6 Anexo

6.1 Guía de usuario para la instalación del cliente twinkle.

Lo primero que se debe de hacer antes de comenzar la instalación, es asegurarse de que se cumplen, al menos, los requisitos que nos indica el autor en su página web:

- Librerías ccRTP compuestas por commoncpp2-1.3.21.tar.gz (versión 1.3 o superior) y ccrtp-1.3.6.tar.gz.
- Librerías de desarrollo de Qt for open source (3.3.0 o superior).
- Librería de sonido libsndfile.

Se da por supuesto que se poseen todas las herramientas necesarias instaladas en el equipo para el desarrollo de software en C++ bajo el sistema operativo Linux.

Esas librerías se deben instalar en el sistema antes de ponerse a compilar twinkle. Se recomienda instalar las librerías en el orden estipulado en el párrafo anterior.

Una vez que se ha completado con éxito la instalación de las librerías anteriormente mencionadas, se debe de configurar las variables de entorno del sistema linux, puesto que sin dicha configuración aparecen ciertos problemas durante el proceso de compilación del programa.

1. *export LD_LIBRARY_PATH=/usr/local/lib*
2. *export QT_DIR=/usr*

Los comandos que puedes observar, deben de ejecutarse en un terminal de consola cada vez que se inicie por primera vez en el sistema y no se termine dicha aplicación, esto quiere decir que si por alguna razón cerramos el terminal de consola, y se vuelve a abrir uno para realizar una nueva compilación desde “cero”, esto implica volver ha realizar ./configure, es entonces cuando se deberá volver a ejecutar los comandos que configuran las variables de entorno, por ello se recomienda la creación de un cript que nos evite tener que escribir repetidas veces dichas variables de entorno.

Continuando con la guía, se va a proceder a compilar el código fuente del cliente twinkle, para lo que se deberá ejecutar lo siguiente:

1. **./configure**

   Si todo ha sido un éxito, y no ha habido problemas (que se tendrán que resolver dependiendo del sistema operativo, y de sus respectivos mensajes de error) se pasará a configurar el *Makefil*e más problemático, pues ha sido generado por el autor y se tiene que añadir manualmente la nueva librería de audio que contiene el nuevo códec G.723.1.

   Al proyecto original se le han añadido algunos directorios y ficheros que se van a detallar a continuación:

   /src/audio/g723

   En este directorio se han añadido los ficheros que contienen las funciones necesarias para añadir un nuevo códec al cliente de software twinkle. Por tanto, puesto que se ha añadido dicho soporte en forma de librería dinámica se debe de realizar lo siguiente para su correcta compilación:

   1. Abrir el fichero Makefile que se encuentra en dir_proyecto/src/gui
   2. Buscar la cadena LIBS =
   3. Sustituir toda la cadena por lo siguiente:

      LIBS = $(SUBLIBS) -L/usr/share/qt3/lib -L/usr/X11R6/lib ../libtwinkle.a ../parser/libsipparser.a ../sdp/libsdpparser.a ../sockets/libsocket.a ../threads/libthread.a ../audio/libaudio.a ../audio/gsm/libgsm.a ../audio/g723/libg723.a ../audits/libaudits.a ../stun/libstun.a -lresolv -lsndfile -L/usr/local/lib -lccext2 -L/usr/lib -lxml2 -lz -pthread -lccrtp1 -lccgnu2 -ldl -lrt -lkdecore -lkdeui -L/lib -lkabc -lasound -lqt-mt -lXext -lX11 -lm -lpthread

   4. Si no se quiere sustituir la cadena completa, se puede añadir a continuación de ../audio/gsm/libgsm.a lo siguiente: ../audio/g723/libg723.a
   5. Guardar el fichero
   6. Realizar la compilación del código fuente.

2. **Make**

   Si todo ha ido correctamente se estará en disposición de realizar la instalación del cliente twinkle compilado en el sistema.

3. Para ejecutar el siguiente comando es necesario obtener los permisos de root, puesto que van a instalarse las librerías en el sistema de archivos. A continuación se ejecuta:

   make install

4. Ahora es cuando se tiene el código fuente preparado para su ejecución.

Una vez que se han terminado de ejecutar los cuatro pasos anteriormente descritos se tiene el cliente totalmente configurado y listo para ser ejecutado. Durante el desarrollo de la aplicación se ha utilizado el entorno de desarrollo KDEVELOP, dicho entorno de desarrollo facilita la búsqueda de funciones, parámetros y permite un desarrollo cómodo para la interfaz gráfica, basada en QT.

Debido a que la interfaz gráfica de twinkle esta desarrollada usando QT se ha decidido realizar la implementación del código fuente en Kubuntu, que unifica el entorno de ventanas de KDE con ubuntu linux.

Si se han seguido todos y cada uno de los pasos que anteriormente han sido descritos, se estará en disposición de desarrollar nuevo código fuente para el cliente twinkle, y por tanto se podrá mejorar y añadir nuevas funcionalidades con el objetivo de alcanzar un cliente de VoIP con soporte completo para calidad de servicio (QoS).

6.2 Código fuente de los Algoritmos de Buffer de Playout.

6.2.2 Algoritmo exponencial.

A continuación se va a ver el código fuente que realiza el calculo para el retraso de playout usando el método exponencial.

```
/** Algoritmo Buffer de Playout Exponencial
* Es un algoritmo dinámico que debemos de calcular aprovechando los periodos de
silencios.
* Datos de los parametros
* @param silencio = es el periodo del silencio en ms
* @param delay = Es el retraso entre paquetes.
* @param PlayoutDelay = El tiempo que tiene que estar los paquetes en el Buffer ( el
paquete primero de una rafaga )
* @param ref_PlayoutDelay = varible auxiliar para calcular el valor de playout quitando el
periodo de silencio.
* @param Secuencia = es el numero de secuencia del paquete o trama
* @param Media y varianza son los valores para aplicar el algoritmo exponencial.
*/
int Playout_exponencial(unsigned int *silencio, float *delay, float *PlayoutDelay, float
*ref_PlayoutDelay,int secuencia,float *media, float *varianza)
{
 float antmedia;
 float antvarianza;
 float antPlayoutDelay;
 antmedia = *media;
 antvarianza = *varianza;
 antPlayoutDelay = *PlayoutDelay;

 /*INFO("[PlayoutExp] Valores de las variables \n","");
 INFO("[PlayoutExp] Valor de la antmedia %f \n", antmedia);
 INFO("[PlayoutExp] Valor de la antvarianza %f \n", antvarianza);
```

```
INFO("[PlayoutExp] Valor de la antPlayoutDelay %f \n", antPlayoutDelay);
INFO("[PlayoutExp] Valor de la ref_PlayoutDelay %f \n", *ref_PlayoutDelay);
INFO("[PlayoutExp] Valor del silencio %d \n",*silencio);*/



if((*silencio) != 0)
{
(*ref_PlayoutDelay) = (*PlayoutDelay);
if( (*ref_PlayoutDelay) < antPlayoutDelay - (*silencio))
                {
                        (*ref_PlayoutDelay) = antPlayoutDelay - (*silencio);
                }
                //*silencio = 0; lo he eliminado.
                //INFO("[PlayoutExp] Valor de referencia cuando hay silencios %d
\n",*ref_PlayoutDelay);
        }

        //INFO("[PlayoutExp] Delay:%f \n",*delay);
        if((*delay) > antmedia)
        {
                (*media) = (ALFA * antmedia) + ((1 - ALFA) * (*delay));
        }

        (*varianza) = (ALFA * antvarianza) + ((1 - ALFA) * fabs( (*media) - (*delay) ) );
        /*INFO("[PlayoutExp] ************************** \n","");
        INFO("[PlayoutExp] Valores aplicando algoritmo \n","");
        INFO("[PlayoutExp] Valor de la media %f \n", *media);
        INFO("[PlayoutExp] Valor de la varianza %f \n", *varianza);*/
        (*PlayoutDelay) = (*media) + (BETA * (*varianza));
        /*INFO("[PlayoutExp] PlayoutDelay: %f \n",*PlayoutDelay);*/
        return 0;
}
```

6.2.3 Algoritmo Spike.

A continuación se va a ver el código fuente que realiza el calculo para el retraso de playout usando el método control de Spike.

```
/**
 * Funtion: Playout_spike
 * Descripcion: Determina el playout que debemos de tener cuando las tramas nos llegan con retrasos en modo impulso.
 * @param silencio indica el periodo de silencio que se ha sucedido.
 * @param ldelay indica el inicio de la lista enlazada
 * @param PlayoutDelay El tiempo en ms dentro de cuál vamos a retrasar los paquetes antes de reproducirlos
 * @param ref_PlayoutDelay Este parametro es tomado como referencia para poder hacer algunos calculos, siempre se corresponde con el primero de la rafaga de voz
 * @param secuencia Como su nombre indica, almacena el número de secuencia actual.
 * @param media Nos permite calcular la media, que nos va permitir calcular el playoutdelay
 * @param varianza es una variable que nos permite calcular la varianza , y posteriormente el playoutdelay
 * @param var
 * @param mode nos indica en que modo de funcionamiento nos encontramos: NORMAL o IMPULSO
 */
int Playout_spike(unsigned int *silencio, NODO *ldelay, float *PlayoutDelay, float *ref_PlayoutDelay, int secuencia, float *media, float *varianza, float *var,int *mode)
{
        float delay;
        float antdelay;
        float antantdelay=0;
        float antmedia;
        float antvarianza;
        float antPlayoutDelay;
        int cambio;
```

```
//var=0; lo elimino pq no tiene sentido inicializar a cero en todo momento, y al
mismo tiempo pasarlo como referencia.
        cambio = 0;
        antmedia = *media;
        antvarianza = *varianza;
        antPlayoutDelay = *PlayoutDelay;
        delay = ldelay->delay;
        antdelay = ldelay->sig->delay;

        if((ldelay->sig->sig) != NULL){
                antantdelay = ldelay->sig->sig->delay;
        }
        if(*silencio != 0){
                *ref_PlayoutDelay = *PlayoutDelay;
                if(*ref_PlayoutDelay < antPlayoutDelay - *silencio){
                        *ref_PlayoutDelay = antPlayoutDelay - *silencio;
                }
                *silencio = 0;
        }
        if(*mode == NORMAL){
                if(fabs(delay-antdelay) > ((fabs(antvarianza)*2) + 800)){
                        *var = 0;
                        *mode = IMPULSE;
                }
        }
        else{
                *var = (*var/2) + fabs((2*(delay) - antdelay - antantdelay)/8);
                if(*var <= 63){
                        *mode = NORMAL;
                        cambio = 1;
                }
        }
        if(cambio == 0){
                if(*mode == NORMAL){
```

```
                *media = 0.125 * delay + 0.875 * antmedia;
            }
            else{
                *media = antmedia + delay - antdelay;
            }
            *varianza = 0.125 * fabs(delay - *media) + 0.875 * antvarianza;
        }
        cambio = 0;

        *PlayoutDelay = *media + (BETA * (*varianza));
        INFO("[PlayoutSpike] Algoritmo Spike ","");
        INFO("[PlayoutSpike] ************************** \n","");
        INFO("[PlayoutSpike] Valores aplicando algoritmo \n","");
        INFO("[PlayoutSpike] Valor de la media %f \n", *media);
        INFO("[PlayoutSpike] Valor de la varianza %f \n", *varianza);
        INFO("[PlayoutSpike] PlayoutDelay: %f \n",*PlayoutDelay);
        return 0;
}


//TODO: debo crear una lista cirular para poder manejar el valor que pide el algoritmo spike, de antantdelay. ( para ello necesito 3 NODOS)
// es decir, debemos de tener 3 NODOS en la lista circulares, de modo que cuando venga un nuevo paquete, en vez de crear uno se machaca el valor
// más viejo y se actualizan los valores de los punteros que dan acceso a los NODOS.
// Por modificar las funciones para que se adapten a la nueva situación, y contabilizar el número de NODOS creados para adaptar la lista a
// a la cantidad de nodos necesarios

/**
* Función que crea un NODO de la lista que contiene los valores de retrasos obtenidos por cada paquete.
*/
NODO *CreaNodo(float delay){
 NODO *n_prueba;
```

```
 n_prueba->delay = delay;
 n_prueba->sig = NULL;
 return n_prueba;
}

/**
* Función que añde los Nodos a la lista, enlazandolos con el principal.
*/
void Enlazaspike(NODO **lfirst, NODO *lnodo){
        if( lnodo != NULL ) {
                (*lfirst)->sig->sig = lnodo;
        }
}

/**
* Función que borra la lista de delay que tenemos almacenada.
*/
void Borra_Nodo(NODO **lnodo){
        (*lnodo) = NULL;
        delete lnodo;
}
```