp/h
- src/audio/audio_session.cpp/h
- src/session.cpp/h
- src/audio/audio_codec.cpp/h
- ...

Por supuesto para la adaptación de algunas de las funciones ha sido necesario la modificación de la definición de algunas de ellas, pero dichas modificaciones se consideran de bajo impacto y son consideradas de baja importancia para el desarrollo del proyecto. Esas pequeñas modificaciones son simplemente la realización de las rutinas oportunas para la correcta compilación e integración en el código fuente previamente desarrollado.

Los ficheros anteriormente listados son los que se han modificado en el código fuente, no solo para integrar las funcionalidades que soportan la calidad de servicio, sino que además han sido

añadidas nuevas funciones, nuevas variables, etc. En definitiva todas esas modificaciones ha significado una completa reestructuración del código fuente original y por consiguiente son motivos suficientes para ser mencionados en este apartado de la memoria.

No solamente se ha reestructurado parte del código fuente original, o se ha limitado a realizar una simple integración de las nuevas funciones, sino que se han añadido nuevos ficheros fuente, que contienen dichas modificaciones para añadir e integrar el soporte de QoS al cliente elegido (twinkle). Los ficheros que han sido añadidos son:

- directorio src/audio/g723: contiene todos los ficheros que componen el total de funciones que hacen posible la codificación y decodificación de las tramas de audio al códec G.723.1.
- audio_net_tx.cpp/h
- audio_play_tx.cpp/h

Hasta el momento se ha presentado una serie de ficheros que contienen los cambios necesarios para convertir el cliente elegido en un cliente con soporte para QoS, pero todavía no se ha mencionado cuales son dichos cambios, por lo tanto se va a proceder a ver los cambios que se han adoptado.

Los cambios más importantes que se han llevado a cabo, están localizados en el directorio src/audio, es en este directorio donde se concentra prácticamente la totalidad del código fuente que se necesita y sobre el que se ha trabajado para añadir el soporte para la calidad de servicio (QoS).

La calidad de servicio (QoS) consiste en reproducir las tramas de voz en los instantes de tiempo correspondientes, sin que se produzcan retrasos o perdidas de la información.

Para realizar la adecuada reproducción de las tramas de voz es fundamental llevar a cabo una correcto uso de los buffer de comunicación del sistema de audio.

Por ejemplo, si se realiza la reserva de los recursos por Internet, se supone que no existe el efecto jitter, es decir que no hay retrasos en la transmisión de la voz. Pero si al introducir los datos en el buffer de la tarjeta de sonido se insertan una cantidad mayor que la soportada por el mismo, se producirá un desbordamiento del buffer con la consiguiente pérdida de información, es decir la reproducción de las tramas de voz están siendo adulteradas y no obteniéndose así la calidad de sonido esperada. Por otro lado, si no se completase el tamaño total del buffer se introducirían fenómenos inesperados tales como ruidos, chasquidos, etc, no relacionados con los retrasos o pérdidas de la información.

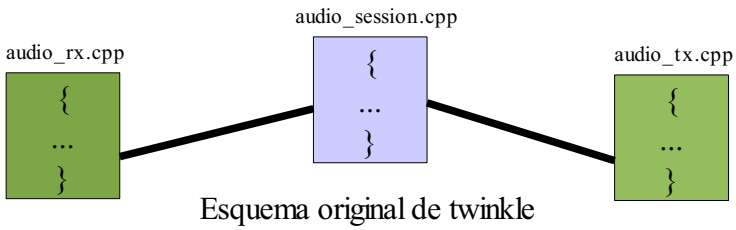
Por tanto se debe de disponer de una correcta implementación de las funciones que intercambian información con la tarjeta de sonido, con el objeto de no adulterar las tramas de sonido que nos llegan a través de la red y que posteriormente serán reproducidas.

Precisamente el fichero ***audio_rx.cpp/h*** es el encargado de contener las funciones que hacen posible la recepción de las tramas de audio procedentes de la tarjeta de sonido y que tras procesarlas adecuadamente (codificarlas y empaquetarlas) envía la información al hilo encargado de transmitir los paquetes RTP.

En las comunicaciones de VoIP al realizar una llamada se crea una sesión multimedia (audio) entre los extremos remotos, siendo el protocolo llamado SIP el encargado de negociar las condiciones de la comunicación entre los mismos.

Twinkle tiene un fichero llamado ***audio_session.cpp/h*** encargado de contener las funciones necesarias para la creación de los flujos de audio utilizados en el protocolo RTP. Estas funciones hacen posible el establecimiento de los hilos (procesos) que manejan dichos flujos y permiten la comunicación entre los extremos.

En la siguiente ilustración se puede ver un pequeño esquema representativo de la creación de los hilos que manejan los flujos de audio en la comunicación por RTP.



Esquema original de twinkle

El fichero ***audio_tx.cpp/h*** es el encargado de contener las funciones necesarias para la recepción de los paquetes de voz RTP procedentes de la red y de enviar las tramas de audio contenidas en ellos a la tarjeta de sonido del equipo.

Twinkle por si solo no tiene soporte para ofrecer la calidad de servicio (QoS). Pues como se ha visto en el apartado anterior no cumple con las características siguientes:

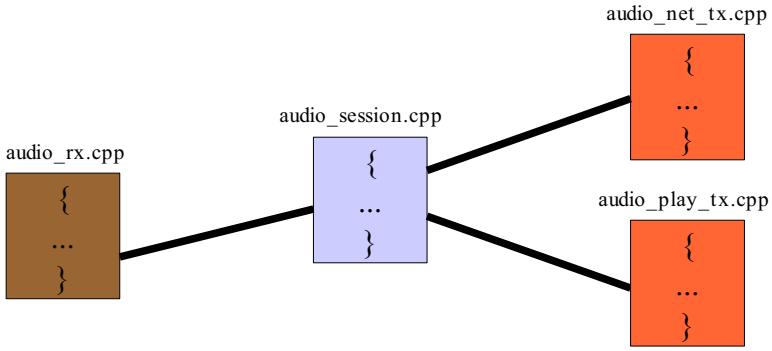
- Empaquetado y Desempaquetado
- Algoritmo de buffer de playout
- Parámetros configurables y el códec G.723.1

Para ofrecer la calidad de servicio se van a añadir las funciones de empaquetado, desempaquetado, así como los algoritmos de playout y las condiciones que se deben de cumplir

para la aceptación de las tramas que se reciban.

El esquema original de twinkle, va a ser modificado dividiendo el fichero *audio_tx.cpp* en dos procesos independientes:

- control de la recepción de paquetes RTP procedentes de la red
- control de la reproducción de las tramas de voz



Esquema modificado para añadir soporte QoS a twinkle

Al observar el esquema se puede ver claramente la modificación que se ha llevado a cabo: por un lado, se ha adaptado convenientemente el fichero *audio_session.cpp* de modo que sea capaz de crear los nuevos hilos *audio_net_tx.cpp* y *audio_play_tx.cpp*, y por otro lado se han creado dichos ficheros con las funciones necesarias para ofrecer la calidad de servicio (QoS) deseada.

- ***audio_net_tx.cpp***: consiste en esperar a que llegue un paquete RTP, una vez recibido el paquete RTP se ha de analizar la carga que contiene, determinándose la longitud del mismo y averiguando cual es el número de tramas (Nfpp) que contiene. Posteriormente se analiza si dicho paquete ha llegado a tiempo de reproducirse o no, es decir, dependiendo del *retraso de playout* configurado se puede conocer si la primera trama que contiene dicho paquete ha llegado a tiempo de reproducirse o no lo ha hecho. En caso de no llegar en tiempo de reproducción la trama sería descartada y se continuaría con el análisis de la siguiente trama contenida en el mismo paquete. Debido a la complejidad que esto conlleva, esta característica no ha sido implementada proponiéndose como una posible linea de avance en el desarrollo del cliente VoIP.

  En este proyecto, se ha considerado que si la primera trama no llegase a tiempo se descartará el paquete por completo. Posteriormente se comprobará si ha habido un silencio, de ser así se procederá al cálculo del nuevo valor del parámetro *retraso de playout.* Para esta tarea se usarán los algoritmos de cálculo de buffer de playout convenientemente integrados en el archivo *audio_codec.cpp/h*. Sin embargo, si el paquete recibido llegase a tiempo de reproducirse, se desencapsulará y se introducirá en la cola de espera para que el hilo de play_tx lo capture y lo envíe a la tarjeta de audio para su reproducción.

- ***audio_play_tx.cpp***: es el hilo encargado de sacar los elementos de la cola de audio, y enviarlos al sistema de sonido del equipo. Este hilo se ejecuta continuamente a la espera de

obtener información de la cola de audio. Una vez que la cola de audio contiene datos, este hilo lo extrae y comprueba que se encuentra en su instante de reproducción, de estarlo se reproducirá en la tarjeta de sonido y volverá a esperar a que le llegue nuevos datos, en caso contrario el paquete será descartado, volviéndose el mismo al estado de espera.

El tiempo que va a estar suspendido dicho hilo dependerá de los periodos de silencios acaecidos. Si hay un periodo de silencio se reiniciará el valor del temporizador al retraso de playout, y se esperará a que lleguen nuevas tramas, corrigiéndose de este modo el efecto jitter de la red. Sin embargo, si la trama que ha sido extraída de la cola de audio para su reproducción no es la primera de la ráfaga, es decir, no se ha producido un periodo de silencio, el hilo se suspenderá solamente el tiempo que dure dicha trama, es decir, 30 ms, 60 ms , 90 ms ó 120ms, estos valores dependerán del códec que se este utilizando. En este caso los valores coinciden con los periodos de tiempo de las tramas pertenecientes al códec G.723.1, siendo 120ms el máximo valor permitido por la norma (se corresponde con Nffp=4).

Hasta ahora solamente se han comentado las funciones que proporcionan la calidad de servicio, en el sentido que va desde el cliente remoto al cliente local, es decir, que tanto solo se han comentado las funciones que son necesarias para recibir paquetes que provienen de un cliente con soporte para la calidad de servicio:

- desempaquetado y decodificación
- calculo de spike
- calculo del retraso de playout
- reproducción de las tramas de voz

Finalizada la explicación de las funciones encargadas de la recepción de paquetes se podría decir que se tiene el cliente de VoIP a medio realizar. Faltaría detallar las características y funciones de la otra mitad del cliente. Estas funciones se dedican a codificar y empaquetar los datos que van a ser enviados por RTP al otro extremo de la comunicación. El fichero que se ha modificado para recoger dichas tareas es ***audio_rx.cpp.***

• ***audio_rx.cpp:*** este fichero contiene las funciones necesarias para realizar la codificación de las tramas de audio capturadas por la tarjeta de sonido. La captura de las tramas se realiza en periodos de 30 ms. Una vez obtenidas las tramas son codificadas y encapsuladas en un paquete RTP único. El sistema de encapsulado va a trabajar conforme a la norma RFC 3551 aplicable para el códec G.723.1. Cuando se selecciona cualquier otro códec que no sea el G.723.1, el funcionamiento del programa es el mismo que originalmente, es decir no se realiza el empaquetado (QoS) para otro códec que no sea el G723.1.

Los cambios realizados en el cliente se corresponden a la adaptación e integración del soporte QoS para el códec G723.1.

De modo que a excepción de los ficheros antes mencionados, se han retocado algunos otros ficheros de menor importancia y que vienen perfectamente identificados en la documentación adjunta, pero sobre todo se han adaptado algunas funciones para que soportasen los nuevos parámetros de configuración:

- Nffp
- Algoritmo usado
- códec G723.1: VAD y régimen binario.

En el siguiente apartado se va a describir las estructuras software que han sido añadidas al programa original, es decir, se va a realizar una explicación detallada de los algoritmos principales que hacen posible la prestación de la calidad de servicio objetivo fundamental de este proyecto.

4.2 Estructura de software realizado.

Desde el principio se esta haciendo hincapié en el diseño basado en la creación de hilos independientemente de las funciones que realizan el soporte para la calidad de servicio. Es por ello que en este apartado se va a seguir la siguiente tónica a la hora de exponer las diferentes estructuras de código utilizado, primero se va a observar el diagrama de flujo de la estructura algorítmica implementada y posteriormente se analizará el código fuente en su esencia.

En la siguiente ilustración se puede observar la estructura general del software realizado, esto es, se va a ver la estructura modificada y adaptada al cliente de software libre twinkle. Posteriormente se irán analizando las diferentes estructuras de datos utilizadas y especialmente los módulos que han sido añadidos y/o modificados para hacer posible el soporte de la calidad de servicio.

Se observan tres estructuras de software como piezas angulares en el desarrollo del proyecto:

- audio_rx.cpp
- audio_net_tx.cpp
- audio_play_tx.cpp

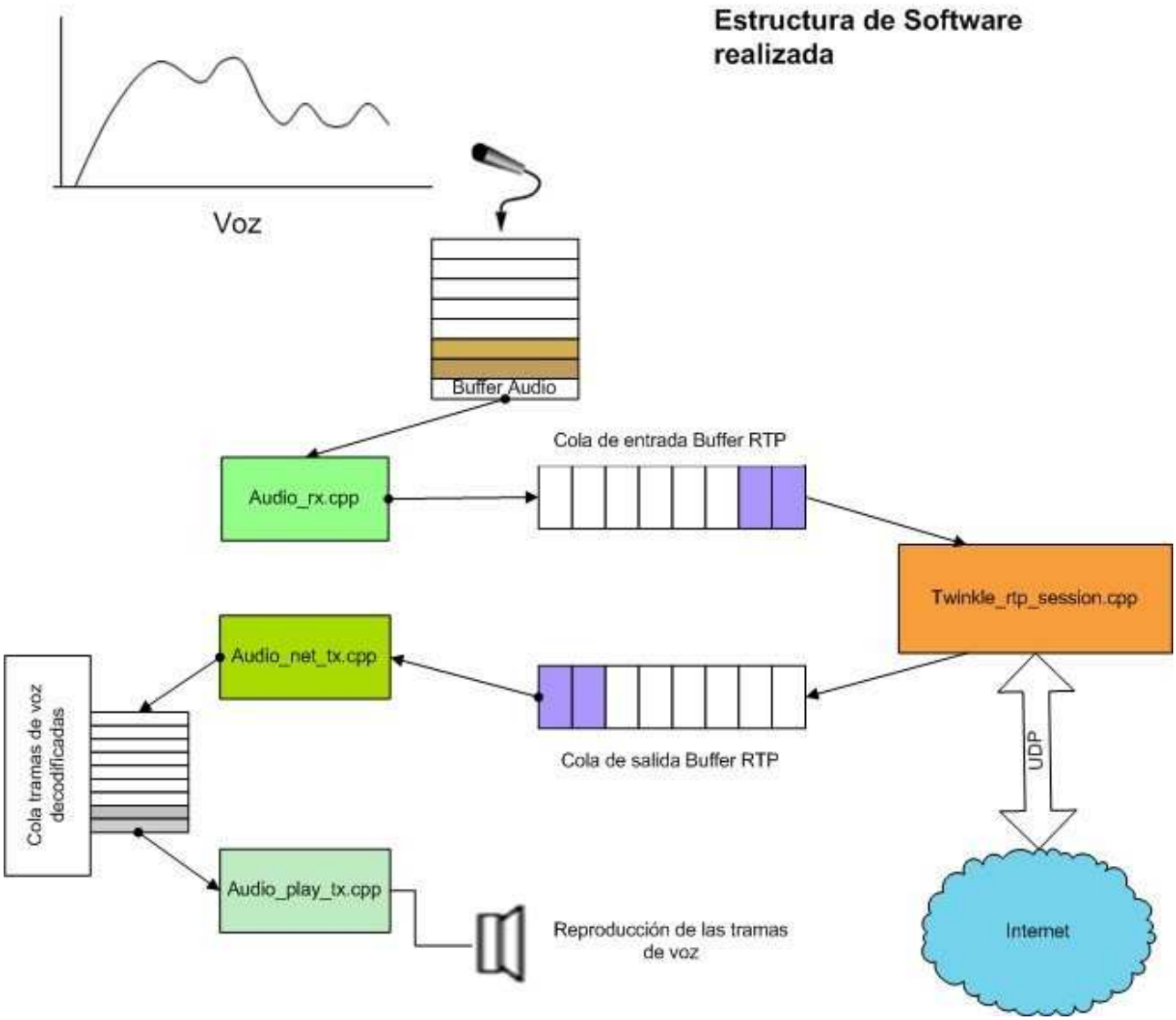


Ilustración 18: Estructura de software realizada.

Se ha decidido comenzar a exponer la estructura de código contenida en la función principal del hilo en ejecución del fichero *audio_rx.cpp*.

Obviamente cuando se crea un hilo, este debe de contener la función principal del mismo, que recibe el nombre de run() (ejecución del hilo). Esta función debe de recoger la secuencia de tareas que se desean realizar en el momento de la ejecución. En el fichero *audio_rx.cpp* contiene todas las funciones necesarias para capturar los samples de audio extraídos de la tarjeta de sonido, codificarlos, realizar el empaquetado y enviarlos al extremos remoto usando RTP.

En la ilustración 18, se puede observar el diagrama de flujo de dicha función principal.

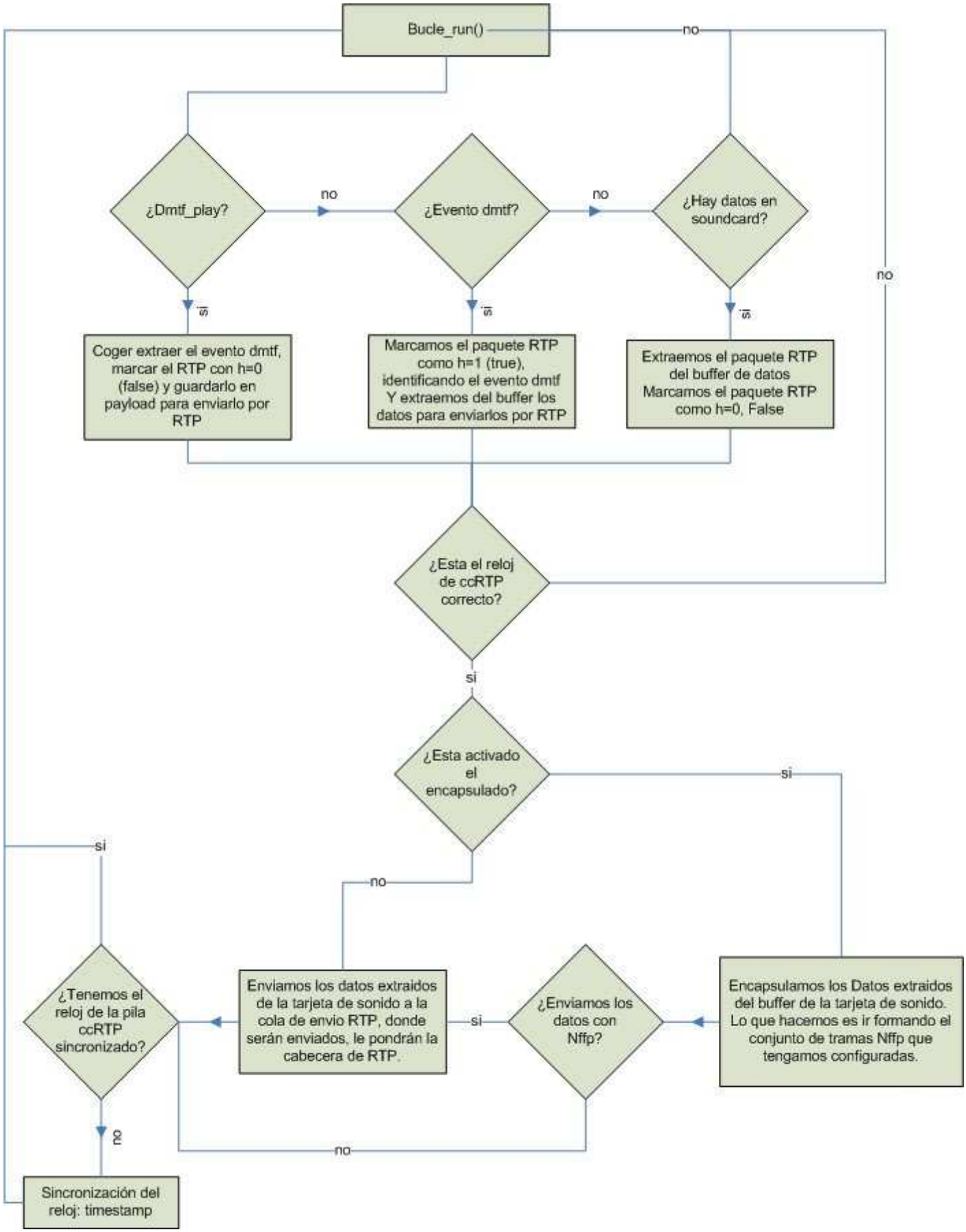


Ilustración 19: Diagrama de bloques de la función principal del fichero audio_rx.cpp

En el análisis al programa twinkle, se ha comentado que dicho cliente esta programado con ayuda de librerías externas. Una de estas librerías es la denominada *ccRTP*, esta librería implementa completamente el protocolo RTP, y posee funciones básicas para añadir carga a las cabeceras de este protocolo, configurarlas y enviar los paquetes generados al extremo de la comunicación.

Observando la ilustración 19, se puede llegar a intuir que el método utilizado es relativamente sencillo. La ayuda facilitada por la librería hace posible que en la creación de los paquetes RTP no sea necesario la configuración de las cabeceras, sino solamente se deba de introducir los datos de voz que van a ser transportados por el datagrama UDP. Esto implica que en el desarrollo del cliente solo se ha de preocupar por la implementación de las funciones que hacen posible el envío de las tramas de voz al otro extremo de la conversación.

En la ilustración 20, se observa el diagrama de bloques del fichero *audio_net_tx.cpp*, en dicho diagrama puede observarse con total detalle la estructura de software que ha sido añadida a la originalmente desarrollada por el autor de twinkle, en este caso, se ha implementado todo el soporte para la detección y cálculo del retraso de playout, dependiendo del tipo de algoritmo configurado. Además se realiza la detección de los paquetes que llegan fuera de tiempo de reproducción y se descartan.

La detección de los paquetes que se reciben con un retraso superior al estipulado, son descartados por completo independientemente si llevan una o varias tramas por paquete. Realizar una exhaustiva comprobación de las tramas contenidas en ellos proporciona una línea de mejora en el comportamiento del programa. Esta línea de actuación consistiría en detectar y comprobar cada trama empaquetada por separado, con el fin de detectar cuales de estas llegan o no a tiempo de reproducción. Con esto queda patente que el método utilizado podría estar descartando tramas válidas sin realizar las oportunas comprobaciones y corroboraría la mejora argumentada.

La comunicación con el hilo de reproducción se realiza a través de una cola de datos, en la que se almacenan todas las tramas de voz decodificadas y listas para ser reproducidas. En el módulo *audio_net_tx.cpp* se reciben las tramas RTP, se validan y se calcula el nuevo valor del retraso de playout dependiendo de si anteriormente hubo o no un silencio. Posteriormente si las tramas son válidas se procesan y se dejan listas y preparadas para su reproducción. Entonces son añadidas a la cola de tramas decodificadas, lugar donde el hilo encargado de la reproducción de las mismas estará observando continuamente para extraerlas y enviarlas al sistema de audio del equipo.

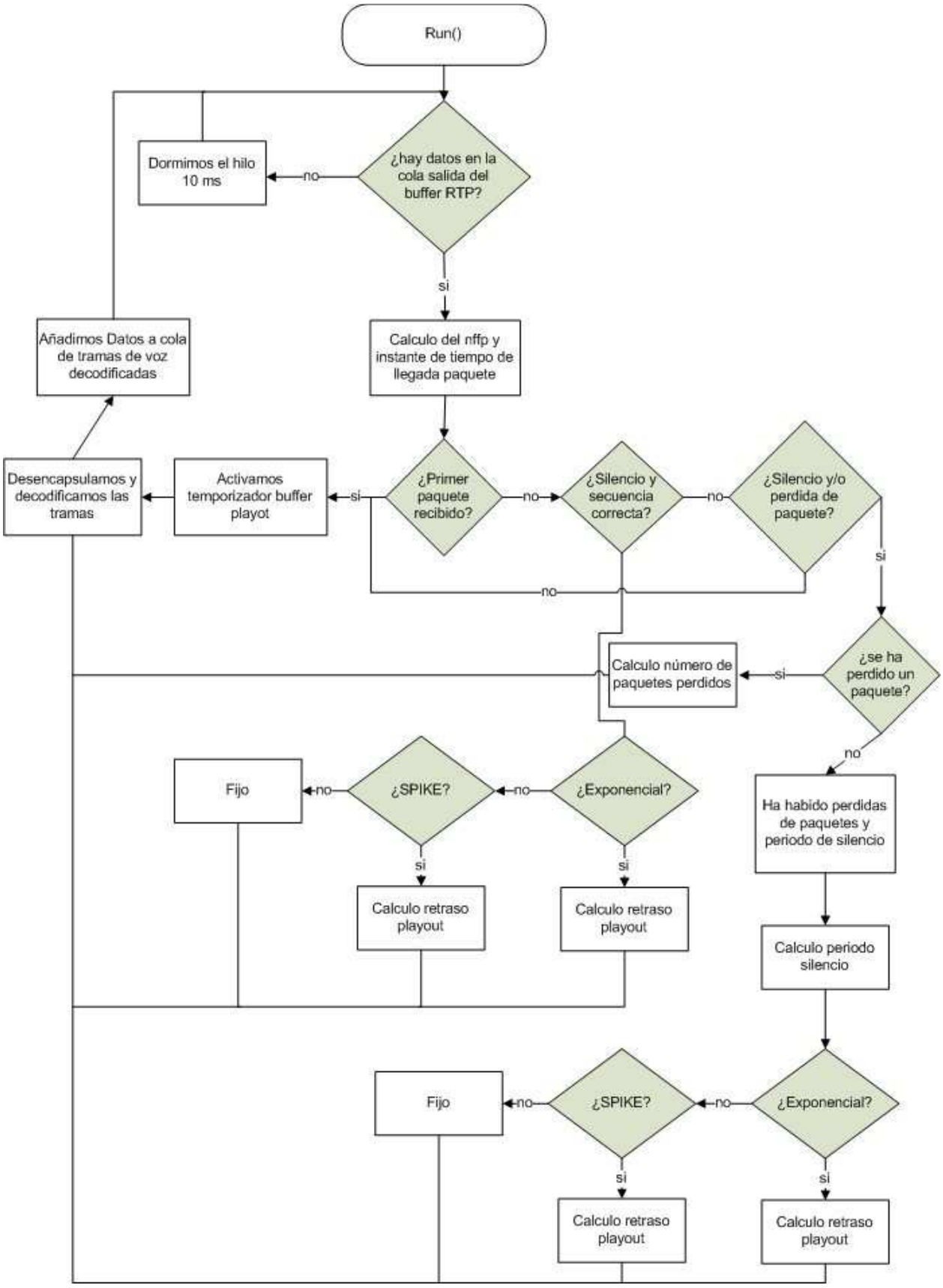


Ilustración 20: Estructura software audio_net_tx.cpp

Siguiendo con el esquema visto en la ilustración 18, se va a explicar a continuación en que consiste el modulo *audio_play_tx.cpp*.

audio_play_tx.cpp es el hilo de ejecución encargado de enviar las tramas de voz al sistema de audio del sistema operativo, el módulo audio_net se encarga de procesar los paquetes RTP e introducirlos en una cola por orden de llegada[7], posteriormente el hilo de ejecución audio_play_tx recoge dichas tramas, comprueba que están dentro de su instante de reproducción y las envía al sistema de audio. En caso de no estar en su instante de reproducción dichas tramas son descartadas, y se contabilizarán las perdidas, además se calcularán las tramas que van a ser retenidas para ser reproducidas de nuevo cuando falte alguna de las tramas siguientes, es lo que se denomina sistema de "*conceal packet loss*".

En la ilustración 21 puede observarse el diagrama de bloques del modulo *audio_play_tx.cp*p. Observándose que el hilo de ejecución calcula el tiempo que va ser suspendido entre trama y trama, dependiendo este del tamaño del paquete recibido.

Se ha decidido enviar todas las tramas juntas al sistema de reproducción, evitando así cortes en el relleno del buffer del sistema de audio, y permitiendo una correcta reproducción de las mismas. De esta forma el hilo de reproducción se va a suspender el tiempo determinado por las tramas contenidas en el paquete recibido. Transcurrido este tiempo el hilo de ejecución es despertado comprobando si hay nuevas tramas disponibles en la cola y volviendo a repetir el proceso de reproducción antes mencionado.

En la ilustración 18, aparecía un módulo denominado *twinkle_rtp_session.cpp* dicho módulo es el encargado de facilitar la comunicación entre la librería ccRTP y el programa twinkle. Por tanto mediante el módulo *twinkle_rtp* se pueden utilizar cómodamente las funciones que están incluidas en la librería mencionada, para así establecer el flujo de datos RTP entre los clientes remotos. Este módulo es el encargado de manejar el flujo bidireccional entre ambos extremos.

7 Como linea de avance se propone realizar una ordenación de los paquetes antes de introducirlas en las cola de tramas decodificadas.

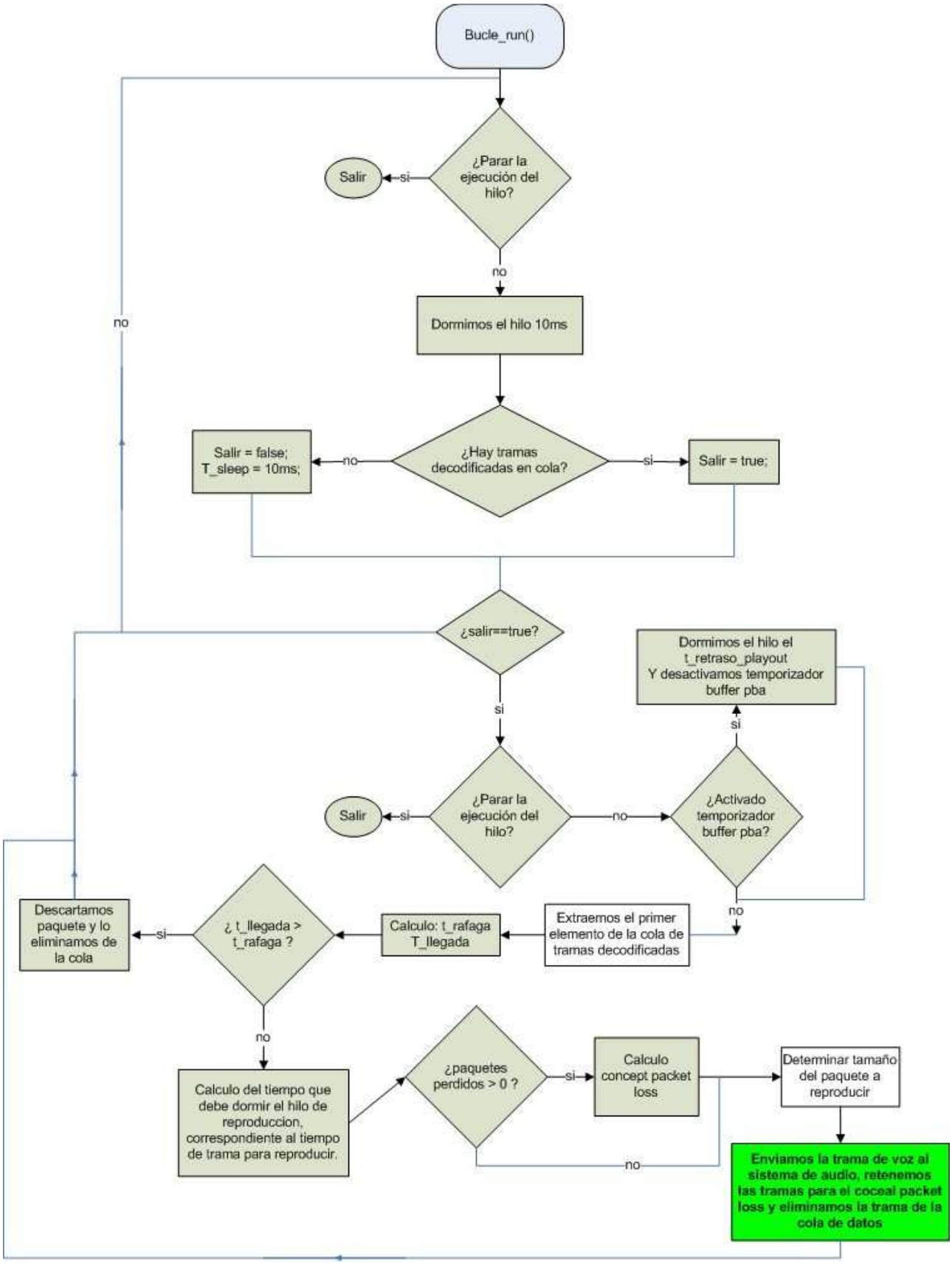


Ilustración 21: Estructura de software audio_play_tx.cpp

4.3 Pruebas de funcionamiento realizadas.

Para el desarrollo del cliente de comunicaciones por VoIP se ha dispuesto de:

- 2 PC
- 1 router
- 1 servidor proxy VoIP

La implementación ha sido llevada a cabo bajo la distribución Kubuntu Hoary 5.10, dicha distribución linux esta basada en ubuntu, con un entorno de ventanas denominado KDE, de hecho la base sobre la que descansa dicha distribución es la rama estable de debian. Es por ello que utiliza el mismo sistema de paquetes para la instalación de software.

Para crear un ambiente de desarrollo lo más parecido a un entorno real de trabajo, se tiene que instalar la distribución Kubuntu en dos ordenadores y que se conecten a una red LAN mediante un router ADSL, a su vez dicho Router debe de estar conectado a Internet lo que permitirá aumentar la fiabilidad de las pruebas.

Uno de los PC es un equipo portátil mientras que el otro equipo es un PC de sobremesa, las características de los equipos son:

- Portátil: 1,6Ghz Intel Centrino y 512MB de RAM.
- Sobremesa: 733Mhz con 320MB de RAM.

Puesto que el cliente de comunicaciones por VoIP esta basado en SIP, era necesario disponer de un servidor de VoIP. Un servidor de VoIP consiste en una herramienta que se encarga de registrar a los usuarios, con el objetivo de mantener una lista de los usuarios que están conectados y poder dirigir el tráfico hacia los mismo, dependiendo de las solicitudes. Es decir, cuando un cliente de VoIP se inicia es recomendable darse de alta en un servidor de VoIP para poder localizar a los usuarios de forma rápida y sencilla.

Por este motivo se instaló el servidor www.openser.org en la maquina portátil de manera que uno de los clientes comunmente el que se encontraba iniciado en la máquina de sobremesa se registraba en el servidor proxy de VoIP instalado en el ordenador portátil, mientras que el cliente desarrollado en el ordenador portátil era registrado en un servidor de Internet denominado ekiga.net.

No solo se han realizado las pruebas de funcionamiento entre clientes twinkle, sino que además se ha comprobado la compatibilidad con otros clientes como el caso de kphone, ekiga, etc...

En general cualquier cliente de VoIP que soporte los mismo códec y el protocolo SIP será capaz de entenderse con el cliente de twinkle en su versión sin empaquetado. En el momento en el que el sistema de empaquetado es activado solamente será capaz de entenderse con el cliente de comunicaciones twinkle o cualquier otro que soporte dicho empaquetado, de hecho una versión de pruebas desarrollada en el cliente kphone verifican que efectivamente existe la compatibilidad deseada.

Por tanto las pruebas se realizaban de la siguiente forma:

1. Iniciar el servidor Proxy de VoIP (openser.org).
2. Iniciars los clientes twinkle en ambos ordenadores.
3. A continuación, una vez obtenido el registro satisfactoriamente en los servidores local y ekiga.net, se puede realizar una llamada de VoIP al identificador SIP que comunmente se escribe: usuario@dominio.com (en caso de no tener un dominio se puede indicar la dirección IP directamente).
4. Con el identificador de la persona a la que se desea realizar la llamada, solamente se tiene que darle al botón de llamada y el cliente de VoIP se encargará de realizarla.
5. Se esta manteniendo una conversación de VoIP (basada en los parámetros que previamente se ha de haber configurado en la interfaz para tal efecto).
6. Para terminar la llamada tan solo tenemos que darle al botón de finalizar.

Básicamente estos son los pasos que se deben de dar para probar dicho cliente, tal y como se debería de hacer para probar cualquier otro cliente de comunicaciones de VoIP, la diferencia principal es que el cliente desarrollado para este proyecto no dispone de un sistema de paquetes binarios de fácil instalación, sino que únicamente se dispone del código fuente que se ha desarrollado bajo una condiciones concretas y que se deben de cumplir para poder compilar con éxito el programa y por consiguiente poder ejecutarlo.

No obstante en el anexo se encuentra una guía de usuario que proporciona los pasos que se necesitan seguir para que una vez instalado el sistema operativo con las herramientas de programación necesarias, este pueda compilarse y ejecutarse satisfactoriamente.

4.4 Problemas encontrados.

La principal dificultad ha sido comprender correctamente la implementación realizada por el autor principal del software elegido, de hecho gran parte del tiempo de implementación ha sido empleado en comprender el esquema y la estructura del software, es decir, conocer internamente todo el funcionamiento, sus ficheros, la localización de las funciones, la utilización de cada una de las mismas y sobre todo se ha concentrado la atención en todas las funciones que permitían controlar el funcionamiento de los paquetes RTP, el sistema de audio y la interfaz gráfica.

Una vez examinado completamente el código fuente del autor, se puede comenzar a conocer la mecánica de compilación de la herramienta GNU (software libre), este punto es relativamente sencillo, al estar desarrollando software bajo linux la instalación de las herramientas para su desarrollo es bastante sencillo puesto que tan solo se tiene que buscar los paquetes adecuados. Normalmente todas las librerías y herramientas que se necesitan están en paquetes binarios y por lo tanto su instalación es muy sencilla. En esta ocasión algunas de las librerías usadas por el cliente twinkle han tenido que ser compiladas parar la distribución en cuestión y los errores que pudieran producirse se deberán solucionarse dependiendo de los mensajes de error propios de cada sistema operativo.

Cuando se dispone de la herramienta de trabajo completamente configurada y lista para ser utilizada, es cuando se comienza a desarrollar código fuente.

La parte más complicada ha sido conseguir que las estructuras de reproducción y red estén separadas y sincronizadas, de hecho con el primer cliente de software libre elegido, kphone, su error de diseño que no contemplaba la necesidad de dividir el programa en procesos independientes unos de otros[8], dificulto la creación de dos nuevos módulos separados para la recepción de paquetes y la reproducción de las tramas, de hecho tal fue dicha complicación que prácticamente fue imposible la adaptación de dicho código al soporte de calidad de servicio y para conseguir dicha adaptación era necesario la reestructuración desde la base del proyecto lo que nos conllevaría un excesivo trabajo y la consiguiente pérdida de tiempo. Es por ello que tras sopesar e intentar varias veces sin éxito la adaptación de dicho código se decidió elegir un segundo cliente que fuera sencillo y que a su vez tuviera un diseño basado en programación multihilos.

Con este segundo cliente la implementación de la separación de los procesos de recepción de paquetes y reproducción fue llevada a cabo con éxito y sin ningún tipo de problema, salvo en el momento de sincronizar los tiempo de ejecución de cada hilo.

En el código de kphone, los intentos por sincronizar dicho hilos de ejecución dieron lugar al abandono de dicho cliente, de echo cuando se comenzó a tener problemas con el cliente twinkle, se

8 En definitiva concebir dicha aplicación como un conjunto de miniaplicaciones que coexisten y se comunican entre sí.

detecto un problema de granularidad en la ejecución de la función que permite que el hilo en ejecución se duerma. Este problema de granularidad implementada por el sistema operativo, no permitía que ambos hilos se fueran a dormir con la misma función, es decir, cuando deseamos que un hilo de ejecución se duerma se realiza un llamada a una función del sistema al que le podemos pasar los datos en nanosegundos, microsegundos, milisegundos... por tanto si se mandaban los datos en ambos hilos con una granularidad de nanosegundos, se obtenía un efecto no deseado. Cada hilo se ejecutaba de manera aleatoria y a expensas del planificador del sistema operativo. En cambio cuando se efectuó la llamada a la función utilizando microsegundos, se observo que su funcionamiento era el correcto, es decir, uno de los hilo se envía a dormir con granularidad de nanosegundos, mientas que el otro se envía a dormir con granularidad de microsegundos, esto provoca el correcto funcionamiento del planificador del sistema, otorgando a cada hilo los tiempos de ejecución que realmente les pertenecen.

Una vez que se solvento dicho problema, el resto de la implementación resulto ser más sencillo, consistía en ir añadiendo código fuente e ir enlazando las funciones adecuadamente para que puedan ser utilizadas allí donde eran necesarias, de hecho, solamente se encontró una pequeña dificultad en la obtención del tiempo de llegada en milisegundos, puesto que las funciones que realizan el manejo del reloj del sistema no proporcionan la facilidad de extraer los milisegundos contenidos en un instante de tiempo determinado, es por tanto que se tuvo que diseñar la manera de obtener dicho dato y que pudiera ser utilizado con éxito para los propósitos del proyecto en cuestión.

Para terminar varios de los problemas que se han encontrado han estado relacionados con la compilación del programa, o la depuración de los problemas existentes, quizás en todo desarrollo de software la parte que más problemas pueda presentar es la depuración de los fallos, no solo de compilación sino de funcionamiento.