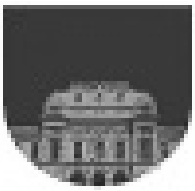


Universidad de la República
Facultad de Ingeniería
Instituto de Computación

Proyecto de Grado

Monitoreo y Estadísticas en una Red de Distribución de Multimedia

Andrés Chadarevian
Daniel De Vera
José Luis Fernández

Tutores:
Héctor Cancela
Pablo Rodríguez Bocca

Contenido

Resumen

En la actualidad la transmisión de contenidos multimedia en Internet ha cobrado sustantiva importancia debido a las necesidades de las personas de comunicarse y poder disfrutar de diferentes espectáculos que suceden en cualquier parte del mundo en el mismo instante o revivir eventos que ya sucedieron.

Entre muchas aplicaciones de la transmisión de audio y video por Internet se encuentran las videoconferencias que permiten que diferentes personas se puedan comunicar como si estuviesen reunidos, independientemente de la ubicación física de las mismas.

También es muy útil para la difusión masiva de espectáculos, por ejemplo los eventos deportivos como ser los campeonatos de fútbol. Además contamos con la posibilidad de ver videos, como ser avances de películas en el momento que deseemos (a demanda).

Todas estas aplicaciones tienen como objetivo el receptar audio y video y que el servidor encargado de su difusión lo transmita a el o los clientes. A partir de que los requerimientos de calidad del streaming cada vez son mayores, es vital contar con datos de la percepción del cliente con respecto a la transmisión del video.

Existen algunas soluciones para el problema planteado pero todas son propietarias.

Nuestra solución brinda información sobre indicadores de los que se puede extraer la calidad en la reproducción del contenido multimedia en el cliente. Para esto se toman medidas de esos indicadores en clientes del reproductor que seleccionamos, los cuales deben tener aplicado nuestro patch. Estos son capaces de loguear dicha información en forma muy detallada y de responder mediante el protocolo SNMP esta información de forma condensada.

Además se agrega una aplicación que posee la funcionalidad de visualización de información que se brinda por SNMP con las cantidades registradas y en forma de gráficos.

De esta forma se completa el ciclo de uso de esta solución, con el servidor que trasmite el streaming, los clientes que lo consumen y capturan los indicadores y otra aplicación que consume esos indicadores y los despliega para que puedan ser clara y sencillamente visualizados.

Palabras claves
QoS Quality of Service, VLC, VideoLan, Streaming, Multimedia, VDN Video Delivery Network.

1. Introducción

Podemos observar el notorio aumento de las aplicaciones de transmisión de audio y video por internet, lo cual conlleva a un mayor uso de las mismas y aumenta los requerimientos de calidad de dichos servicios. Estos requerimientos pueden estar favorecidos por algunas condiciones de naturaleza fisica, de red y transporte, pero bajo buenas condiciones puede haber insuficiencias de calidad percibidas por el cliente.

Nuestro principal objetivo es realizar una medición desde un cliente y poder registrar dicha información y transmitirla. Para poder realizar mediciones hay que contar con un cliente de streaming de audio y video. Existen muchos clientes, pero nos decidimos por VLC básicamente porque es un software libre, porque en el mismo software contamos con servidor y cliente y por la amplia gama de formatos soportados, lo que en comparación con otros software's de la industria lo evaluamos superior (ver [Ref18]).

En nuestro trabajo nos enfocamos en el streaming de video, pero cabe destacar que generalmente está acompañado de streaming de audio y ocasionalmente de subtítulos. Antes de comenzar a entender realmente el problema hay que contar con una base teórica. A continuación describimos algunos conceptos los cuales son ampliados y detallados en el capítulo 2 (Marco teórico).

Streaming es la técnica de transferir datos en un flujo continuo. En el caso de los archivos de contenidos media, como ser audio y video, permite la transmisión en vivo de los mismos. Antes de la aparición del mismo, las diferentes aplicaciones multimedia usaban Internet únicamente para realizar transferencias de archivos a PCs y una vez que estos eran completamente descargados los mismos podían ser ejecutados. Actualmente es posible visualizar un contenido multimedia a medida que éste es transferido, es decir los contenidos son presentados en la misma tasa en que son transferidos, por lo que un video de 10 minutos demorará 10 minutos en ser descargado mediante la red. A esta red conformada por los servidores que transmiten video, audio y otros datos y los clientes que lo consumen es llamado video delivery network (VDN).

Dentro de la arquitectura del Streaming existen 4 componentes básicos los cuales deben existir para que se lleve a cabo el proceso de Streaming, debe existir una etapa de captura, codificación y compresión del audio/video, un servidor el cual provea los contenidos, un canal de distribución el cual conecte al cliente con el servidor y finalmente un reproductor multimedia el cual decodifique y despliegue el audio/video.

En el proceso de captura y codificación, como se puede observar en la Figura 1, existen 2 diferentes etapas, la primera refiere a la compresión de los datos mientras que la segunda refiere a la paquetización de los datos anteriormente comprimidos. Para estas etapas existen una numerosa cantidad de estándares y RFCs definidos. A su vez también existen diferentes protocolos mediante los cuales se pueden transmitir estos datos a través de la red.

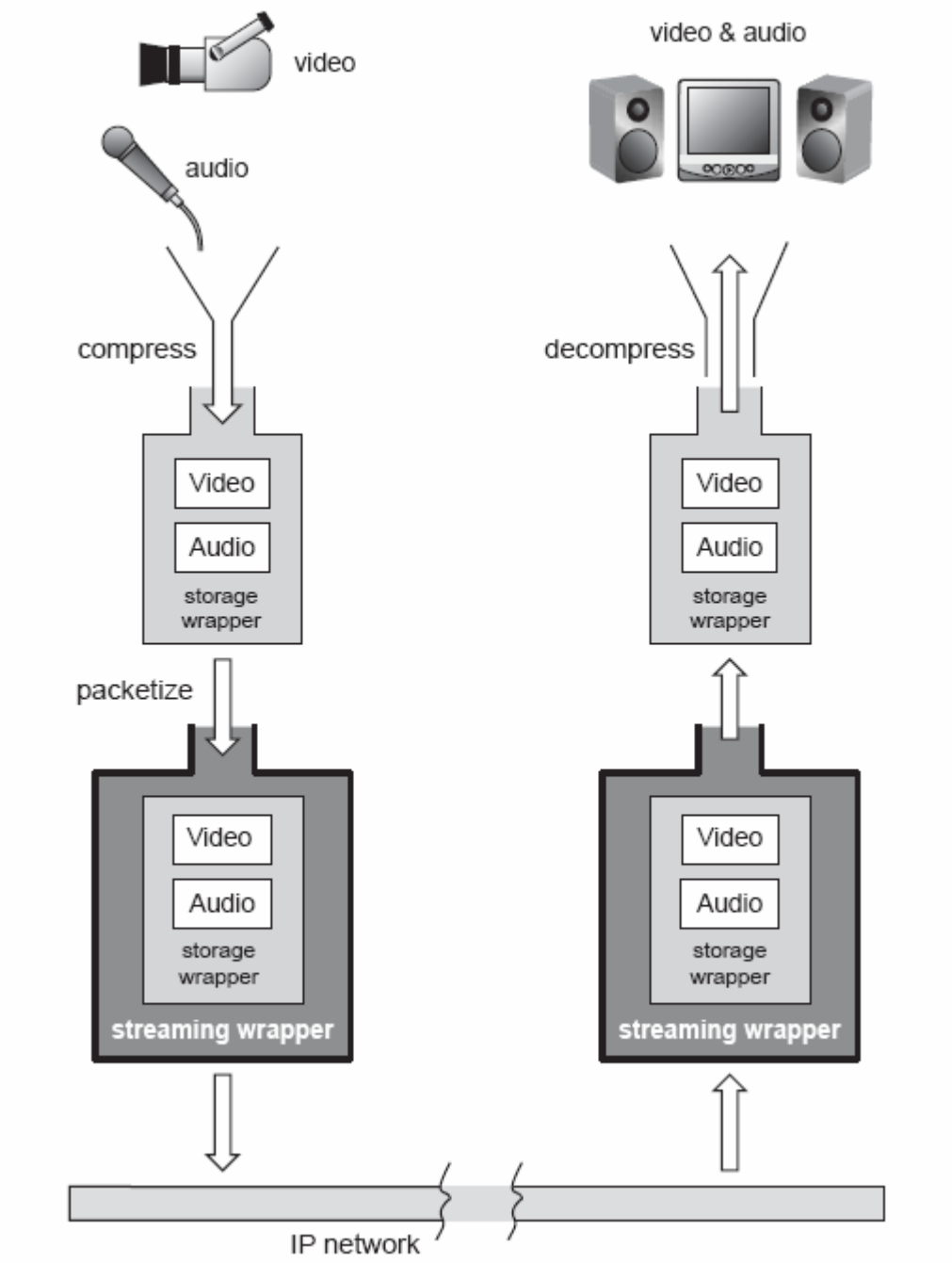


Figura 1: Compresión y descompresión

La parte izquierda de la Figura 1 corresponde con lo que sucede del lado del servidor y respectivamente del lado derecho en el cliente. Nosotros nos concertaremos en el cliente (sin dejar de lado el servidor) que es donde queremos detectar la calidad del streaming.

Puntualmente el problema surge en la complejidad de detección de los problemas de calidad percibidas por el cliente. Lograr detectar que el cliente está recibiendo la información de forma inadecuada no es para nada tarea fácil. Como se presenta en [Ref19] las aplicaciones web clásicas no presentan estas dificultades, ya que en estas la calidad se limitaba a minimizar los tiempos de respuesta. En este documento se hace énfasis en la complejidad de realizar estas medidas.
El gran desafío es escoger indicadores adecuados que a partir de ellos se pueda extraer la calidad del servicio que se le brinda al cliente. Además luego de escogidos estos indicadores se debe saber como medirlos, lo cual parece obvio mencionarlo, pero no lo es a la hora de su implementación. En otras palabras se tomaran medidas sobre ciertas variables las cuales nos guiaran de cuan bien está reproduciendo el cliente. Luego de tener los resultados de cada indicador, debemos ser capaces de la transmisión de esos datos para ser procesados por un servidor central de estadísticas.

El escenario planteado consta de tres roles o grupos de actores (ver Figura 2): los **servidores de streaming**, los cuales transmiten el contenido multimedia; **los clientes** que consumen este flujo de video, los cuales perciben la calidad del mismo y son capaces de tomar las mediciones de los indicadores seleccionados y su posterior registro y transmisión; y finalmente **servidores de procesamiento** de la información de calidad, los cuales consultan la información de calidad recogida a los clientes mediante el protocolo SNMP.

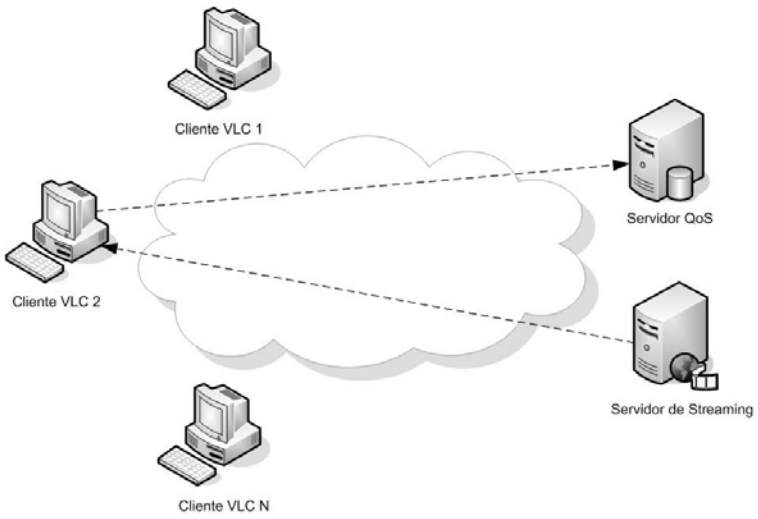


Figura 2: Escenario del Problema

La solución que planteamos consta de tres partes (ver Figura 3), en primer lugar tenemos en el cliente nuevos módulos y cambios en el código existente para la captura de los datos que consideramos relevantes y están disponibles. Como segunda parte tenemos el registro de los datos obtenidos mediante un log en el cliente. Y finalmente sobre estos datos se extraen condensaciones de los mismos y se brindan para su transmisión mediante el protocolo SNMP.

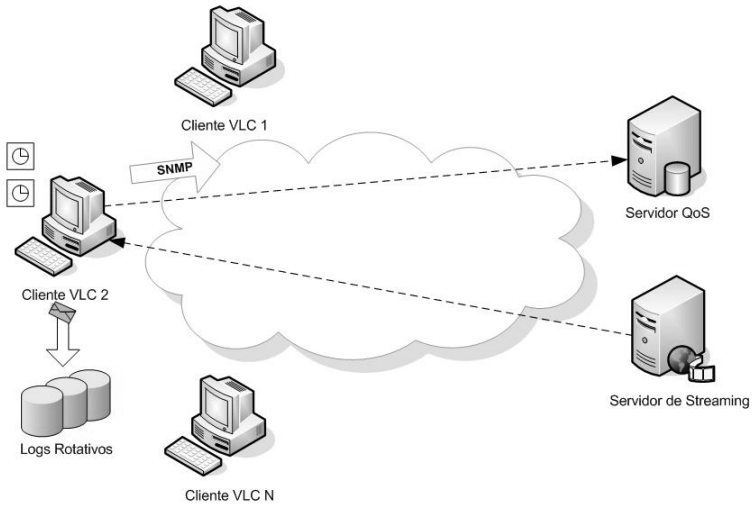


Figura 3: Registro y Transmisión de Datos

La arquitectura del reproductor (y servidor) elegido, VLC es modularizada (ver ANEXO I

Por este motivo hemos realizado nuestro agregado de funcionalidad de la misma forma. De este modo, como se puede observar en la Figura 4 desde el cliente y desde módulos extras incluidos por nosotros, pertenecientes al reproductor (sobre el que realizamos la mayor parte del desarrollo), se recogen los datos y se transmiten al núcleo del sistema. Este consta de 3 módulos, uno principal que recepciona la información recogida, y luego este trasmite esta información según la modalidad en la que se haya ejecutado el reproductor al módulo que registra en log esta información, al que transmite condensaciones de esta información vía SNMP o a ambos. Por más detalle de la solución referirse a capítulo 3).

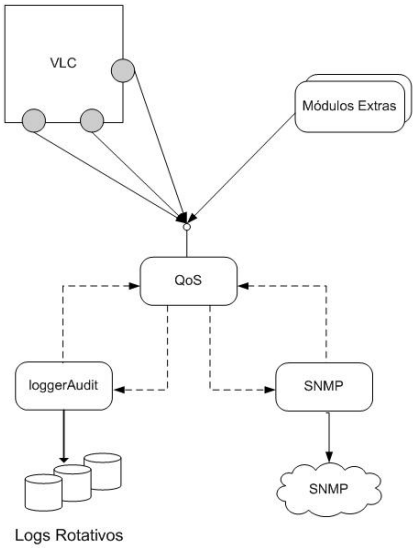


Figura 4: Arquitectura Modularizada para la Solución Propuesta

Además implementamos una aplicación web que brinda la funcionalidad de servidor de estadísticas. Este consume la información transmitida vía SNMP y la reporta en gráficas la cual es actualizada cada cierto tiempo. Por más información sobre esta aplicación ver ANEXO VII

Realizamos pruebas de correctitud para corroborar los resultados arrojados por las mediciones del sistema las cuales tuvieron resultados favorables (sección 4.2). Además contamos con pruebas complejas en las que apuntamos a probar el sistema en circunstancias más cercanas a un caso real (sección 4.3).

2. Marco teórico

En este capítulo vemos conceptos que son de utilidad para el entendimiento de los demás capítulos. Tratamos la transmisión de video por Internet para comprender métodos y protocolos utilizados para la transmisión del video desde el servidor hasta el cliente (sección 2.1). También se tratan específicamente los protocolos de streaming (sección 2.2) y los protocolos de Monitoreo de Red (sección 2.3). Finalmente damos una reseña de los servidores y clientes que se presentan actualmente en la industria (sección 2.4).

2.1. Transmisión de video por Internet

2.1.1. Introducción

En esta sección estudiamos conceptos básicos de lo utilizado para la transmisión de audio y video por Internet. Esto nos aporta para el entendimiento del problema y la solución adoptada.

2.1.2. Métodos de Distribución

2.1.2.1. En vivo – A demanda

La transmisión de audio y video a través de una red se puede realizar en vivo (live), o sea que en el mismo instante en el que se está recibiendo por parte del servidor la imagen y el sonido se transmite. La otra forma es a demanda (VoD – Video on Demand) por lo que el archivo multimedia debe estar pre-grabado y almacenado. En esta última modalidad el cliente tiene la posibilidad de comenzar la transmisión desde donde quiera, e incluso volver hacia atrás la transmisión.

2.1.2.2. Unicast – Multicast

Por otro lado la distribución se puede realizar desde un servidor hacia cada cliente que lo solicite (unicast) o se puede que un mismo servidor distribuya lo mismo para un grupo de clientes (multicast) lo cual podemos observar en la Figura 5. Esto realiza un mejor uso de la red ya que la cantidad de clientes no afecta la distribución del servidor, es lo mismo distribuir a 1 que a infinitos clientes. En el caso de a demanda este sistema de distribución no tiene sentido ya que cada cliente puede querer ver partes diferentes. Sin embargo el método de distribución unicast tiene la desventaja que en caso de las transmisiones en vivo por cada cliente el consumo de recursos es acumulativo.

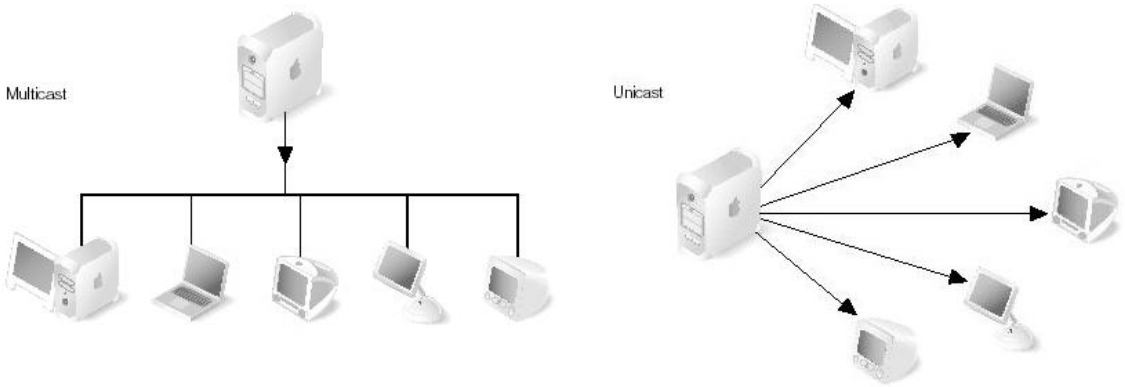


Figura 5: Distribución Unicast - Multicast

2.1.3. Protocolos de Red y Transporte

En lo que se refiere a transmisión están las capas más básicas, que son las capas físicas, correspondientes a las capas 1 y 2 del modelo OSI de ISO. Ejemplos de protocolos correspondientes la capa 1 son Cable coaxial, Cable de fibra óptica, Cable de par trenzado, Microondas, Radio, RS-232; y de la capa 2 son Ethernet, Fast Ethernet, Gigabit Ethernet, Token Ring, FDDI, ATM, HDCL.

En la capa de red (capa 3) se encuentran los protocolos ARP, RARP, IP, X.25, ICMP, IGMP, NetBEUI. En nuestro trabajo de investigación nos enfocamos particularmente en el protocolo IP, ya que es el que aplica directamente. Los otros protocolos de esta capa son mayormente para control y sincronización de red, los que si bien están relacionados no son sobre los cuales tomaremos mediciones y sobre los cuales se realiza la transmisión de streaming. Nos hemos basado en [Ref2] y [Ref4] para el desarrollo de esta sección.

2.1.3.1. Internet Protocol (IP)

Protocolo no orientado a conexión para la comunicación a través de una red de paquetes conmutados. Definido en el RFC 791 (ver siglas y abreviaturas). Estos paquetes son enviados con información extra para poder realizar seguridad mediante checksums de su header, pero no proporciona ningún tipo de seguridad de que los datos realmente lleguen a destino (de esto se deben encargar los protocolos de capas superiores). El header de los paquetes IP son compuestos por 5 palabras de 32 bits (20 bytes) y luego opciones de hasta 5 palabras más (opcionales). Como presentan en [Ref5] siguen el siguiente formato:

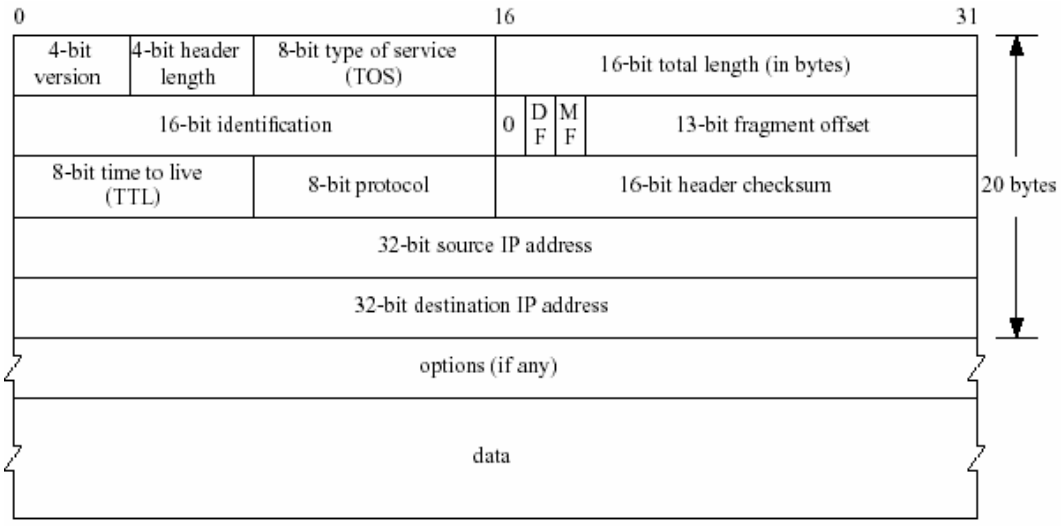


Figura 6: Paquete IP

La capa 4 es la encargada del transporte, es de su responsabilidad la transferencia libre de errores de datos entre el emisor y el receptor. Se brinda a continuación un detalle de los protocolos correspondientes a esta capa:

2.1.3.2. Transport Control Protocol (TCP)

Protocolo que nos asegura que los paquetes lleguen y lo hagan en orden. Definido en el RFC 793 (ver siglas y abreviaturas). Esto es conseguido mediante un mecanismo en el cual, cuando no se recibe por parte del destinatario la confirmación del paquete durante cierto tiempo se reenvía. Debido a este comportamiento es posible detectar paquetes perdidos y pedir retrasmisión de los mismos.
Los paquetes de este protocolo van dentro de los paquetes del protocolo de red (IP) y contienen la siguiente información:

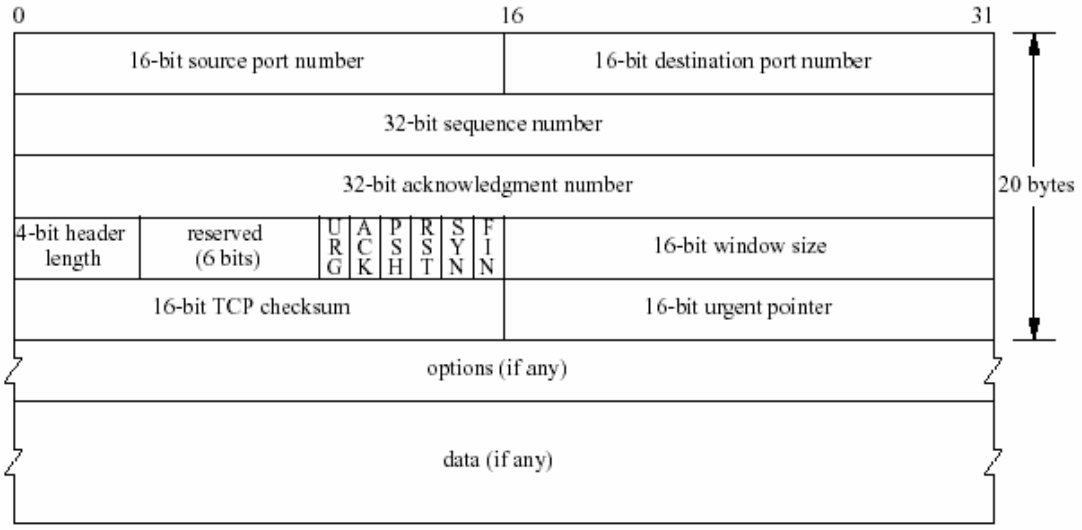


Figura 7: Paquete TCP

2.1.3.3. User Datagram Protocol (UDP)

Protocolo que provee un servicio "best-effort" de envío de datagramas. Definido en el RFC 768 (ver siglas y abreviaturas). No provee garantía sobre pérdida de paquetes ni paquetes duplicados. Igualmente cuenta con mecanismos de chequeo de datos. Los manejos de errores en la transmisión de paquetes se deben realizar por parte de las aplicaciones que usan el mismo.

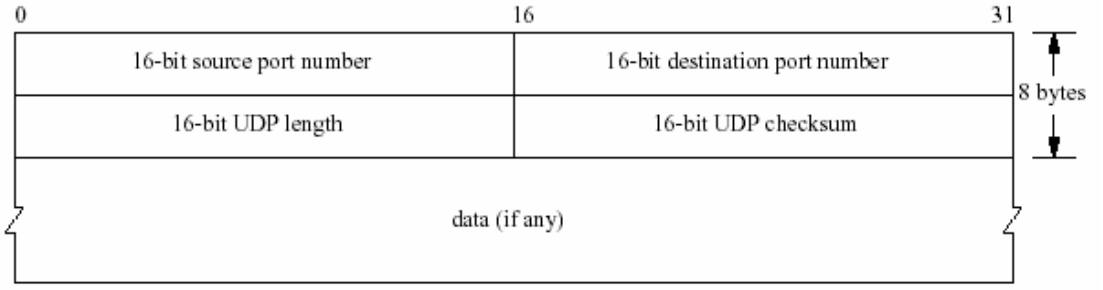


Figura 8: Paquete UDP

2.1.4. Protocolos de Tiempo Real

2.1.4.1. Real-time Transport Protocol (RTP)

Es un protocolo de transporte desarrollado para streaming. Esta definido en el RFC 1889 (ver siglas y abreviaturas). Este incluye datos extras no presentes en TCP, como timestamp y número de secuencia lo que contribuye a transporte en forma continua. También hay datos de control que permite al servidor realizar el streaming a una tasa correcta.

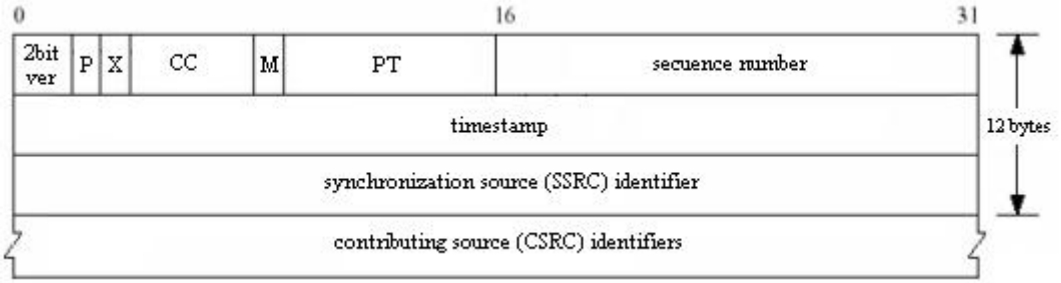


Figura 9: RTP Header

Aquí tenemos la estructura del header de RTP donde se muestra en este protocolo la información disponible. Los campos que contiene son los siguientes:

- **CSRC count (CC): 4 bits**

Cantidad de identificadores CSRC que siguen al header fijo.

- **Sequence number (número de secuencia)**

Número sucesivo por paquete, de 16 bits, usado por el reproductor para detectar paquetes perdidos y reordenarlos. El número inicial se elige al azar.

- **Timestamp**

Instancia derivada de un reloj de referencia que permite la sincronización y el cálculo de jitter. Es monótono y lineal con el tiempo.

- **Synchronization Source Identifier SSRC**

Es el identificador único para la sincronización del stream RTP. Es elegido de forma aleatoria. Pueden existir uno o más CSRC cuando el stream RTP contiene información de múltiples orígenes.

- **Contributing Source Identifier CSRC**

Contiene los identificadores de los orígenes de contribución para el payload contenido en el paquete.

2.1.4.2. Real-Time Control Protocol (RTCP)

Es usado en conjunción con RTP para la recepción de reportes de estadísticas. Permite la detección de fallas en el árbol de distribución de multicast, por ejemplo número de paquetes perdidos y estadísticas de jitter.

Por ejemplo los tipos de reportes del emisor (SR) son acumulativos de frames o cantidad de bytes y de reportes de receptor Frames perdidos/Tasa de Frames entregados.

Una característica particular que tiene este protocolo es que para amortizar el overhead del header se pueden reunir varios mensajes RTCP y ser enviados en un mensaje RTCP compuesto. Los paquetes deben estar compuestos al menos por un mensaje de receptor o emisor y el nombre canónico del participante (CNAME) y deben ser enviados periódicamente sin llegar a consumir el 5% del ancho de banda de la sesión.

Los paquetes RTCP se transportan sobre datagramas UDP, o sea que el paquete queda conformado de la siguiente forma:

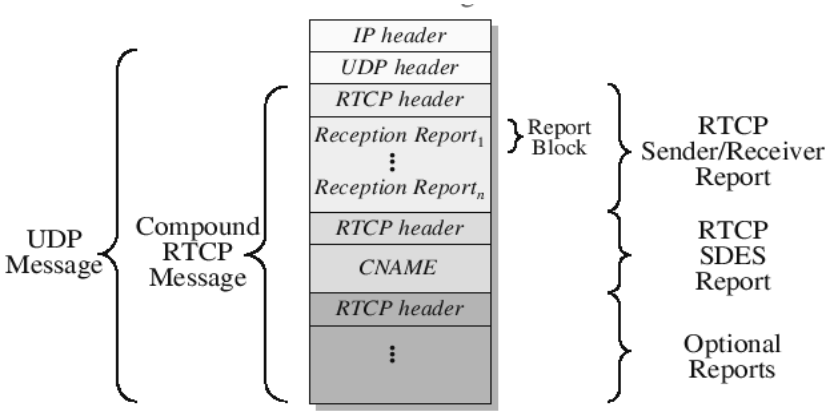


Figura 10: Paquete RTCP en paquete UDP

El header RTCP depende según sean un paquete del receptor o del emisor:

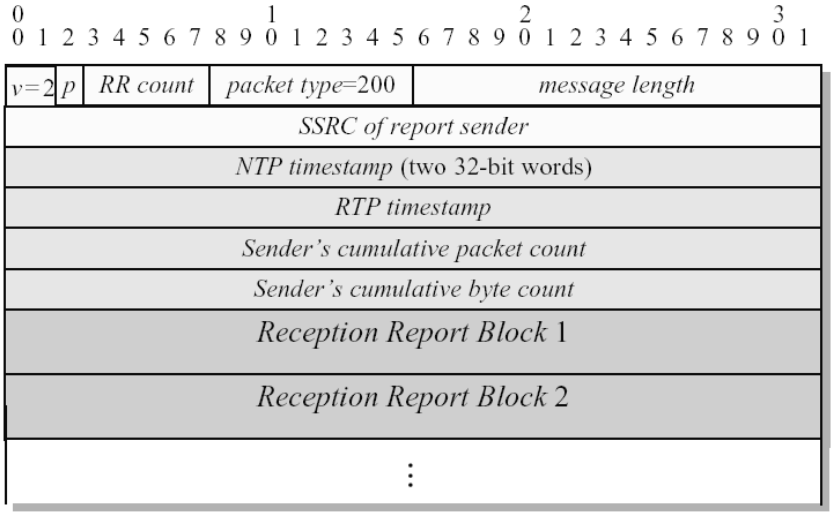


Figura 11: Header RTCP del emisor

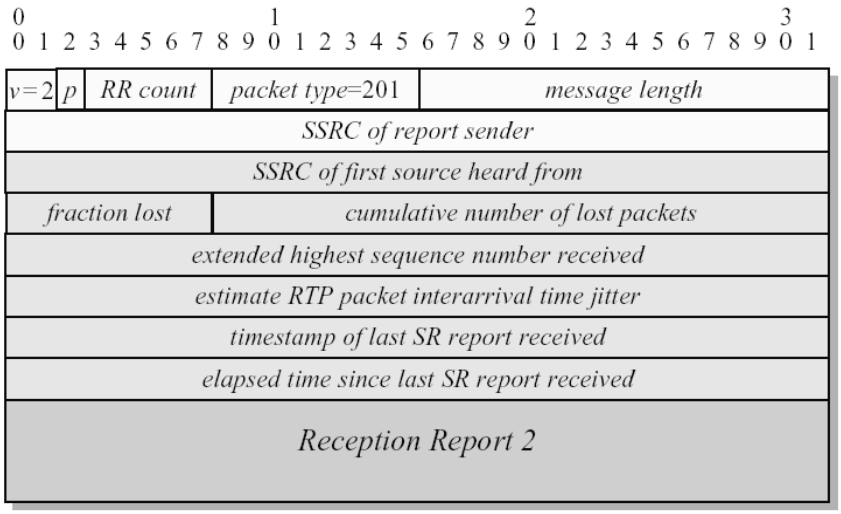


Figura 12: Header RTCP del receptor

2.1.4.3. Real Time Streaming Protocol (RTSP)

Realiza control sobre datos multimedia de tiempo real. Brinda la posibilidad de interactividad con el reproductor, como el reproducir, el pausado y más. También puede reaccionar a congestiones en la red y reducir el ancho de banda. Este fue desarrollado similar a HTTP 1.1 pero con las diferencias que tiene es que ambos (cliente y servidor) pueden realizar pedidos y que la conexión posee estados. RTSP soporta RTP como protocolo de transporte. Su objetivo es brindar la forma escoger el canal de distribución óptimo hacia el cliente. Por ejemplo algunos clientes pueden tener filtrados en su firewall los paquetes UDP por lo que el servidor de streaming debería proveer la posibilidad de escoger entre diferentes protocolos de streaming basados en diferentes protocolos de transporte como ser UDP, UDP multicast, TCP.

2.1.4.4. Microsoft Media Server protocol (MMS)

Protocolo de capa de aplicación propietario de Microsoft para streaming. Opera sobre TCP, UDP y HTTP, esto lo selecciona el servidor según el estado de la red. En caso de que se pueda tratará con HTTP que controla duplicaciones y pérdidas de paquetes, pero si la red es muy lenta para el overhead que este protocolo representa, lo cambia por TCP y en el mismo caso lo cambia a UDP.
El header del paquete MMS (según [Ref9]) sigue el siguiente formato:

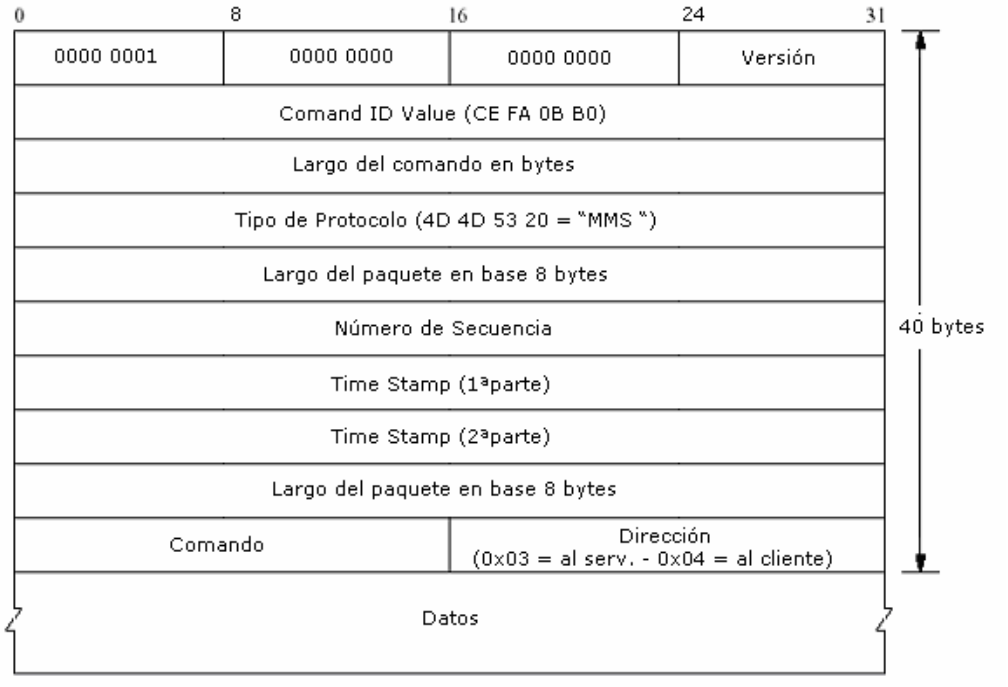


Figura 13: Paquete MMS

2.2. Protocolos de Streaming

En esta sección se estudia los protocolos de streaming sobre los cuales se trabajó con profundidad en el proyecto.

2.2.1. MPEG-2

Estándar de formato de compresión creado en 1994 por el grupo MPEG, evolucionado del estándar MPEG-1 y manteniendo compatibilidad hacia atrás. En este se soporta la reproducción de video entrelazado y a nivel de audio se permite más de dos canales por programa (hasta 6), lo cual sólo se soportaba 2 canales maestros para la reproducción estéreo. Por otra parte una desventaja de este estándar es que su rendimiento no es bueno para tasas de transferencias bajas (menores a 1Mbit/s).

Este formato es utilizado para la codificación de audio y video para señales de transmisión, incluyendo Satélite Digital y Cable TV. También es utilizado con algunas modificaciones para los discos SVCD y DVD comerciales de películas.

La codificación MPEG-2 es un esquema híbrido de compresión para imágenes que usa codificación inter-trama y codificación intra-trama y combina la codificación predictiva con la codificación con la transformada coseno discreta (DCT). Esta transformada es un algoritmo matemático que convierte del dominio del tiempo al dominio de la frecuencia y que es aplicado típicamente sobre los bloques de 8x8 de la imagen. La DCT elimina redundancia en la imagen a través de la compresión de la información contenida en 64 píxeles. El cuantizador otorga los bits para los coeficientes DCT más importantes, los cuales son transmitidos.

Maneja la compresión de video a nivel de GOP y para esto define 3 tipos diferentes de imágenes (también llamados frames), los frames I, los frames B y los frames P.

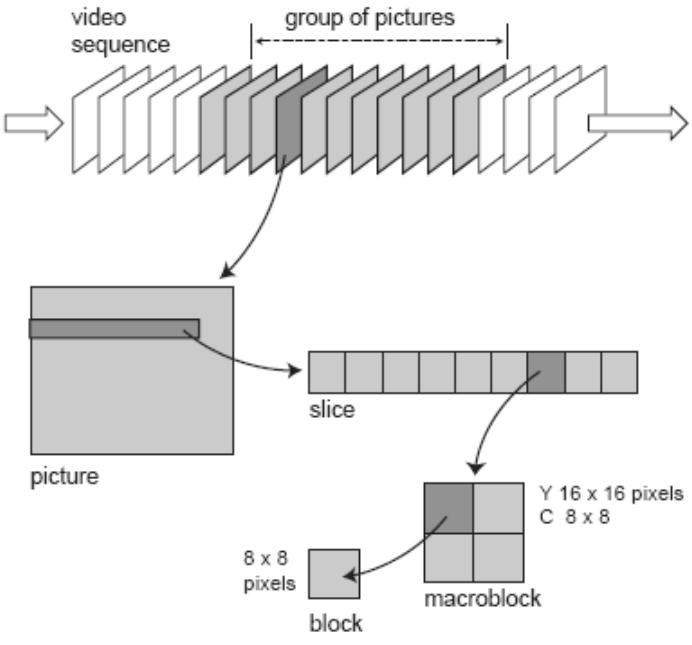


Figura 14: Construcción de un VES

Como podemos observar en la Figura 14, una secuencia de video se compone de una sucesión de imágenes (pictures) las cuales se dividen en slice's. Cada slice se divide en macroblocks los cuales son compuestos por block. Los block's pueden ser de tres tipos: luminancia (Y), crominancia roja (Cr), crominancia azul (Cb). Los macroblock generalmente en los perfiles más simples son compuestos por 4 blocks de luminancia (Y) y 2 de cromancia, uno azul (Cb) y otro rojo (Cr).

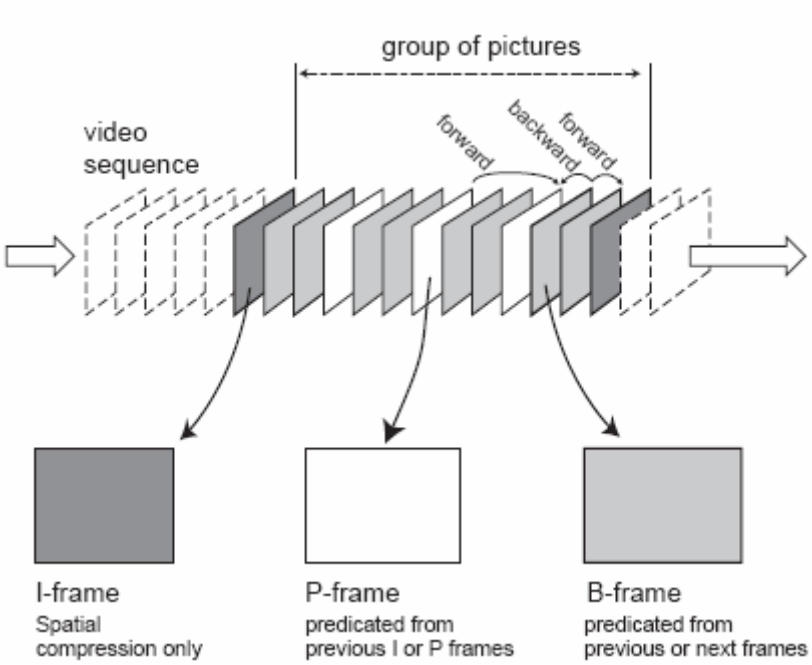


Figura 15: Frames I, P, B

Frame I – intra codificado

Estas se codifican como si fuesen imágenes fijas utilizando la norma JPEG, por tanto, para decodificar una imagen de este tipo no hacen falta otras imágenes de la secuencia, sino sólo ella misma.

Frame P – predecible posterior

Estas están codificadas como predicción de de la imagen I ó P anterior usando un mecanismo de compensación de movimiento. Para decodificar una imagen de este tipo se necesita, además de ella misma, la I ó P anterior

Frame B – predecible bidireccional

Estas se codifican utilizando la I ó P anterior y la I ó P siguiente como referencia para la compensación y estimación de movimiento. Para decodificarlas hacen falta, además de ellas mismas, la I ó P anterior y la I ó P siguiente. Estas imágenes consiguen los niveles de compresión más elevados y por tanto son las más pequeñas.

Como se puede observar en la Figura 15: Frames I, P, B, dentro de un GOP siempre se mantiene la misma secuencia de imágenes, es decir en nuestro ejemplo, primero se codifica una imagen de tipo I, luego 2 imágenes B, luego una imagen P, etc. El largo de la secuencia de imágenes dentro de las GOP puede variar de un stream a otro, en generan se encuentra dentro del rango de 12 a 15 imágenes por GOP.

De todas las combinaciones posibles de configuraciones, se definen un reducido conjunto de combinaciones bajo "perfiles" y "niveles". Definiendo la combinación del perfil y el nivel se tiene bien definida la arquitectura para una cadena particular de bits. Los perfiles limitan la sintaxis (por ejemplo los algoritmos), mientras los niveles limitan los parámetros (velocidad de muestreo, dimensiones de las tramas, velocidad binaria codificada, etc.).

A continuación explicamos los niveles y perfiles predefinidos.

Niveles: proveen un rango de cualidades potenciales, definen los máximos y mínimos para la resolución de la imagen, muestras Y por segundo (luminancia), el número de capas de audio y vídeo soportados por los perfiles escalados, y la máxima velocidad binaria por perfil. A continuación una explicación resumida de cada uno de ellos:

- Nivel Bajo*:* tiene un formato de entrada el cual es un cuarto de la imagen definida en el registro ITU-R 601.
- Nivel Principal: tiene una trama de entrada completa definida en el registro ITU-R 601.
- Nivel Alto 1440: tiene un formato de alta definición con 1440 muestras por línea.
- Nivel Alto: tiene un formato de alta definición con 1920 muestras por línea (para aplicaciones sin cualquier limitación en velocidades de datos).

Perfiles: son definidos subconjuntos con características de sintaxis (por ejemplo: algoritmos), usados para converger la información. Hay cinco diferentes perfiles y cada uno es progresivamente más sofisticado y agrega herramientas adicionales (y por supuesto más costoso para el cliente) con la característica adicional de ser compatible con el anterior. Esto significa que un decodificador equipado con un alto perfil decodificará perfiles simples. A continuación una pequeña explicación de los perfiles:

- Perfil Simple: es el que ofrece pocas herramientas.
- Perfil Principal: tiene herramientas extendidas o mejoradas del perfil simple y predicción bidireccional. Tendrá mejor calidad para la misma velocidad binaria que el perfil simple.
- Perfil Escalable SNR y Perfil Escalable Espacial: son los próximos pasos. Estos dos niveles son llamados escalables porque ellos permitirán codificar datos de video que sean particionados dentro de una capa base y una o más señales "Top-up". La señal Top-up puede tanto tratar la proporción S/N (SNR escalable) o la resolución (escalable espacial).
- Perfil Alto: este incluye todas las herramientas de las versiones anteriores y mejoradas. Tiene la habilidad de codificar diferencias de color entre líneas simultáneamente. Este es un super sistema diseñado para aplicaciones donde no están contraídas sobre las velocidades de los bits.

2.2.2. MPEG-4

Introducido en 1998, se agrega a las herramientas provistas en MPEG-2 otras que posibilitan que el contenido multimedia sea más reusable y con más flexibilidad brindando aún mejores prestaciones que la televisión digital, gráficos animados, páginas de Internet y sus extensiones. Además permite un mejor manejo y protección de los derechos de autor.

Algunas de las aplicaciones de este estándar son:

- Televisión digital
- Multimedia móvil
- Producción de TV
- Streaming video
- Juegos y Realidad Virtual

En esta versión del estándar se logra unificación de los tipos de contenido multimedia en un concepto de "media objeto", el cual permite la representación de escenas tanto en dos dimensiones como en tres dimensiones, incluyendo la componente de audio que corresponde a cada emisor y fondos tanto visuales como sonoros.

Se describe en el estándar ISO/IEC 14496 constituido por 21 partes. En estas se definen diferentes características ahora contempladas en esta nueva versión.

Algunas de estas son:

- Parte 6: Delivery Multimedia Integration Framework (DMIF), en la que se contempla el manejo de archivos compuestos en orientación a objetos (incluyendo objetos audio, video y Virtual Reality Modeling Language)

- Parte 13: Extensiones para el manejo y protección de Propiedad Intelectual (IPMP).

- Parte 16: Animation Framework eXtension (AFX).

- Part 17: Formato de subtítulos.

Según datos de [Ref27] la comparación entre el codificado de imagen de MPEG-2 y MPEG-4 según sus características se resume en la Tabla 1.

	MPEG-2	MPEG-4 H.264
I, P, B frames	Si	Si
Coding	Huffmann	Huffman o aritmética
Block size	16 x 16	down to 4 x 4
Quarter pixel resolution		Si
Deblocking filter		Si
Sliced-based motion prediction		Si
Multiple reference frames		Si
Weighted Prediction		Si
Switching pictures		Si
Bit rate comparison	100%	40%

Tabla 1: Comparación codificación MPEG2 vs. MPEG4

La estructura definida para este estándar se generaliza más que la propuesta en MPEG-2. Como se puede observar en la Figura 16 aquí no hay GOP's, se definen una agrupación parecida que se denomina GOV (Group of VOP's, o sea un grupo de Objetos de Planos de Video). En este caso generalmente no se mantienen una secuencia predefinida de tipos de frames (en este caso los VOP's), sino que la cantidad de VOP's dentro de un GOV es variable.

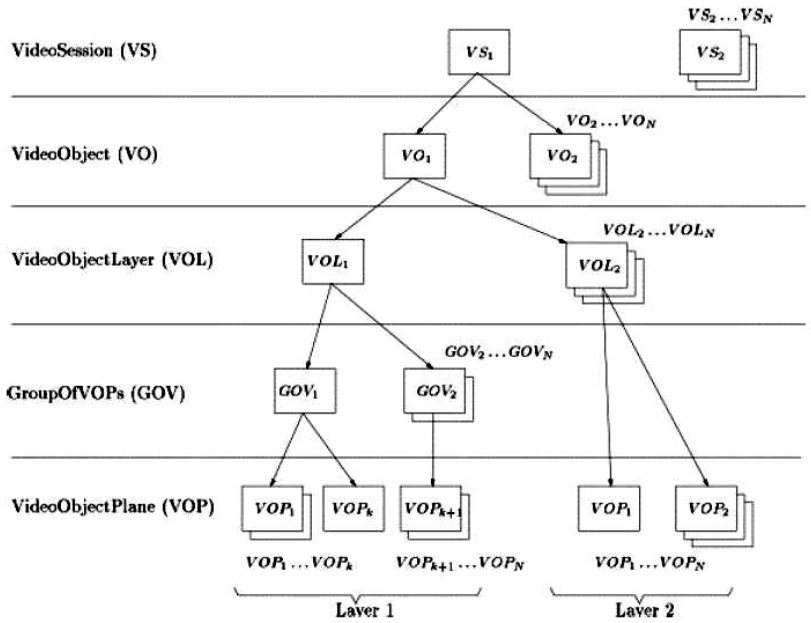


Figura 16: Estructura MPEG 4

2.2.3. MPEG Transport Stream

El Moving Picture Experts Group o mejor conocido como MPEG, ha desarrollado un conjunto de estándares para la compresión y transmisión de señales de audio y video. En particular el MPEG TS es un protocolo que nos brinda un mecanismo para multiplexar (combinar) los stream de audio y video, y así transmitirlos por la red.

Antes de profundizar en éste protocolo, veremos ciertos conceptos manejados por los diferentes estándares MPEG, los cuales nos son indispensables para la comprensión del mismo.

2.2.3.1. Elementary Streams

Un ES es básicamente la salida del encoder, y este contiene toda la información necesaria para que un decoder puede crear una aproximación lo más certera posible, ya sea del audio como del video. Existen 2 tipos diferentes de ES, los de audio y los de video, a continuación se describen cada uno de ellos.

Video Elementary Streams

Un VES esta compuesto por la salida del encoder de video, la cual se define como una secuencia de video. En esta se identifican grupos de imágenes (GOP's) los que describimos en el estándar MPEG-2.

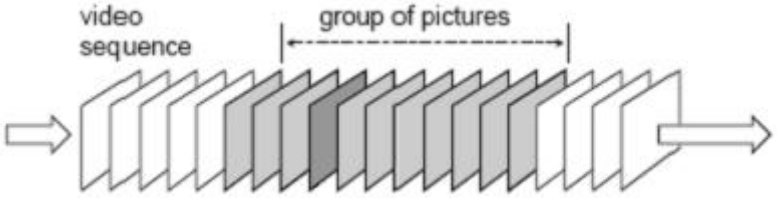


Figura 17: Secuencia de Video

Audio Elementary Streams

Un AES es la salida del encoder de audio, este caso es muy diferente al de compresión video, aquí no existen diferentes tipos de frames, los frames de audio son todos codificados de la misma manera, por lo que todos tienen el mismo tamaño.

2.2.3.2. Packetized Elementary Streams

Para lograr una mayor practicidad en el manejo de los diferentes ESs, estos son divididos en paquetes de diferente tamaño, según la conveniencia de la aplicación la cual va a decodificar estos streams.

A cada paquete PES se le agrega un header con el formato mostrado en la Figura 18, el cual contiene un campo de 24 bits denotando el inicio del paquetes (el mismo tiene el siguiente valor: 0000 0000 0000 0000 0000 0001) , un identificar de stream de 8 bits el cual identifica el tipo de ES con el cual se corresponde el paquete PES (por ejemplo “110x xxxx” corresponde con MPEG-1 o MPEG-2 audio stream número “x xxxx” y “1110 xxxx” se corresponde con MPEG-1 o MPEG-2 video stream número “xxxx”), el largo del paquete PES, para el cual se le asignan 16 bits y finalmente un conjunto de campos opciones entre los que se encuentran el Presentation Time Stamp (PTS) y el Decode Time Stamp (DTS) usados para lograr sincronización a la hora de decodificar y presentar el audio y video.

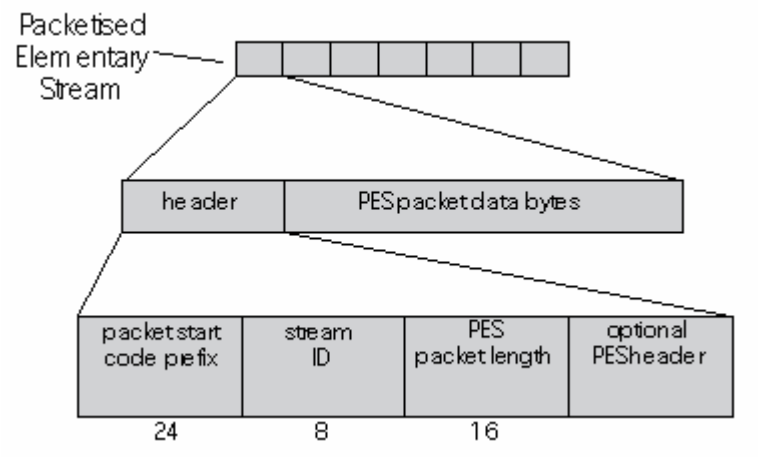


Figura 18: PES Header

2.2.3.3. Detalle del Transport Stream

El MPEG TS, no brinda simplemente una forma adecuada de realizar la multiplexación de los diferentes ES, sino que también ataca el problema de recrear el reloj del servidor en cada uno de los clientes, para lograr así una correcta descodificación y presentación del audio y del video.

Los paquetes TS tienen un largo fijo de 188 Bytes y el formato del mismo se puede observar en la Figura 19, a continuación se describen los campos más relevantes dentro del Header de estos paquetes:

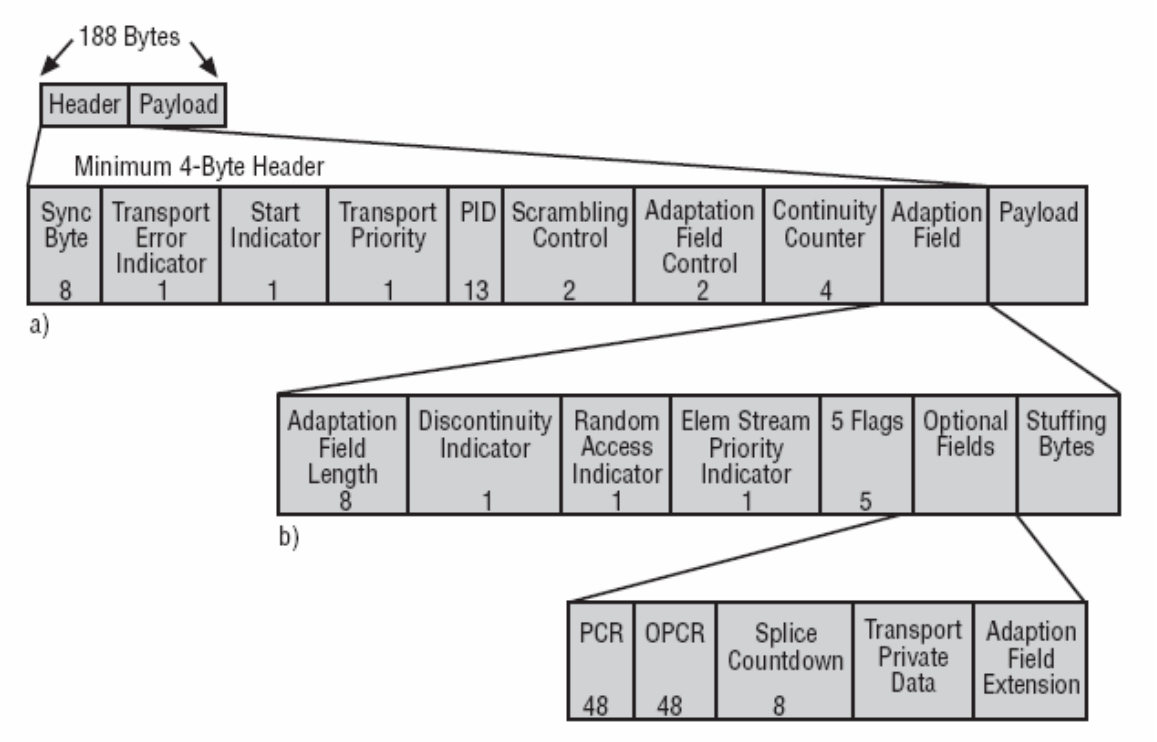


Figura 19: TS Header

Sync Byte

Este byte sirve para sincronizar el comienzo del paquete TS, el valor que se encuentra en este campo es 0x47.

Transport Error Indicador

En caso de que se encuentre algún error en la capa de transporte se setearía este bit en 1.

Start Indicador

Este campo indica el comienzo de un paquete PES, a continuación se profundizará en el uso de este indicador.

PID

Este campo se usa para identificar a que ES pertenece el paquete TS que acaba de llegar, por lo que cada ES tiene asociado un PID propio. Esta relación entre ES y PIDs es provista al cliente mediante los Program Specific Information (PSI) que son paquetes que a su vez tienen un PID fijo el cual es conocido. En este punto no se entrará en detalle.

Adaptation Field Control

Este campo puede tomar los siguientes valores:

01 – no trae adaptation field, pero si payload

10 – trae adaptation field, pero no payload

11 – trae adaptation field seguido por payload

00 – Reservado para uso futuro

Continuity Counter

Este campo es incrementado en 1 por el encoder por cada paquete enviado con el mismo PID, este campo es usado para detectar discontinuidades.

2.2.3.4. Uso del Transport Stream

Una vez que los encoders generan los diferentes ESs, estos son paquetizados generando un PES, como por lo general el tamaño de un paquete PES es mayor al de un paquete TS (el cual ocupa 188 Bytes) cada paquete dentro de este PES es dividido en una secuencia de paquetes TS, los cuales son transmitidos al cliente.

A esto se debe la existencia de campo Start Indicador dentro del protocolo TS, si este se encuentra en 1, esto indica que el primer Byte del payload del paquete TS se corresponde con el primer Byte del Header del paquete PES.

Uno de los aspectos que el protocolo TS permite, es dentro de un mismo Stream enviar ESs de diferentes “programas”, es decir por ejemplo en TV sobre IP, TS nos brinda la posibilidad de enviar diferentes canales televisivos dentro de un mismo Stream. Esto se llama múltiple Program Transport Streams.

En la Figura 20, se muestra una posible situación en donde dentro de en un mismo Stream TS, se están transmitiendo 2 programas simultáneamente. Como se puede observar en la Figura 20 se suele intercalar paquetes de audio con los de video dentro de la secuencia de paquetes TS enviada al cliente. Para este punto no existe ninguna consideración especial, la cantidad de paquetes enviados de cada uno de los ES y el orden en que estos son enviados, es dispuesto por quien genera los paquetes TS.

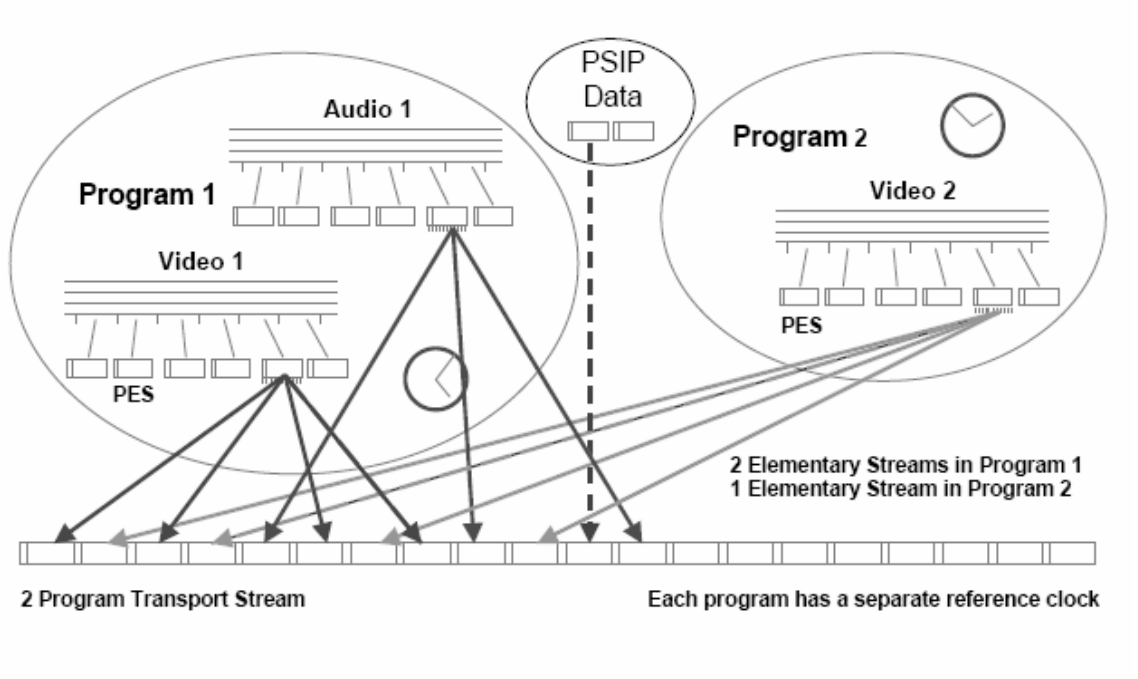


Figura 20: Uso de MPEG-TS

2.3. Protocolos de Monitoreo de Red

2.3.1. Simple Network Management Protocol (SNMP)

El Simple Network Management Protocol, tal como su nombre lo sugiere es un protocolo para el monitoreo y administración de redes definido en los RFC 1157 ("A Simple Network Management Protocol"), RFC 3411 ("An Architecture for Describing Simple Network Management Protocol Management Frameworks") y RFC 3416 ("Version 2 of the Protocol Operations for the Simple Network Management Protocol").

Como podemos observar en la Figura 21, en un escenario habitual en donde se lleve a cabo el uso de SNMP, se tienen agentes, quienes se pueden definir como los dispositivos a ser monitoreados y uno o más managers, quienes realizan el monitoreo y la administración de la red en cuestión.

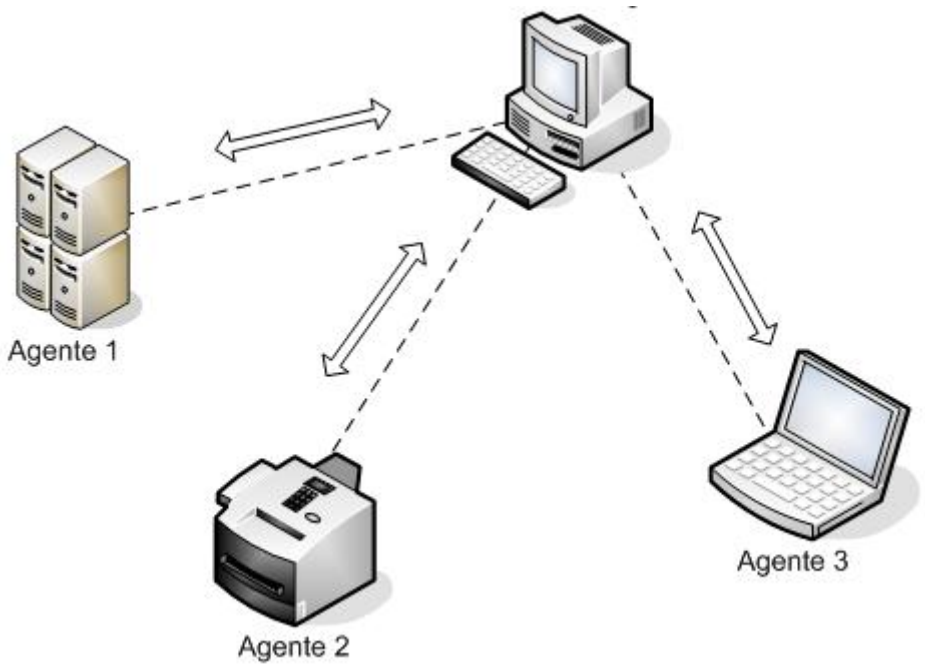


Figura 21: Introducción a SNMP

Como se puede observar en la Figura 22 dentro de este protocolo existen 2 diferentes vías de comunicación entre el agente y el manager, una la cual es inicializada por el manager quien realiza "Polling" sobre cada uno de los agentes (en otras palabras el manager consulta a cada cliente por el estado de cierta variable) y la otra, la cual es inicializada por el agente debido a la ocurrencia de cierto evento, llamada dentro del protocolo "Traps" (en este caso el agente se comunica con el manager con el fin de reportarle cierto evento).

Dentro de SNMP el método más usado es el de “Polling”, mientras que el método de “Traps” es usado únicamente para ciertos casos límites, como por ejemplo en el caso en que la carga de un servidor supere un cierto umbral o en el caso de que una impresora corporativa se esté quedando sin tinta (casos en los cuales es el agente quien emite una alerta al manager).

Continuando con el protocolo, dentro del método de “Polling” a su vez existen 2 diferentes tipos de comunicación (como se puede observar en Figura 22), una y a su vez la mas usual, es mediante la orden GET, mediante la cual el manager obtiene información acerca de cierta variable en el agente y la otra mediante la orden SET, en donde el manager setea cierta variable en el agente.

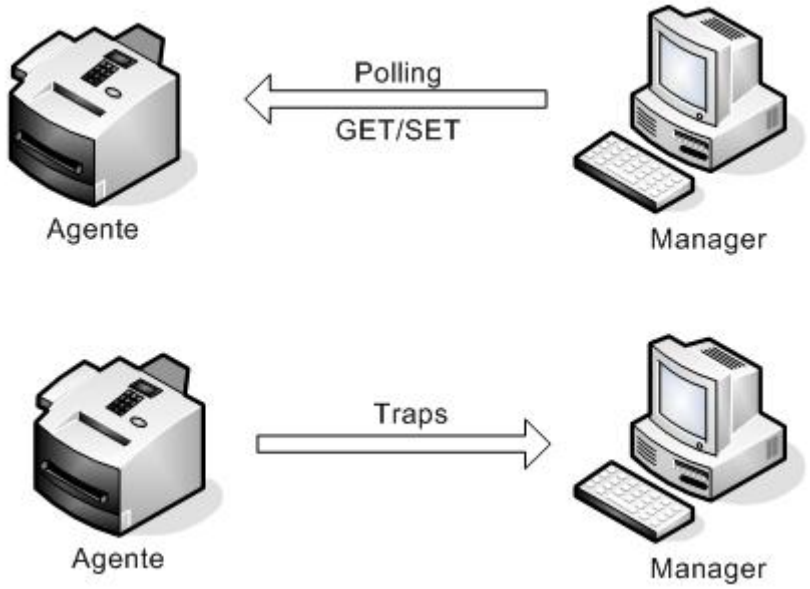


Figura 22: Tipos de comunicación

A continuación veremos las estructuras en base a las cuales se lleva a cabo la comunicación entre el manager y el agente.

Para poder establecer la comunicación entre el agente y el manager es necesario definir una MIB (***M****anagement* ***I****nformation* ***B****ase*), lo cual se podría ver como la interfaz mediante la cual estos se comunican (en otras palabras, es la estructura que permite que el manager y el agente hablen el mismo idioma).

En la Figura 23, se presenta un ejemplo en donde el manager tiene definidas 4 MIBs (por lo cual este únicamente podrá consultar datos sobre estas) y por otro lado tenemos agentes que implementan algunas de estas MIBs. Lógicamente estos agentes responderán consultas únicamente sobre las MIBs que implementan, en el ejemplo, el agente 1 únicamente responderá consultas sobre la MIB1 y la MIB4 respondiendo error en otro caso.

La principal diferencia entre los agentes y el manager es que en los agentes se tienen instancias concretas de las MIBs (es decir que para cada atributo de la MIB

se tiene un valor asociado) mientras que en el manager solo se tiene una definición de éstas.

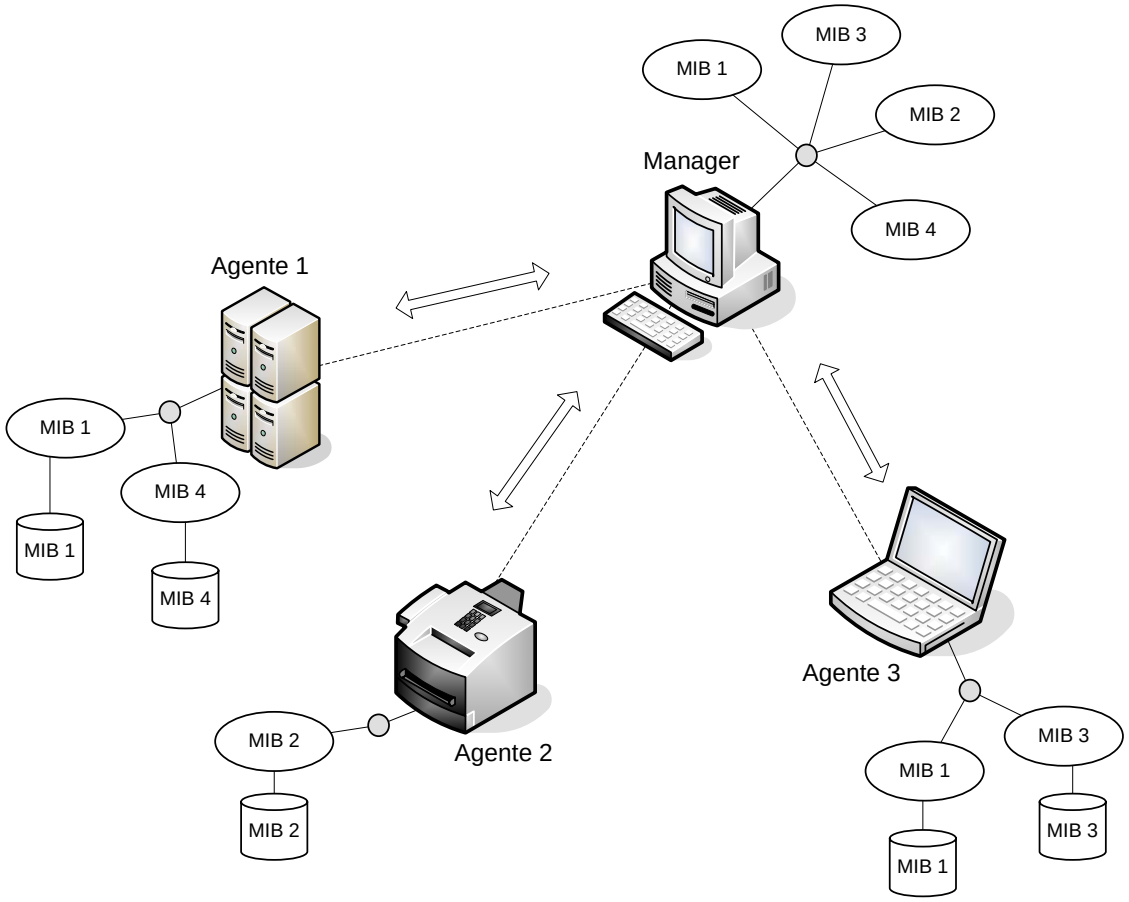


Figura 23: MIB en SNMP

A continuación profundizaremos en la definición de las MIBs. Como vimos anteriormente existen 2 conceptos diferentes relacionados con las MIBs, uno la definición de la MIB (presente en los managers y agentes) y otro la implementación de estas (presente únicamente en los agentes).

Respecto a la implementación de las MIBs existen varias librerías para desarrollar a estas, en particular en este proyecto usamos Net SNMP, no profundizaremos en este tema, por información acerca del mismo ver ANEXO IV

.

Respecto a la definición de las MIBs, estos son archivos de texto planos en donde usando una sintaxis llamada SMI (Structure of Management Information) (definida en los RFC 1155 (SMIv1) y RFC 2578 (SMIv2)) se expresa el contenido de las MIBs.

No se entrará en detalles acerca de esta sintaxis, por información acerca de la misma ver el ANEXO IV

.

Como se puede observar en la Figura 24, las MIBs son estructuras arborescentes mediante las cuales se definen variables (representadas por las hojas de dicho árbol) a las cuales en cada agente se le van a asociar un valor en concreto. Cada nodo del árbol definido en la MIB tiene un identificador único en su nivel de profundidad, en el ejemplo la raíz está identificado con el 1, el nodo address con el 1, el info con el 2, etc. lo cual permite concatenando todos los identificadores de los nodos, tener un identificado único para cada nodo dentro de la MIB. Por ejemplo en la Figura 24, el ID: 1.2.1 identifica al nodo name, el 1.1 al address, etc.

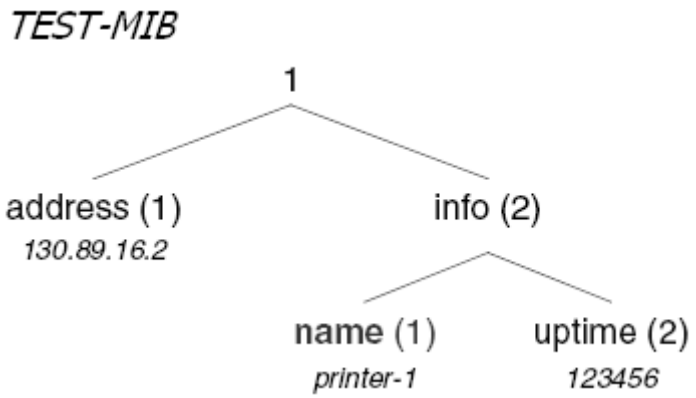


Figura 24: Definición de una MIB

A su vez como se puede observar en la Figura 25, cada MIB se encuentra ubicada dentro de un árbol global de MIBs, el cual es administrado por la organización IANA (***I****nternet* ***A****ssigned* ***N****umbers* ***A****uthority*).

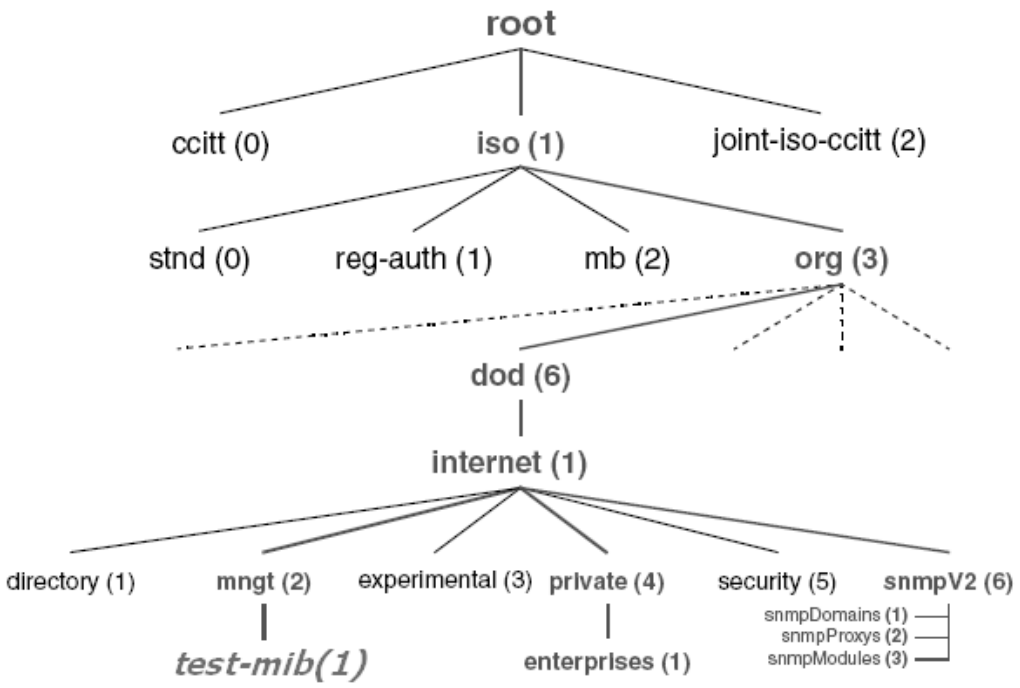


Figura 25: MIB árbol global

De esta manera cada atributo de cada MIB tiene un único identificador a nivel global (llamados OID). Por ejemplo: el OID de TEST-MIB:info.name es el 1.2.3.6.1.2.1.2.1.

Estos OID serán usados para realizar las diferentes consultas. Por ejemplo, si un manager realiza un GET del OID 1.2.3.6.1.2.1.2.1 (imaginando que en el agente se encuentra instanciado con los datos presentados en la Figura 24), el agente responderá “printer-1”.

Resumiendo, todo las variables a ser monitorizadas mediante SNMP deben pertenecer a una MIB, teniendo de esta manera un OID único a nivel global, el cual será usado para realizar las diferentes operaciones sobre esta variable (ya sea para que el manager realice GETs o SETs o para definir TRAPs en los agentes).

Finamente este protocolo pertenece a la capa de aplicación en el modelo OSI y como se puede observar en la Figura 26, este se trabaja en base a UDP como protocolo de transporte.

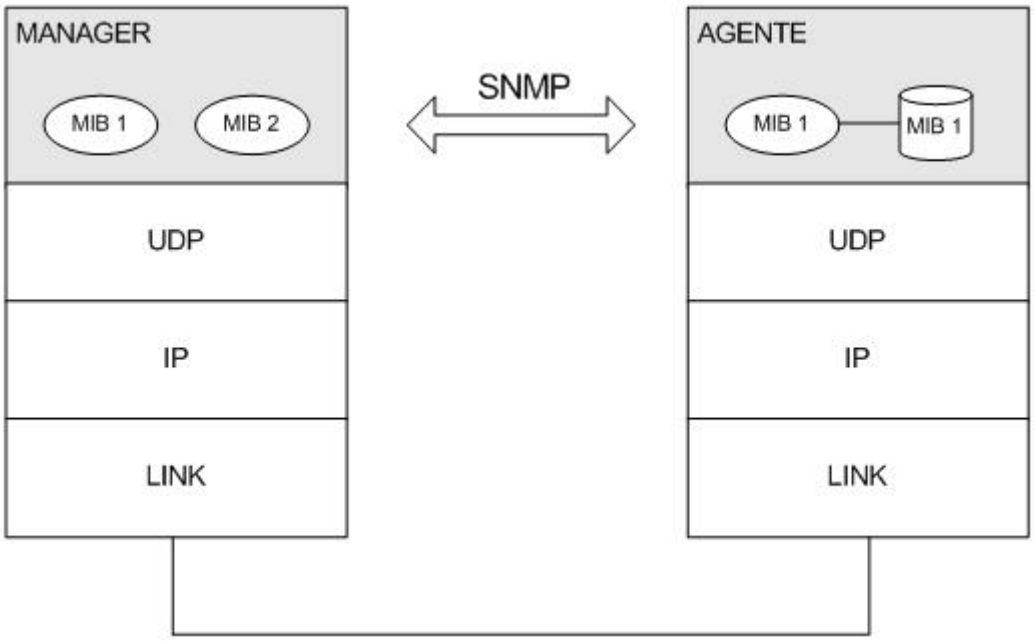


Figura 26: Protocolos de base

2.4. La Industria

2.4.1. Introducción

En esta sección damos una reseña de los software's utilizados por las VDN de Internet. Estos los categorizamos en reproductores y servidores. Se trata características generales de cada uno, prestaciones en sus nuevas versiones y licenciamiento.

2.4.2. Reproductores (players)

Existe una amplia gama de reproductores de audio y video, algunos con licenciamiento libre y otros propietarios. Presentamos a continuación ejemplos de ellos, que hoy en día son usados.

Real Player

Es desarrollado por RealNetwork y brinda la reproducción de múltiples formatos incluyendo la generación múltiple de RealAudio y RealVideo (.rm), así como la gama de los MPEG y QuickTime. Estos formatos de archivos de audio y video propios apuntan a la mejora en la compresión para aliviar el overhead, pero como consecuencia se dice que se ha perdido calidad, aunque la gente de Real dice que mejora a muchas de las soluciones actuales. Apunta al comercio de contenidos ya que no solo brinda soporte a la transmisión liviana para muchos clientes, sino que además en caso de querer reproducir los archivos se debe estar en línea, o sea que no se pueden grabar los archivos reproducidos. Esto se hace a través del .rm que incluye metadata entre otros de autoría y la ubicación del archivo multimedia. También brinda soporte a cualquier formato y cualquier dispositivo, lo cual también es característica de QuickTime presentado a continuación.

QuickTime

Es desarrollado por Apple el cual es compatible con muchos formatos de codificación. No es solo un reproductor sino toda una arquitectura multimedia que ofrece servidor, cliente e incluso herramientas que permiten la edición de recursos multimedia. Igual que Real Player ofrece su formato particular para video (.mov) y audio (.aiff). La lista de codecs que maneja incluye a los MPEG, H.264, 3GPP. Este último brinda además soporte a los dispositivos móviles tales como celulares y otros equipos móviles GSM.

Windows Media Player

Desarrollado por Microsoft el cual ha liberado varias versiones del mismo. La última versión del mismo es Windows Media Player 10 y en esta se brinda un nuevo diseño a nivel de interfaz, prestaciones en línea y mejoras de compatibilidad con dispositivos. Al igual que los anteriores posee un licenciamiento propietario.

VLC Media Player

VLC es un reproductor que se originó del proyecto VideoLan y es un software libre. VideoLAN es un proyecto que comenzaron estudiantes franceses de la Escuela Central de Paris (École Centrale Paris) en el año 1996.

Su objetivo es dar una solución completa en reproducción multimedia y puede ser usado en distintos escenarios. Inicialmente estaba dividido en dos proyectos, uno que contemplaba las funcionalidades de cliente y otro con las funcionalidades de servidor. Hoy en día estos proyectos están fusionados.

2.4.3. Servidores

Presentamos una reseña de algunos servidores actualmente usados, algunas de sus características y datos relevantes.

Helix Product Suite

La solución brindada por RealNetwork para la transmisión de streaming consta de dos paquetes, Helix Service Delivery Suite – Starter y Helix Service Delivery Suite – Full.

El paquete Starter incluye servicio de autorización a los recursos multimedia para usuarios, un mediador de pago y un adaptador de eventos de servidor.

El paquete Full además de incluir lo que se incluye en el Starter, se incluye Monitoreo de Calidad de Servicio, Notificación de Canal en Vivo y Procesador multimedia.

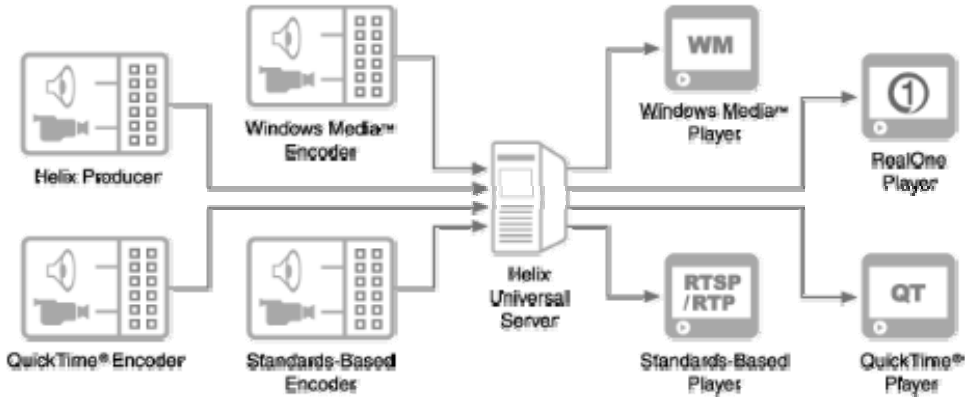


Figura 27: Encodeado y transmisión del Servidor Helix

Soporta los formatos RealAudio, RealVideo, Windows Media, Quicktime, MPEG-4, MP3 entre otros, pero para ello se debe contar con un encoder que logre dicho formato. Como podemos observar en la Figura 27 para algunos casos Helix cuenta con encoders, pero para otros no. Para esos casos se debe contar con un encoder externo como ser el Windows Media Encoder para los casos de los formatos correspondientes a las los archivos multimedia de Windows (.asf, .wma, .wmv).

Darwin Streaming Server y QuickTime Streaming Server (QTSS)

Ambos servidores de streaming son desarrollados por Apple al igual que el Reproductor "QuickTime Player". Estos más el QuickTime Pro, el cual brinda la capacidad de creación de contenido multimedia (por ejemplo combinar distintos tipos de archivos multimedia en una película) y el QuickTime Broadcaster, el cual permite realizar una transmisión broadcast de un evento (free download) componen la QuickTime suite.

El servidor QuickTime Streaming Server está incluido en el Mac OS X Server versión 10.2 y permite la transmisión de contenidos multimedia tanto en tiempo real como a demanda.

El Servidor Darwin Streaming Server está basado en QTSS pero está licenciado en código libre bajo la licencia APSL (Apple Public Source License). Este no posee algunas herramientas multimedia y de administración extendidas incluidas en QTSS.

Otras características de relevancia son que incluyen funcionalidad de streaming MPEG4 de forma nativa (no tiene que ser convertido a archivos .mov); Streaming de Audio MP3; autenticación de acceso para la protección del contenido multimedia; manejo de listas de reproducción del lado del server (ideal para radios o televisoras virtuales).

Windows Media 9 Series

El servidor de streaming desarrollado por Microsoft, ubicado dentro del paquete de solución Windows Media 9 Series es el Windows Media Services 9 Series. Este se debe usar en combinación con el Windows Media Encoder 9 Series, el cual realiza la codificación de los archivos de audio y video al formato aceptado por el servidor.

Algunas de las nuevas características son:

- Advanced Fast Start: Mejora el tiempo de respuesta cuando se inicia una transmisión eliminando el tiempo de buffering.
- Play while Archiving: Se permite la transmisión de contenidos multimedia que están siendo archivados para su transmisión tanto en broadcast como a demanda.
- Encoder Fallover URL Modifiers: Si se cuenta con encoders o fuentes de streaming alternativos brinda la posibilidad de en caso de falla o caída del encoder primario, se pueda retomar la transmisión con otro alternativo luego de cierto tiempo.

2.4.4. VLC Media Player

Es muy versátil dado que soporta una amplia gama de codecs, formatos, protocolos, plataformas y otras flexibilidades (ver [Ref10] y [Ref11]).

Además cuenta con alguna documentación, foros y el código se mantiene en constante renovación por programadores de todo el mundo, debido a que es libre y distribuido bajo licencia GPL desde el 1º de febrero del 2001. Su home page es http://www.videolan.org/.

Para nuestro proyecto, es una gran ventaja que este reproductor pueda ser usado tanto como cliente, como servidor. Dicho servidor puede transmitir en diferentes protocolos como ser udp, http, mms, tanto en unicast como multicast, en IPv4 o IPv6. Puede transcodear y al igual que el cliente posee amplio soporte de formatos y codecs.

Actualmente está disponible la versión VLC media player 0.8.6, la cual incorpora correcciones y mejoras a nivel de compatibilidad con Windows Media Video 9 y Flash videos. También se añade mejoras de decodificación para h.264 y de interfaz en Mac OS X. Es muy usado ya que para la versión VLC media player 0.8.5 se registraron 29 millones de descargas.

Debido a todas estas características es el que hemos elegido como reproductor para utilizarlo como cliente que realiza las mediciones y además como servidor.

Arquitectura

El VLC posee una arquitectura modularizada.

Cuenta con una librería de base, la librería LibVLC en la cual se encuentra el núcleo de la aplicación y módulos específicos. Estos módulos son levantados dinámicamente según sean necesarios. Estos módulos están categorizados y ordenados según su aplicación. Algunas categorías son access, codec, demux y packetizer.

Sobre la Arquitectura se amplia en el ANEXO I

.

Modo de Uso

Tanto cliente como servidor es posible ejecutarlos mediante interfaz gráfica mediante los Skins, a través de línea de comando o telnet (Ver [Ref10] Interfaces and control).

Por ejemplo:

Server

Telnet

1. Crear un archivo (vlc-use-vlm.conf) con el siguiente contenido:

```
new channel1 broadcast enabled
setup channel1 input  file:///home/videos/sequence.mp4
setup channel1 option loop
setup channel1 output #duplicate{dst=std{access=udp,mux=ts,dst=10.10.2.80:1234}}
```

```
control channel1 play
```

2. ejecutar desde línea comando:

```
vlc --color -I telnet --vlm-conf vlc-use-vlm.conf
```

Línea de comando

```
vlc --sout '#duplicate{dst=std{access=udp,mux=ts,dst=10.10.2.80:1234}}' \
file:///home/videos/sequence.mp4
```

Cliente

Telnet

1. Crear un archivo (vlc-use-vlm.conf) con el siguiente contenido:

```
new channel1 broadcast enabled
setup channel1 input udp://@10.10.2.80:1234
setup channel1 option loop
setup channel1 output #duplicate{dst=display}

control channel1 play
```

2. ejecutar desde linea comando:

```
vlc --color -I telnet --vlm-conf vlc-use-vlm.conf
```

Línea de comando

```
vlc --sout "#duplicate{dst=display}" udp://@10.10.2.80:1234
```

3. Solución propuesta

3.1. Introducción

El hallar una solución factible al problema planteado por este proyecto fue una actividad que requirió de un largo tiempo de investigación y estudio de las diferentes especificaciones.

Se intentaba conseguir una medida que representara de la mejor manera la calidad de video percibida en un cliente multimedia, pero dicha medida no era un requerimiento sino que era unos de los puntos en los que se debía investigar, es decir que se debía definir en que nivel ubicarse para realizar estas mediciones dentro de las diferentes especificaciones, teniendo en cuenta que cuanto a más bajo nivel se realicen las mismas (por ejemplo midiendo pérdidas de paquetes a nivel de red) estas serán mas alejadas de la realidad y cuanto a más a alto nivel se lleven a cabo (por ejemplo a nivel de imágenes decodificadas) esta tarea se tornaría de mayor complejidad.

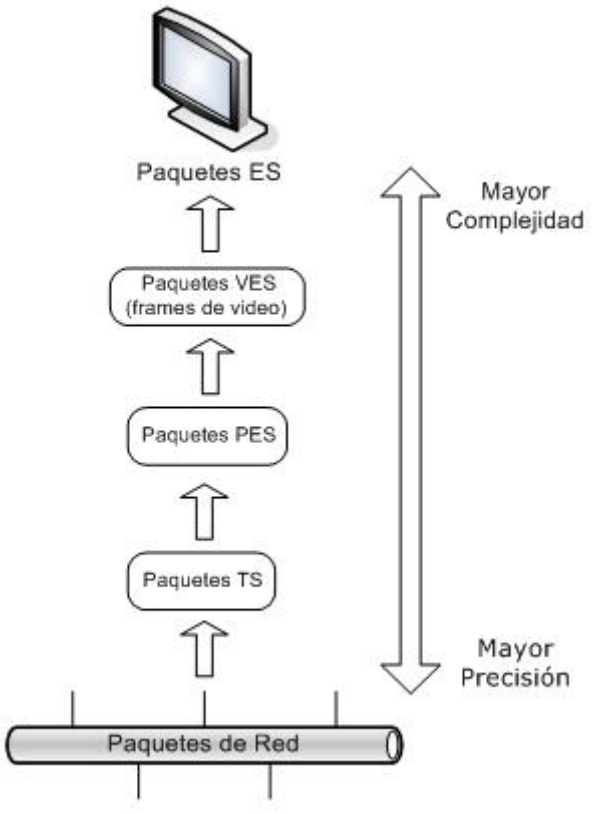


Figura 28: Capas en el proceso de ejecución de un Stream

Por lo anteriormente mencionado se descartó el medir las pérdidas a nivel de red dado que estas no nos brindarían una clara representación de la calidad de video. Esto es debido a que es sumamente complejo el tener una idea de cual puede llegar a ser el impacto en el video una pérdida de un paquete a este nivel y a su vez el hecho que hallamos recibido un paquete a este nivel no nos asegura que este no

sea descartado más adelante, por ejemplo por haber llegado en forma tardía o por algún overflow de algún buffer interno.

Por otro lado el medir calidad de video sobre imágenes decodificadas (lo que sin lugar a dudas seria lo óptimo) es una tarea sumamente compleja debido a que esto requiere de implementar alguna técnica de reconocimiento de patrones de imágenes, lo cual no cabe dentro de las posibilidades de este proyecto, motivo por el cual este tipo de mediciones también fue descartado.

Durante un tiempo considerable se investigó a fondo la especificación MPEG TS con el fin de realizar nuestras mediciones allí, dado que esta especificación nos brindaba una serie de ventajas respecto a otras, como ser el hecho que dentro de esta se mantiene un identificador de secuencia en los diferentes paquetes de un stream (lo cual nos brindaba la posibilidad de tener una noción de cuando se perdió un paquete). Finalmente se llegó a la conclusión que midiendo pérdidas a este nivel no lográbamos tener una clara percepción de la calidad de video por lo que este tipo de medidas también se descartó, pasando a enfocarnos en las medidas sobre pérdidas a nivel de frames de video (es decir medir pérdidas de paquetes dentro de un VES).

Luego del estudio de las especificaciones MPEG2 y MPEG4 en busca de algún atributo dentro de estas que nos permitiera medir las pérdidas en los frames (tal como ser algún número de secuencia) se llegó a la conclusión que el único elemento a tener en cuenta para medir esto eran los DTS y PTS, es decir los tiempos de decodificación y presentación de cada frame. Si un paquete tenía marcado un tiempo de decodificación menor al tiempo actual, entonces este frame sería descartado. Basándonos en este atributo estábamos en condiciones de medir frames tardíos, que básicamente son equivalentes a frames perdidos.

Analizando en mayor profundidad esta solución estábamos cubriendo el 100% de las pérdidas sobre HTTP, debido a que en este protocolo no existe el concepto de pérdida porque todos los paquetes perdidos son retransmitidos (es decir que nunca se pierde un paquete sino que lo peor que puede ocurrir es que estos lleguen tarde) pero esto mismo no ocurría en los protocolos asincrónicos tal como UDP o RTP, en donde únicamente estaríamos reconociendo paquetes tardíos, razón por la cual la solución no era completa y por lo tanto no era satisfactoria.

Dado estos últimos resultados cada vez se tornaba más lejana la posibilidad de medir pérdidas en frames de video independientemente del protocolo de streaming, lo cual nos preocupaba de sobremanera. Antes de pasar a analizar alguna otra solución (que tampoco restaban muchas) se decidió investigar con mayor profundidad las especificaciones MPEG2 y MPEG4 intentando encontrar alguna alternativa a nuestro problema.

Finalmente se encontró un concepto dentro de estas especificaciones que en principio lo habíamos pasado por alto pero que el mismo iba a ser un elemento fundamental en la construcción de nuestra solución, los llamados “User Data”.

Estos “User Data” son secuencias de bytes pertenecientes a una aplicación específica, las cuales son inyectadas dentro de una secuencia de frames. En otras palabras, estos estándares brindan la posibilidad de introducir información propia

de nuestra aplicación dentro de un streaming, la cual será streameada y recibida en el cliente (como se puede observar en la Figura 29).

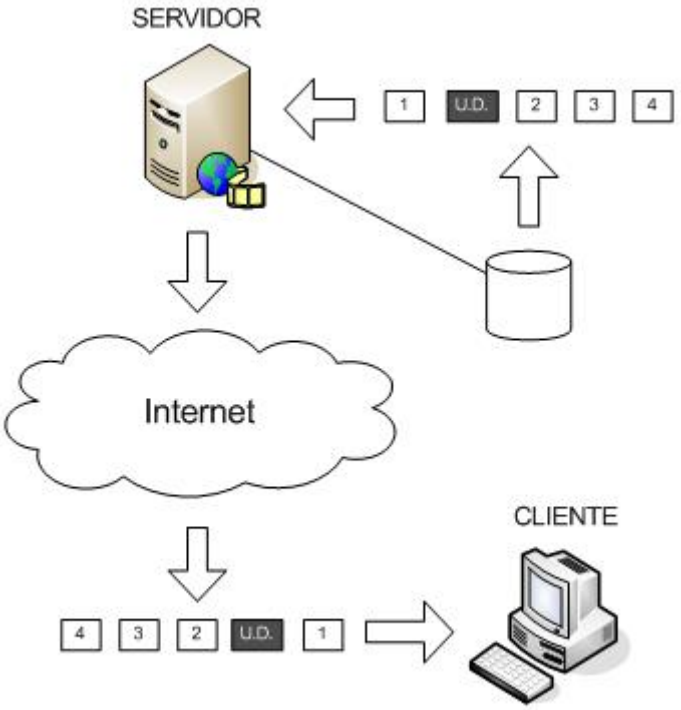


Figura 29: User Data

Profundizando en este concepto dentro la especificación MPEG4 y tal como se puede observar en la Figura 30 tenemos que el más bajo nivel donde el estándar permite introducir User Data es al nivel del header del Group of Video Object Planes.

Como vimos en la sección 2.2 Protocolos de Streaming un Video Object Plane es equivalente a un frame de video (o una Picture dentro de la especificación MPEG2) y un Group of Video Object Planes es equivalente al concepto de Group of Pictures de MPEG2 (también llamada GOP).

Y efectivamente el concepto de User Data es equivalente dentro de MPEG2 dado que dentro de este estándar el nivel más bajo donde se puede introducir User Data es a nivel del header del Group of Pictures.

Siendo más gráfico y tal como se puede observar en la Figura 31, lo que nos permite la especificación es mandar tantos User Data como queramos, pero separados al menos por un GOP o GVOP (recordar que un GOP es la secuencia de frames entre un frame I y el siguiente).

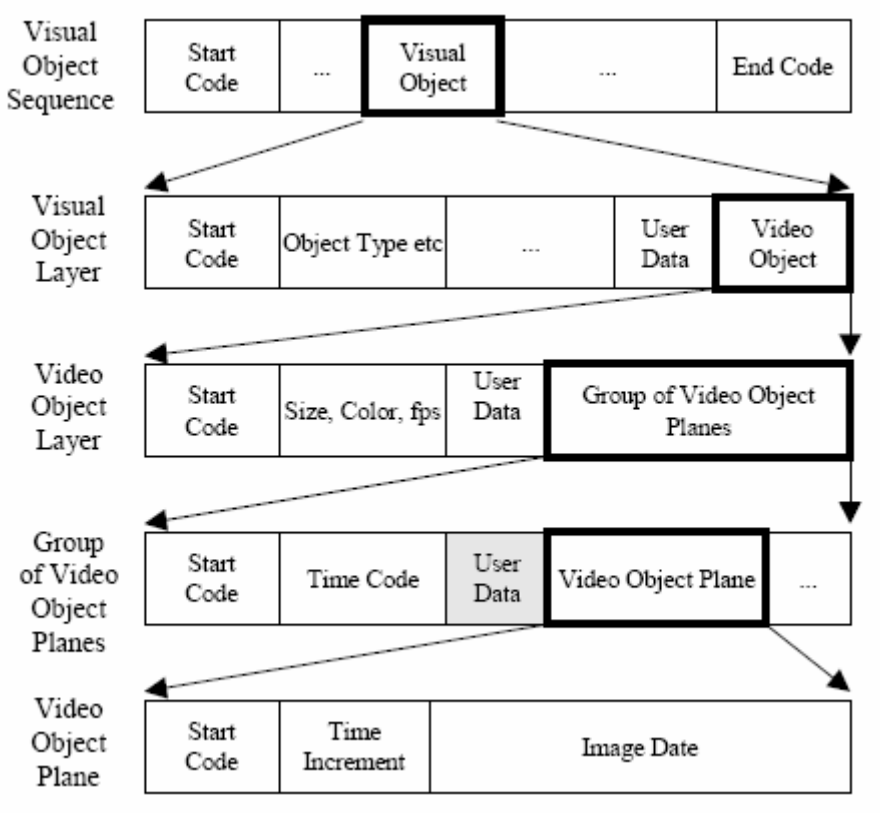


Figura 30: MPEG4 Estructura de VOS

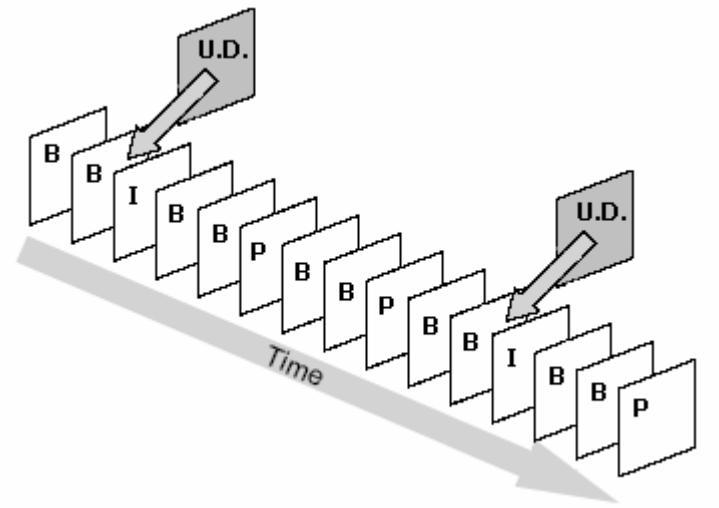


Figura 31: Máximo período entre Users Datas

Dada la falta de algún atributo dentro las especificaciones MPEG que nos permitiera llegar a medir pérdidas a nivel de frames, la posibilidad de poder insertar información propia a nuestra aplicación la cual nos brinde dicho elemento clave para detectar pérdidas, fue considerada una solución más que factible y apropiada dentro del grupo.

A continuación veremos como es que fue diseñada la solución dado el concepto de User Data.

3.2. Diseño de la solución

La solución diseñada consiste en enviar en el header de cada GOP un User Data con información acerca de la cantidad de frames enviados por el Server (discriminada por el tipo de frame) desde el comienzo de la ejecución del Stream.

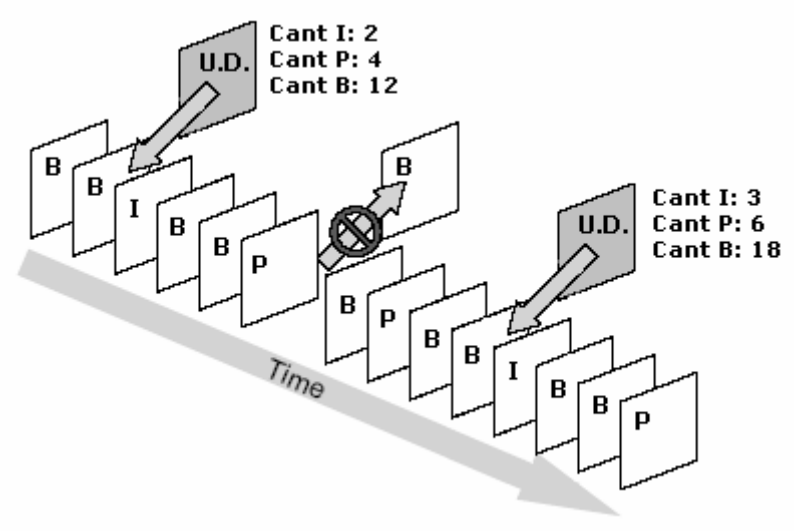


Figura 32: Ejemplo de GVOP con User Data con pérdida de B

Como se puede observar en la Figura 32, el primer User Data establece que hasta ese momento el Server envió 2 frames I, 4 frames P y 12 frames B, continuando con la ejecución en el segundo User Data enviado se presenta que se han enviado 3 frames I, 6 frames P y 18 frames B (que como se puede observar lo incrementado es igual a la cantidad de frames enviados en ese GOP).

La idea de fondo, es que en el cliente se vayan contabilizando la cantidad de frames recibidos y se contraste esta cantidad con la información recibida en los User Data enviados por el Server.

Continuando con el ejemplo presentado en la Figura 32, en donde se puede observar que se pierde un frame de tipo B, el cliente tendría contabilizado que recibió 1 frame I, 2 frames P y 5 frames B en el ultimo período (definiendo un período como el tiempo transcurrido entre la recepción de 2 User Data consecutivos) y comparando eso contra la información recibida en el último User Data se deduciría que se perdió un frame de tipo B en dicho período.

Si bien esta solución es relativamente simple tiene como contra que estamos perdiendo la espontaneidad de la solución, es decir se nos hace imposible reportar que se perdió un frame en un instante de tiempo dado, sino que los reportes siempre refieren a periodos de tiempo concretos. Por otro lado estos períodos de tiempo son relativamente pequeños (el peor caso no pasa de los 5 segundos) por lo que esta desventaja no es del todo importante, en lo que si esta solución no es del

todo completa es en el hecho que con la misma no podemos tener una medida precisa de las pérdidas en ráfagas (por ejemplo conocer el tamaño promedio de las ráfagas), sino que para conocer dicha información se requiere de algún cálculo estadístico, el cual si bien no es imposible de realizar este nunca alcanzaría la misma precisión de un cálculo directo (es decir contando los frames perdidos de forma consecutiva).

Más allá de estas desventajas presentadas anteriormente, hemos diseñado una solución la cual nos permite contabilizar de forma muy precisa los frames perdidos durante la ejecución de un streaming discriminados por tipo de frame, lo cual básicamente era el objetivo trazado.

3.3. Detalle de la solución

A continuación se presenta un detalle de todos los datos recabados en el cliente, los cuales son accesibles mediante SNMP.

3.3.1. Variables Genéricas

- URL del Stream que se está recibiendo.
- Protocolo de Streaming.
- IP del Servidor de Streaming.
- Puerto activo del Servidor de Streaming.
- Muxer del Streaming.
- Codec de Video del Streaming.
- Codec de Audio del Streaming.
- Codec de Subtítulos del Streaming.
- Momento en el que se inicio la lectura del Streaming.
- Tiempo que lleva el Streaming ejecutándose en el cliente.

3.3.2. Variables Semi Estáticas

- Tamaño Promedio de Frame I desde el comienzo de la ejecución del Streaming.
- Tamaño Promedio de Frame P desde el comienzo de la ejecución del Streaming.
- Tamaño Promedio de Frame B desde el comienzo de la ejecución del Streaming.
- Bitrate Promedio desde el comienzo de la ejecución del Streaming.

3.3.3. Variables Dinámicas

- Cantidad Total de frames I que deberían haber llegado al cliente desde el comienzo del Streaming.
- Cantidad Total de frames I que efectivamente llegaron al cliente desde el comienzo del Streaming.

- Cantidad Total de frames P que deberían haber llegado al cliente desde el comienzo del Streaming.
- Cantidad Total de frames P que efectivamente llegaron al cliente desde el comienzo del Streaming.
- Cantidad Total de frames B que deberían haber llegado al cliente desde el comienzo del Streaming.
- Cantidad Total de frames B que efectivamente llegaron al cliente desde el comienzo del Streaming.
- Identificador de Periodo (entero autoincremental)
- Duración del último período (identificado por el ID de período)
- Tamaño promedio de frames I en el último período
- Tamaño promedio de frames P en el último período
- Tamaño promedio de frames B en el último período
- Cantidad de frames I perdidos en el último período
- Cantidad de frames P perdidos en el último período
- Cantidad de frames B perdidos en el último período
- Bitrate en el último período

Cabe aclarar que para el cálculo de todos los bitrates se tuvo en cuenta el payload de los frames de video, es decir que el bitrate se calculó como la relación entre los bits recibidos en los payloads de los frame de video y la duración del período.

Los resultados medidos fueron separados de la manera anteriormente presentada con el fin de optimizar los pedidos mediante SNMP, se siguió la estrategia de agrupar en un mismo conjunto todos los datos consultados con mayor frecuencia los cuales serán consultados usando la primitiva SNMP WALK, ahorrando así traer información relativamente estática como puede ser la información genérica del Streaming muy a menudo.

A continuación veremos como fue llevada a cabo esta solución dentro del marco del proyecto.

En esta sección se presenta una visión a alto nivel de la arquitectura de la solución a desarrollar. Para esto veremos una breve introducción a la arquitectura interna del VLC y para luego centrarnos en la arquitectura de nuestra solución. Luego contrastaremos la arquitectura definida con la solución diseñada y veremos sus ventajas.

3.4. Arquitectura del VLC

El VLC es una aplicación la cual se basa firmemente en las funcionalidades provistas por una librería llamada LibVLC. Esta librería brinda múltiples funcionalidades de carácter genérico entre las que se encuentra la carga dinámica de módulos, el manejo genérico de buffers, las primitivas de acceso a la red, entre otras.

Figura 33: Principales componentes

A su vez el VLC está compuesto por una numerosa cantidad de módulos (muchos de los cuales son cargados en forma dinámica mediante la librería LibVLC), entre los que se encuentran las implementaciones de los diferentes demuxers, (MPEG-TS, MPEG-PS, etc.), implementaciones de los diferentes codecs de audio y video (MPEG2, MPEG4, H264, etc.), implementaciones de las diferentes interfaces gráficas (Telnet, Ventanas, etc.), etc. Es decir que en el VLC cada funcionalidad especifica esta implementada mediante un módulo el cual seguramente use las diferentes funcionalidades provistas por la librería LiBVLC.

A su vez muchos de estos módulos son construidos en base a librerías externas, tales como las implementaciones de varios codecs (por ejemplo usando FFMpeg), la interfaz gráfica (usando WxGidgets), etc.

Claramente la arquitectura de nuestra solución debía estar enmarcada en la arquitectura general del VLC, por lo que se debía seguir ciertos patrones de diseño, tal como el diseñar una arquitectura modularizada.

En la Figura 34 se presenta a grandes rasgos las interacciones entre los diferentes componentes involucrados dentro de la arquitectura de la solución. Se tiene que los módulos AUDIT interactúan con la librería LibVLC (usando así las primitivas propias del VLC para implementación de funcionalidades específicas, como ser el manejo de multiconcurrencia), la Librería externa Net-SNMP para la implementación de SNMP y por último se interactúa también con módulos propios del VLC (como por ejemplo con módulos de codec, packetizer, etc.), de donde se extrae cierta información concreta usada para generar ciertos reportes.

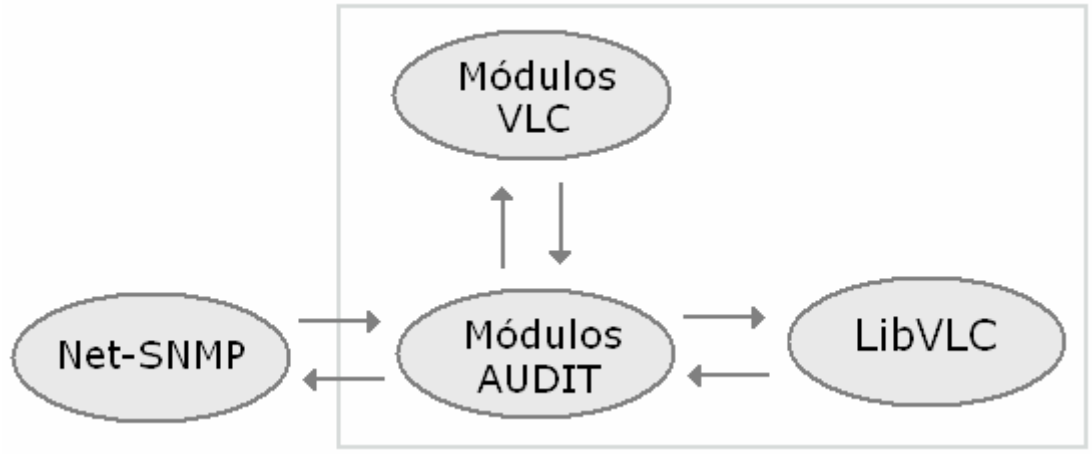


Figura 34: Componentes e Interacciones en la Solución

Los componentes dentro del cuadro de la Figura 34 son sobre los cuales se realizaron cambios o los nuevos desarrollos pertenecientes a la solución. En la sección 3.5 Módulos de la Solución vemos los diferentes módulos a desarrollar y la interacción entre estos.

3.5. Módulos de la Solución

Los módulos a desarrollar serán agrupados en 2 grandes grupos, el de los módulos pertenecientes al núcleo de solución y los módulos auxiliares.

Dentro del grupo de los módulos del núcleo de la solución se encuentran los módulos mediante los cuales se realiza el proceso de cálculo y condensación de los datos a medir, los módulos de logueo de dichos datos y la exportación de dichos datos mediante SNMP.

Los módulos auxiliares consisten en módulos implementados con el fin de recabar cierta información particular a cada caso, la cual luego es procesada en el núcleo del sistema.

En la Figura 35 se presenta un diagrama mostrando lo comentado anteriormente.

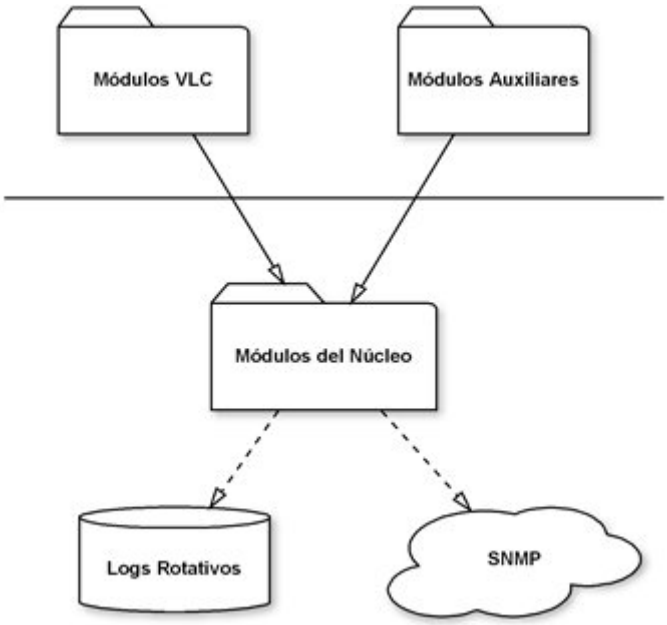


Figura 35: Arquitectura de la Solución

3.5.1. Núcleo del sistema

Como podemos observar en la Figura 36 dentro del núcleo del sistema existen 3 diferentes módulos, el módulo QOS, el módulo LoggerAudit y el módulo SNMP los cuales se explican con mayor nivel de detalle a continuación.

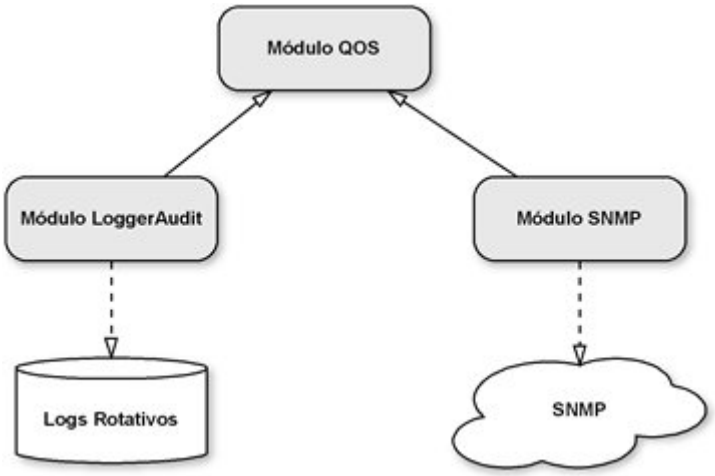


Figura 36: Arquitectura del Núcleo

3.5.1.1. Módulo QOS

Al ocurrir ciertos eventos durante la ejecución del VLC (eventos que abarcan desde la inicialización o finalización de cierto stream hasta la lectura de un nuevo paquete desde los diferentes buffers), se producen llamadas al sistema AUDIT, la cuales son recepcionadas y procesadas por el módulo QOS.

En otras palabras, este módulo es el encargado de administrar la información acerca de la ocurrencia de los diferentes eventos durante la ejecución del VLC, proporcionando los datos necesarios a los módulos LoggerAudit y SNMP para su correcta ejecución.

En los casos que los datos requieren un proceso de condensación periódico (tal como es en el caso de los diferentes promedios), este módulo también es el encargado de realizar dichos cálculos.

Este módulo es inicializado cuando se inicializa el VLC y el mismo se ejecuta en un thread propio. La ejecución del main thread de este módulo es la responsable de realizar las condensaciones periódicas de los datos que corresponden.

A su vez este módulo maneja 2 diferentes estructuras de datos, una referida al almacenamiento de los datos a exportar mediante SNMP y otra que represente una cola de eventos a loguear.

En la Figura 37 se presenta un diagrama con los puntos anteriormente comentados.

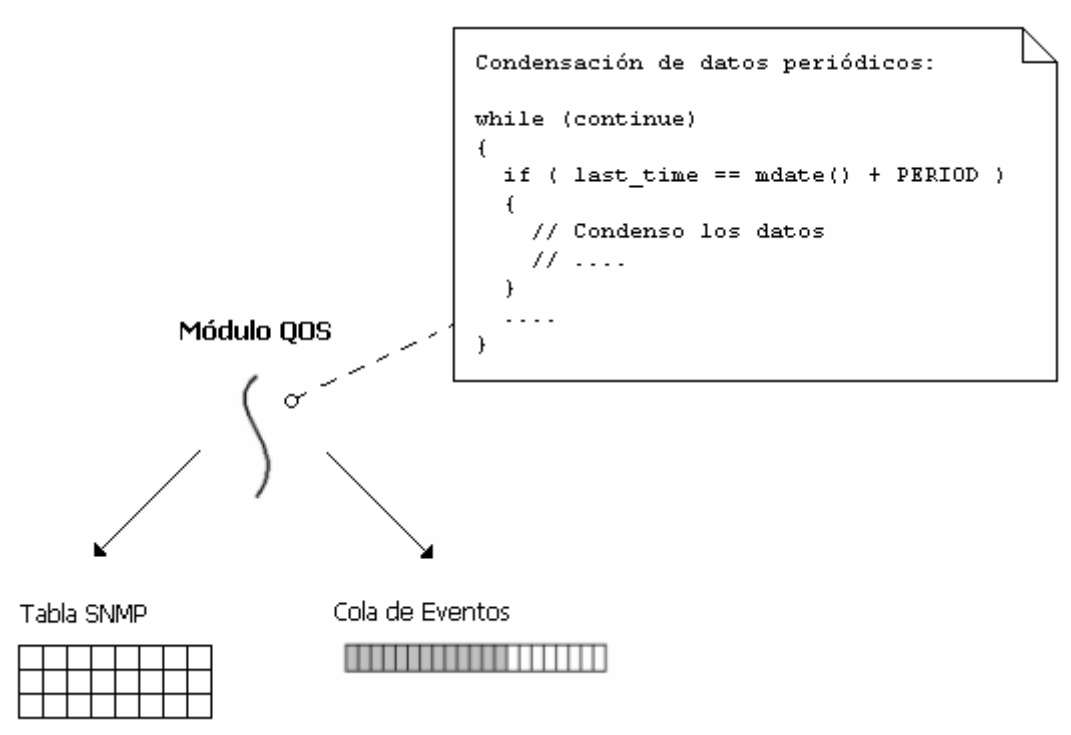


Figura 37: Estructuras Módulo QoS

3.5.1.2. Módulo LoggerAudit

Este módulo, cuando es invocado (es el usuario quien decide que módulo instanciar y que módulo no), se ejecuta en un thread propio desde el inicio del VLC. La responsabilidad de este módulo es la de ir consumiendo registros de la cola de eventos del modulo QOS, logueándolos y borrándolos de dicha cola.

Como podemos observar en la Figura 38 el módulo QOS a medida que tenga la información para loguear, la ira agregando a la cola de eventos para que el módulo LoggerAudit las grabe en los logs rotativos.

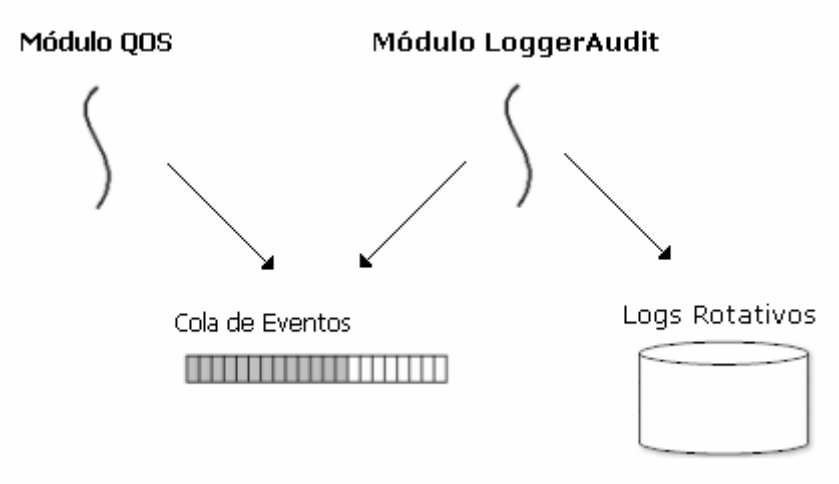


Figura 38: Estructuras Módulo LoggerAudit

3.5.1.3. Módulo SNMP

Por último este módulo es el encargado de la exportación de la información definida en la MIB VLC-AUDIT-MIB (ver ANEXO IV

) mediante SNMP. Para esto este se basa en la información brindada por el módulo QOS, concentrando así en este módulo únicamente la lógica referida al manejo de SNMP.

Al igual que los módulos anteriores en caso de ser invocado por el usuario, este módulo se ejecuta en un thread propio (observar Figura 39).

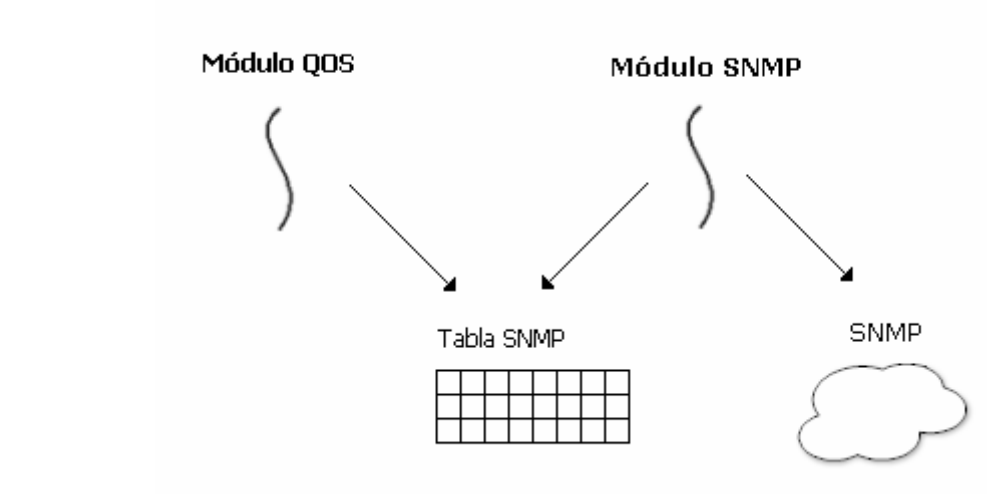


Figura 39: Estructuras Módulo SNMP

3.5.2. Módulos Auxiliares

3.5.2.1. Módulo QOS Duplicate

Este módulo se encuentra ubicado en una posición estratégica dentro del proceso de ejecución del Streaming (justo antes del módulo de decoding), por lo que nos brinda la posibilidad de detectar el arribo de frames prontos para ser procesados por el decorder. Esta información (tipo de frame y tamaño del mismo) es pasada al módulo QOS en donde esta es procesada.

A diferencia con los módulos anteriores, este módulo es instanciado en la inicialización de cada streaming (no en la inicialización del VLC) y existe uno para cada Streaming que este recibiendo el cliente (observar Figura 40).

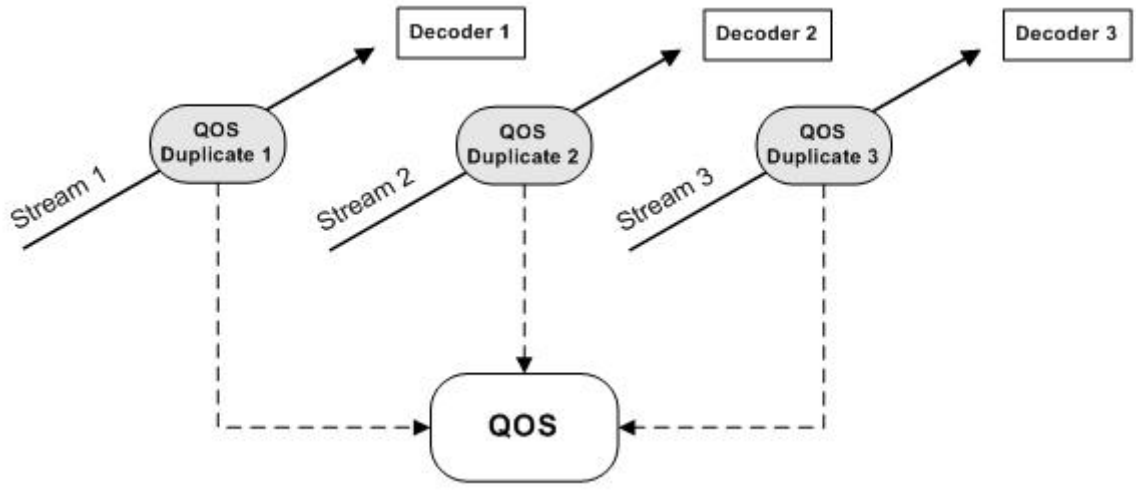


Figura 40: Módulo QOS Duplicate

Como es de esperar este módulo es el principal proveedor de datos al núcleo de la solución, al grado de que si este no se encuentra presente, la única información que se reporta es la correspondiente a los datos genéricos (3.3.1).

3.6. Diseño de la Solución VS Arquitectura de la Solución

Como vimos en el capítulo anterior el diseño de nuestra solución se basa firmemente en la transferencia de información acerca de la cantidad de frames enviados por el servidor mediante los User Data. A continuación vemos como este diseño cabe en la arquitectura anteriormente descripta.

3.6.1. Proceso en el Servidor

En la Figura 41 se presenta a muy alto nivel el proceso llevado a cabo en el servidor al ejecutar un streaming.

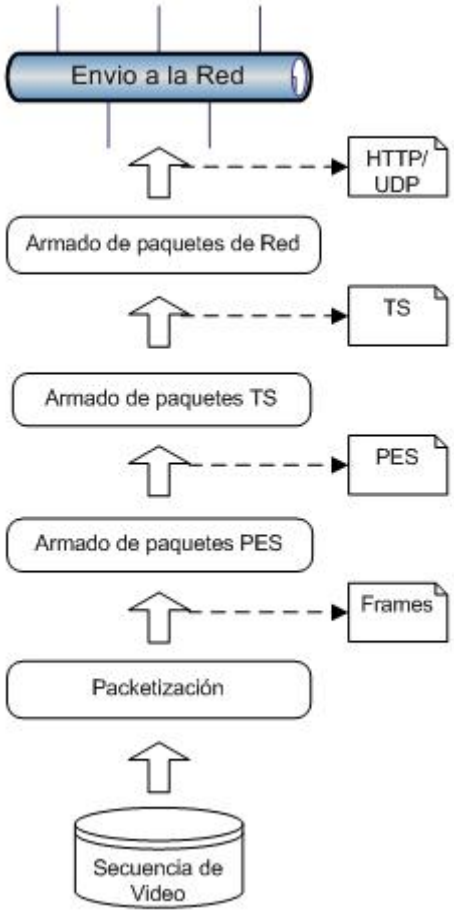


Figura 41: Proceso en el servidor

La secuencia a stremear es leída de algún medio de almacenamiento, esta es paquetizada (es decir se generan los frames tanto de audio como de video). Luego con esta se genera los paquetes PES para generar los paquetes TS y finalmente con estos se arman los paquetes de red (ya sea UDP, HTTP o el protocolo de streaming que sea) los cuales son enviados por la red.

Como se puede observar en la Figura 42 en donde se profundiza en el armado de los paquetes PES, se tiene básicamente un módulo de VLC llamado PES quien se encarga de armar los paquetes PES en base a los frames recibidos. Dicho módulo lo modificamos de forma tal que cada frame recibido se le es pasado al módulo QOS, quien contabiliza los diferentes frames y genera los User Data con la información correspondiente. Estos User Data son integrados al Streaming dentro de los paquetes PES, los cuales finalmente son stremeados por la red.

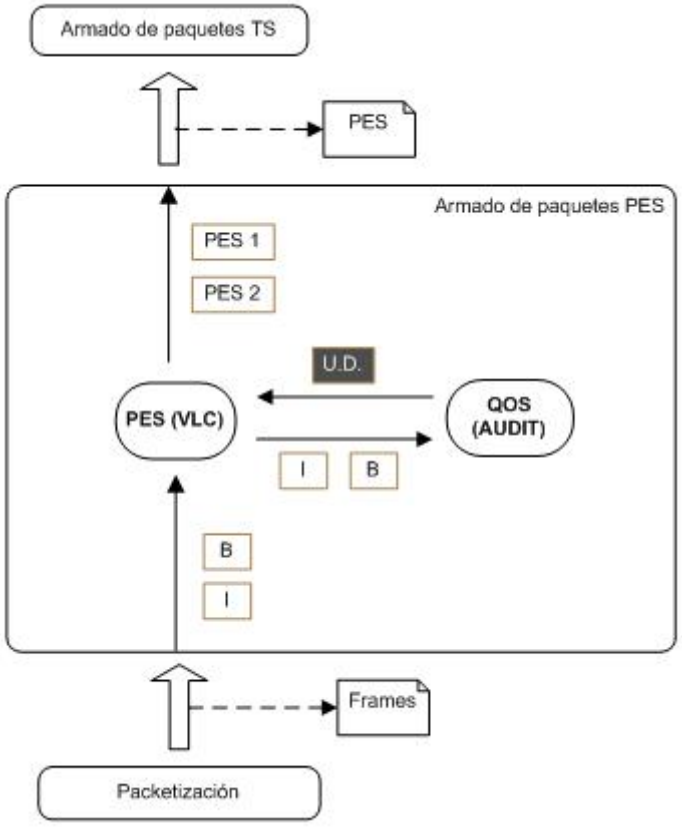


Figura 42: Armado de paquetes PES

3.6.2. Proceso en el Cliente

Al igual que en proceso en el servidor en la Figura 43 se presenta a muy alto nivel el proceso llevado a cabo en el cliente durante la ejecución de un Streaming.

Desde la red se reciben los paquetes de Red (HTTP, UDP, etc.), de estos se extraen los paquetes TS, luego con los paquetes TS se arman los paquetes PES, para que con estos se armen los frames (tanto de audio como de video). Estos frames son decodificados y presentados en pantalla.

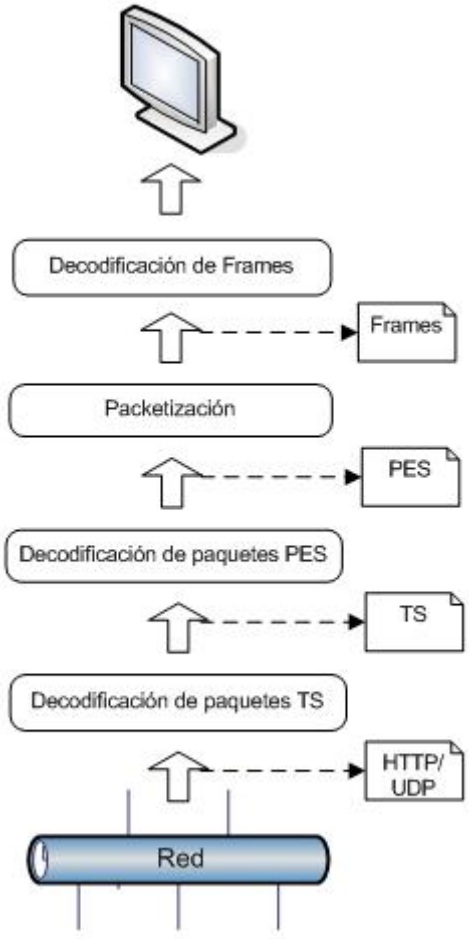


Figura 43: Proceso en el cliente

Como se puede observar en la Figura 44, el proceso de packetizacion se centra en los módulos de packetizacion del VLC, los cuales existen uno para cada tipo de especificación dado que no todas las especificaciones paquetizan de igual manera.

Los módulos de packetizer de MPEG4 y MPEG2 fueron modificados con el fin de leer los User Data, dado que por defecto estos son descartados en estos módulos.

También en esta Figura, presenta al módulo QOS Duplicate, el cual reporta la llegada de frames al módulo QOS y a su vez descarta los User Data llegándole así al Decoder únicamente frames.

De esta manera es como se implementó la solución diseñada en la arquitectura definida. A continuación vemos las ventajas de esta arquitectura.

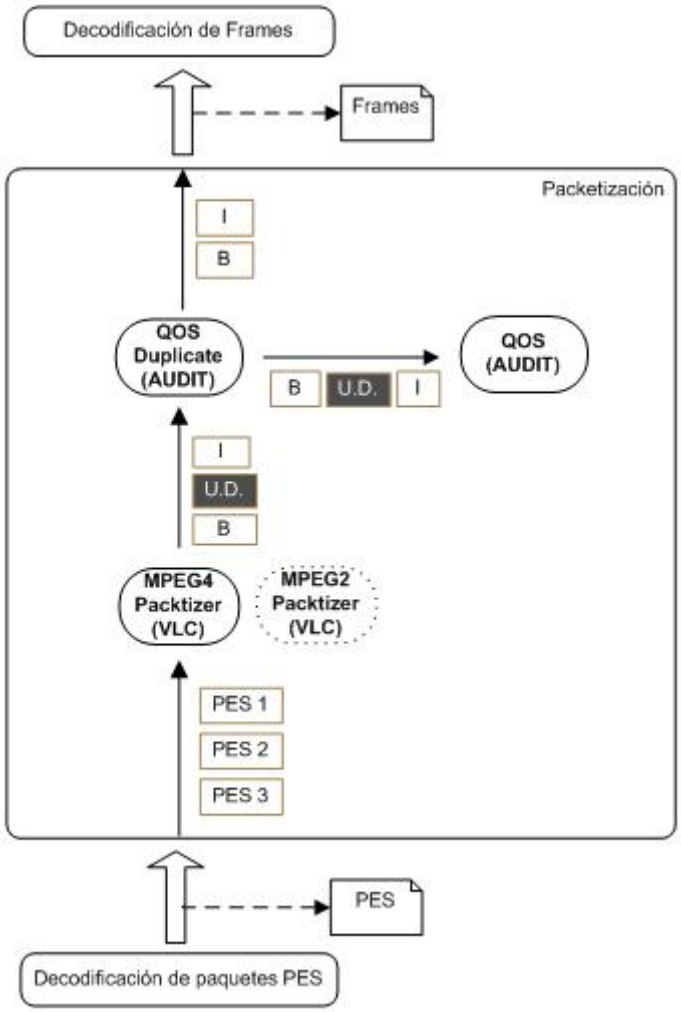


Figura 44: Proceso de packetizacion

3.7. Ventajas de la Arquitectura

Esta arquitectura presenta una gran cantidad de ventajas entre las que se encuentran los siguientes puntos:

Diferentes Ejecuciones del VLC

Esta arquitectura nos permite brindar diferentes tipos de posibles ejecuciones del VLC, entre las que se encuentran el poder ejecutar el VLC sin ninguno de nuestros módulos (es decir ejecutar la versión original del VLC):

./vlc

También es posible ejecutar el VLC con todas las funcionalidades desarrolladas (es decir logueando los diferentes eventos y reportando mediante SNMP), con el siguiente comando:

```
./vlc –extraintf qos
```

O ejecutar el VLC únicamente logueando los eventos (pero sin reportar ninguna información mediante SNMP):

```
./vlc –extraintf loggerAudit
```

En este caso únicamente los módulos QOS y el LoggerAudit serán instanciados al iniciar al VLC

Y finalmente ejecutar el VLC únicamente reportando mediante SNMP (en este caso sin loguear ninguna información):

```
./vlc –extraintf snmp
```

En donde únicamente se instancian los módulos QOS y SNMP.

Mantenibilidad

Como se puede observar toda la lógica de la solución se encuentra concentrada principalmente en los 3 módulos del núcleo, en donde a su vez está claramente separadas las responsabilidades de cada módulo (alta cohesión en cada módulo), lo que lleva a que posibles errores sean fácilmente corregidos y nuevos cambios sean sencillos de realizar.

Por ejemplo el agregar soporte para un nueva especificación (como por ejemplo soporte para H264) solo habría que modificar el packetizer H264 sin tener que modificar el resto de los módulos, lo cual claramente es una gran ventaja.

Con esto damos por cerrado el capítulo de Arquitectura de la Solución, por más detalles respecto a la arquitectura interna del VLC, ver el ANEXO I
, donde se presenta el resultado del proceso de ingeniería inversa realizado sobre el código del VLC, dado que éste no contaba con una documentación actualizada que se ajustara a las últimas versiones del VLC (sino que todo lo contrario las documentaciones existentes se encontraban complemente desactualizadas).

4. Evaluación de la Solución

4.1. Objetivo

El objetivo de las pruebas es garantizar la correctitud de la solución implementada. También se busca obtener una respuesta adecuada del sistema ante un escenario más real, con varios clientes reproduciendo muchos videos al mismo tiempo.

4.2. Pruebas de Correctitud

4.2.1. Pruebas realizadas

Para el desarrollo del testing se siguió la siguiente metodología: primeramente se estableció un conjunto de secuencias base y un conjunto de parámetros referidos a la codificación de una secuencia (como por ejemplo: codec, bitrate, etc.). En base a estos dos conjuntos se generó el conjunto resultante de secuencias de prueba, sobre el cual se definieron todos los casos de pruebas a realizar.

Cada caso de prueba básicamente está compuesto por:

- una secuencia de prueba
- un protocolo de streaming
- un escenario de pérdida (los cuales son definidos a continuación)
- una configuración de pérdida.

Finalmente la idea es contrastar las mediciones realizadas mediante el módulo QOS del VLC contra la configuración definida en cada caso de prueba.

Para las pruebas fue utilizado el Servidor de Estadísticas, cuyo manual de usuario se encuentra en el el ANEXO VII
.

4.2.1.1. Secuencias de prueba

Se han generado 16 secuencias de prueba, para las cuales se fijaron los siguientes parámetros:

- **Secuencia base**: Se establecieron 2 secuencias bases una de 6 minutos 53 segundos de duración y otra de 1 minuto 41 segundos.

- **Especificación MPEG**: Si bien el foco de las pruebas es sobre mpeg2 y mpeg4, nuestro principal énfasis es en MPEG4, dada la importancia que esta

tomando hoy en día, debido a las altas calidades que se logran con bajas codificaciones y la posibilidad de protección de derechos de autor.

Tipo de GOP: Para el caso de MPEG2 se definió un GOP de tamaño fijo, con una cantidad preestablecida de frames B.

Para el caso de MPEG4 solo se definió el tamaño máximo de GOP. En la tabla 1 se muestra el tipo de GOP para cada secuencia. Se utiliza la siguiente nomenclatura en lo que resta del capítulo: en MPEG2 los GOPs tienen la siguiente sintaxis: gopX_bY, donde X es el tamaño del GOP(que es fijo) e Y es la cantidad de frames B que aparecen en forma consecutiva. En MPEG4 la sintaxis de los GOP es: max_iX, donde X es la distancia máxima que puede haber entre dos frames I consecutivos.

- **Bitrate de Video**: Se establecieron dos diferentes bitrates, debido a las diferentes capacidades de ancho de banda que se presentan actualmente en los accesos hogareños a Internet.

Las secuencias generadas son:

Nombre de archivo	Bitrate	Versión MPEG	GOP	Video
seq1_512kb_gop12_b3.mpg	512	MPEG 2	gop12_b3	seq1
seq1_512kb_gop6_b2.mpg	512	MPEG 2	gop6_b2	seq1
seq1_512kb_max_i300.mp4	512	MPEG 4	max_i300	seq1
seq1_512kb_max_i150.mp4	512	MPEG 4	max_i150	seq1
seq1_1024kb_gop15_b2.mpg	1024	MPEG 2	gop15_b2	seq1
seq1_1024kb_gop16_b3.mpg	1024	MPEG 2	gop16_b3	seq1
seq1_1024kb_max_i350.mp4	1024	MPEG 4	max_i350	seq1
seq1_1024kb_max_i50.mp4	1024	MPEG 4	max_i50	seq1
seq2_512kb_gop15_b2.mpg	512	MPEG 2	gop15_b2	seq2
seq2_512kb_gop16_b3.mpg	512	MPEG 2	gop16_b3	seq2
seq2_512kb_max_i350.mp4	512	MPEG 4	max_i350	seq2
seq2_512kb_max_i50.mp4	512	MPEG 4	max_i50	seq2
seq2_1024kb_gop12_b3.mpg	1024	MPEG 2	gop12_b3	seq2
seq2_1024kb_gop6_b2.mpg	1024	MPEG 2	gop6_b2	seq2
seq2_1024kb_max_i150.mp4	1024	MPEG 4	max_i150	seq2
seq2_1024kb_max_i300.mp4	1024	MPEG 4	max_i300	seq2

Tabla 2: Secuencias generadas

4.2.1.2. Protocolos de Streaming

El VLC permite trabajar con un conjunto importante de protocolos tales como UDP, HTTP, HTTPS, FTP, RTP, RTSP y MMS. Las pruebas fueron realizadas en UDP y HTTP.

4.2.1.3. Escenarios de pérdida

Pérdidas en Red

Uno de los escenarios es la pérdida de red, en donde tenemos una red compuesta por tres equipos (como se puede observar en la Figura 45). El equipo A es el servidor de streaming, el equipo C el cliente y el equipo B que funciona como router. El router se configura con porcentajes de pérdida de paquetes de red, delays y anchos de banda, para poder simular de forma simplificada una red real.

En el router se instaló el módulo provisto por el kernel de linux, llamado NetEm.

Para obtener información completa de cómo instalar la arquitectura anteriormente descrita, se puede consultar al ANEXO V

de Arquitectura del Ambiente de Desarrollo.

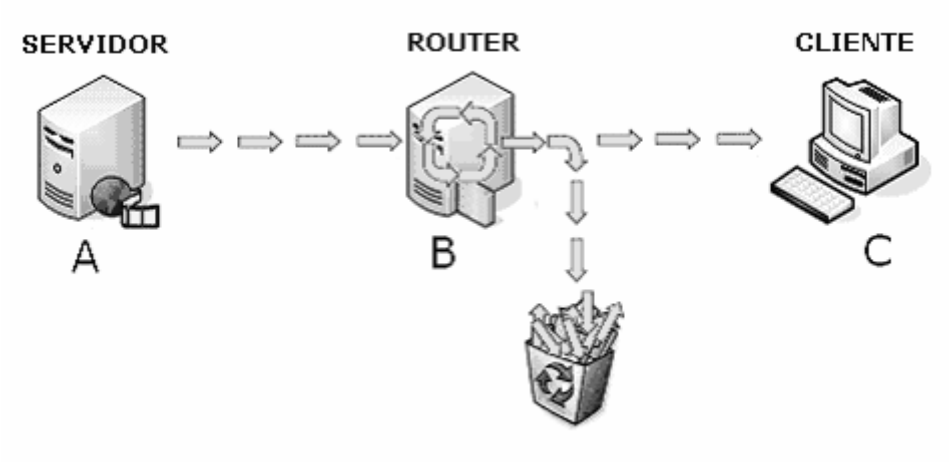


Figura 45: Pérdidas de red

Pérdidas en el Servidor

A diferencia del escenario anterior, en el cual las pérdidas son a nivel de paquetes de red en este caso las pérdidas son a nivel de frames de video y estas se generan descartando frames en el servidor. Estas pérdidas se especifican mediante un archivo de configuración en donde se indica que frame descartar y que frame no.

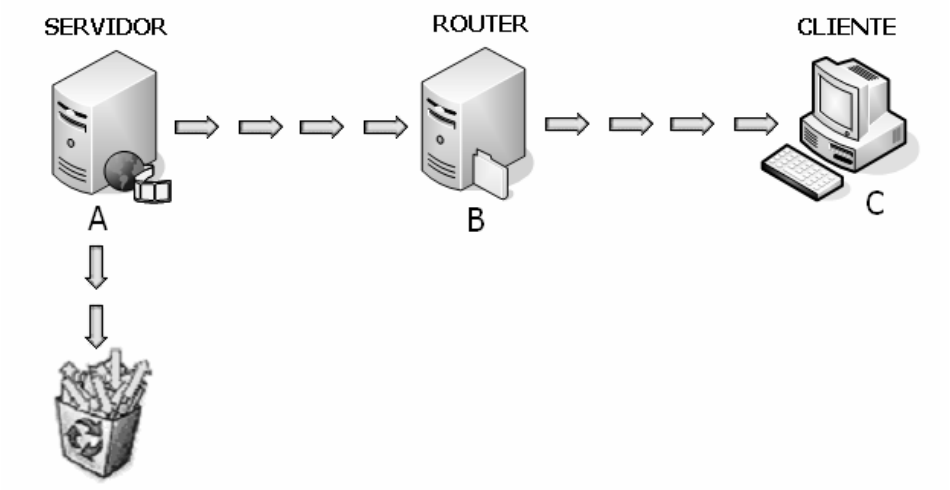


Figura 46: Pérdidas en el servidor

4.2.1.4. Configuraciones de pérdida

En todos los casos de prueba se fijó un delay de 200ms en la red.

Pérdidas en Red

Los distintos porcentajes de pérdida, varían según el protocolo y el bitrate (en el caso de UDP la pérdida es igual para ambos bitrates). A continuación se presentan las configuraciones definidas.

Nombre	Protocolo	Bitrate (Kbps)	Bandwidth (Kbps)	% de pérdida de paquetes
CfgRed1	HTTP	512	717	1
CfgRed2				3
CfgRed3				4
CfgRed4				5
CfgRed5				6
CfgRed1	HTTP	1024	1280	1
CfgRed2				3
CfgRed3				5
CfgRed4				6
CfgRed5				7
CfgRed1	UDP	512	717	3
CfgRed2				7
CfgRed3				10
CfgRed4				15
CfgRed5				20
CfgRed1	UDP	1024	1280	3
CfgRed2				7
CfgRed3				10
CfgRed4				15
CfgRed5				20

Tabla 3: Configuraciones de pérdidas en Red

Pérdidas en el Servidor

Se tienen varias configuraciones para descartar los frames. Las mismas fueron generadas según Gilbert (ver [Ref35]), con los siguientes parámetros:

Nombre	% perdida de frames	Tamaño promedio de ráfaga de frames
CfgSrv1	10	2
CfgSrv2	30	2
CfgSrv3	20	10
CfgSrv4	30	30

Tabla 4: Configuraciones de pérdidas en Servidor

En este caso no se restringió el ancho de banda de la red, dado que el foco de las pérdidas se encuentra en el servidor.

4.2.2. Resultados

Los resultados de la ejecución de los casos de pruebas serán presentados por escenario de pérdida, especificación de codificación, bitrate agrupados por video y tipo de GOP.

4.2.2.1. Pérdidas de red

MPEG4

Bitrate 512

Protocolo	Configuración	% de pérdida de paquetes	% perd. en I	% perd. en P	% perd. en B	% perd.Total de frames
UDP	CfgRed1	3	32.1	29.0	28.5	28.9
	CfgRed2	7	27.3	27.8	27.1	27.6
	CfgRed3	10	28.3	26.4	26.0	26.4
	CfgRed4	15	25.3	21.1	20.1	20.9
	CfgRed5	20	27.9	24.0	23.3	23.9
HTTP	CfgRed1	1	34.3	32.8	32.0	32.5
	CfgRed2	3	32.2	32.2	31.5	31.9
	CfgRed3	4	37.9	35.7	34.5	35.3
	CfgRed4	5	32.9	33.4	32.7	33.1
	CfgRed5	6	47.2	46.8	46.2	46.6

Tabla 5: Resultados pérdidas en Red – MPEG 4 – Bitrate 512

Observando los resultados presentados en la Tabla 5, existe un caso que nos llamó mucho la atención, el mismo se corresponde con el protocolo UDP, en donde se registran pérdidas de mayor tamaño ante porcentajes de pérdida menores, con un porcentaje de pérdida de un 3% de paquetes de red tenemos una pérdida promedio del 28,9% de los frames mientras con una pérdida del 15% tenemos una pérdida del 20,9 % de frames.

Para ahondar más en este caso presentaremos unos gráficos con la información recogida durante la ejecución de un caso de prueba.

En la siguiente gráfica (Figura 47), presentamos el caso de prueba correspondiente con la siguiente configuración:

Secuencia: seq1_512kb_max_i150.mp4
Protocolo de Streaming: UDP
Escenario: pérdidas en red
Configuración: CfgRed1 (3%)

Como se puede observar, las pérdidas se concentran en picos muy altos, de hasta 150 frames. En esos picos el video queda completamente detenido durante unos segundos volviendo luego a la normalidad.

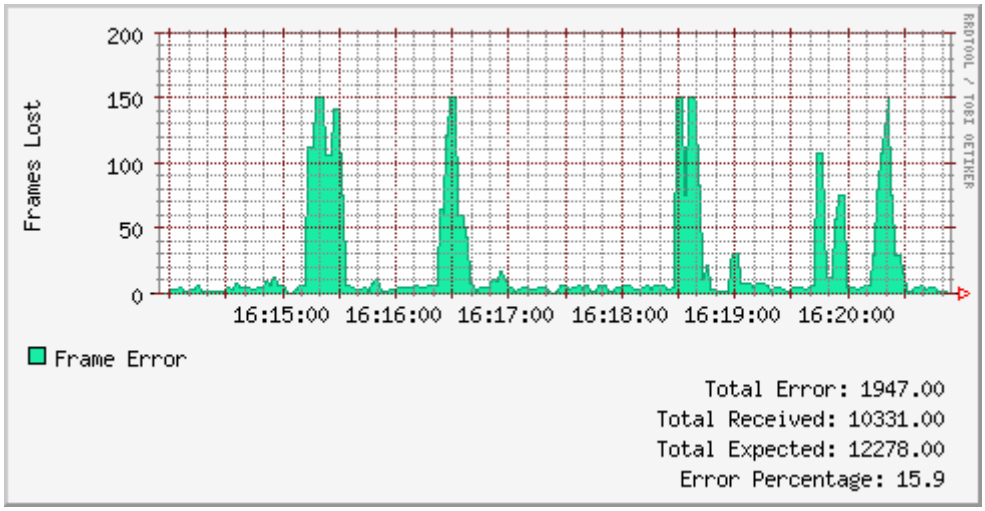


Figura 47: Frames Lost – UDP, seq1_512kb_max_i150.mp4, CfgRed1

En cambio bajo las mismas condiciones pero en lugar de la configuración CfgRed1 usando la configuración CfgRed4 (porcentaje de pérdida de 15%), podemos observar una pérdida de frames mucho más uniforme (mucho más distribuida que en la Figura 47). Basta comparar el pico más alto de la gráfica anterior (de 150 frames) con el de la Figura 48 (de 130 frames), para notar dicha diferencia.

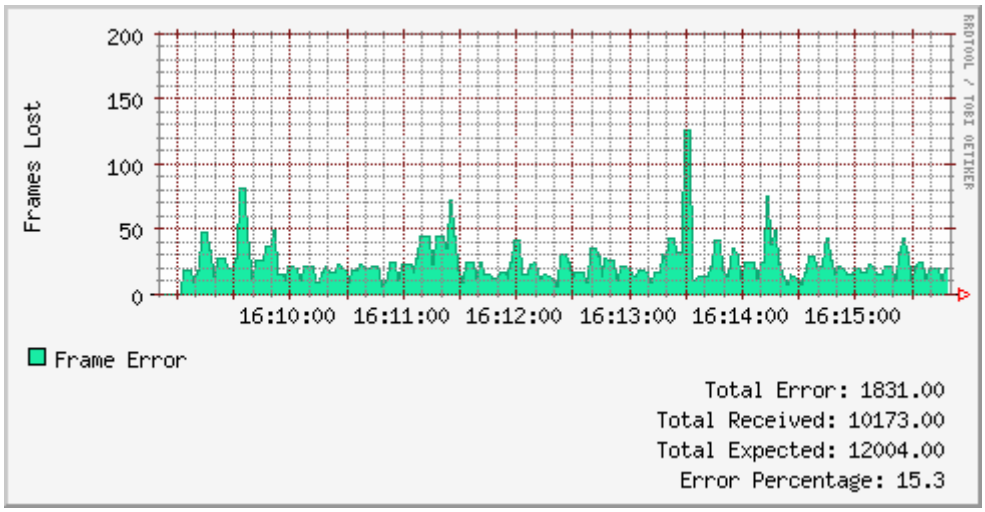


Figura 48: Frames Lost – UDP, seq1_512kb_max_i150.mp4, CfgRed4

Efectivamente en este caso la calidad de imagen presentada en el reproductor es considerablemente peor que en el caso anterior con la salvedad que en este caso el video en ningún momento queda congelado, razón por la cual se explica que los picos de pérdidas sean menores y la pérdida sea más uniforme.

Podemos concluir que una mayor pérdida de paquetes de red provoca una pérdida de frames más uniforme y que pérdidas bajas en la red provocan pérdidas de gran tamaño pero muy localizadas.

Bitrate 1024

Protocolo	Configuración	% de pérdida de paquetes	% perd. en I	% perd. en P	% perd. en B	% perd.Total de frames
UDP	CfgRed1	3	24.5	20.2	18.7	19.8
	CfgRed2	7	26.6	17.2	15.3	16.7
	CfgRed3	10	24.9	18.1	16.1	17.5
	CfgRed4	15	17.2	18.5	17.4	17.9
	CfgRed5	20	22.7	21.2	22.0	21.5
HTTP	CfgRed1	1	24.1	16.7	15.4	16.4
	CfgRed2	3	25.9	18.9	17.8	18.6
	CfgRed3	5	29.4	22.9	22.0	22.7
	CfgRed4	6	36.3	31.9	31.0	31.7
	CfgRed5	7	33.0	33.9	33.0	33.5

Tabla 6: Resultados pérdidas en Red – MPEG 4 – Bitrate 1024

Las conclusiones a las que llegamos en este caso coinciden completamente con el caso anterior con la salvedad que los porcentajes de pérdidas son menores en este caso.

A continuación profundizaremos acerca de la diferencia en el porcentaje de pérdidas dependiendo del bitrate de video.

Compararemos la secuencia seq1_512kb_max_i150.mp4 con la secuencia seq1_1024kb_max_i50.mp4. Sabemos que las cantidades de frames de ambas secuencia son muy similares (aproximadamente 12500 frames cada una) por lo que nos centraremos en el tamaño de los frames en las diferentes secuencias:

Para la secuencia seq1_512kb_max_i150.mp4 (ejecutada sobre UDP, pérdidas en la red y configuración CfgRed1), tenemos las siguientes gráficas referidas al tamaño de sus frames:

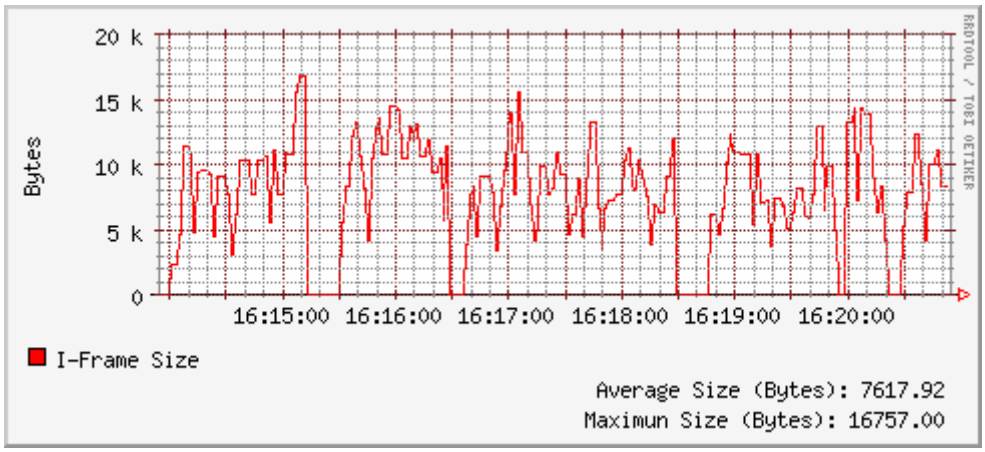


Figura 49: Tamaño Promedio de I-Frames

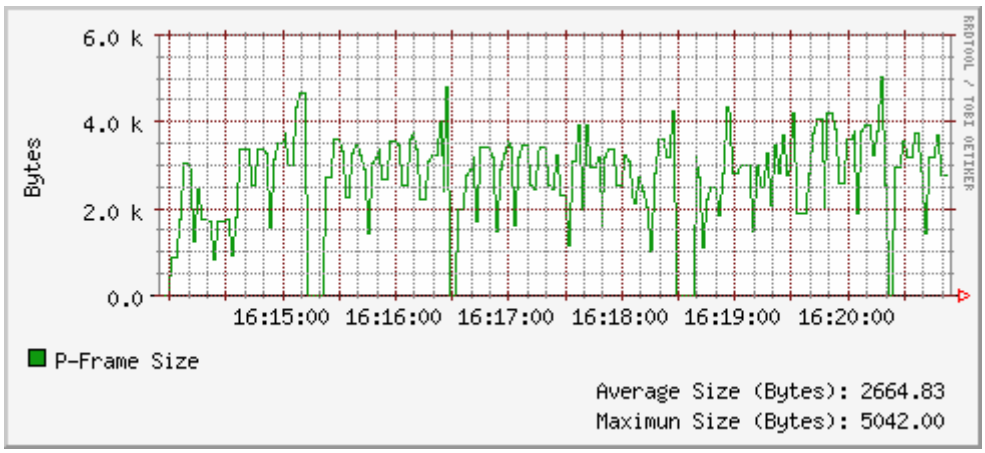


Figura 50: Tamaño Promedio de P-Frames

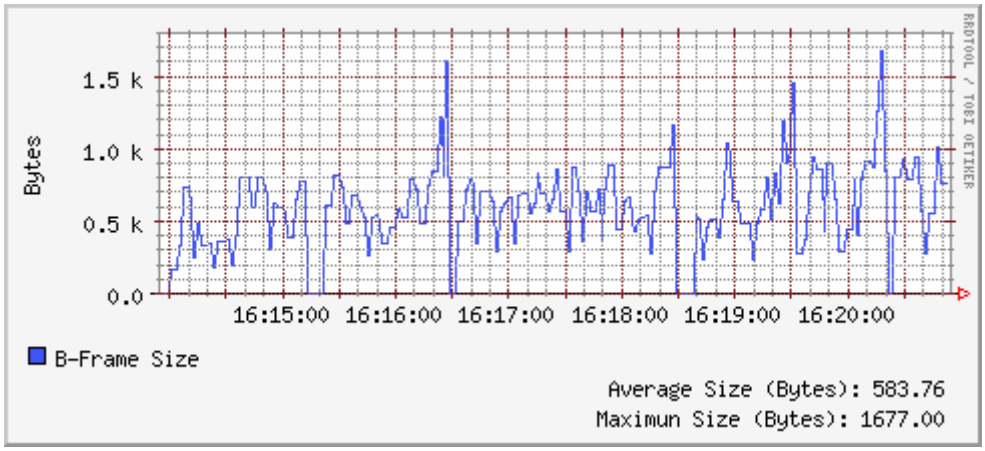


Figura 51: Tamaño Promedio de B-Frames

De las que podemos concluir que el tamaño promedio de I-Frames es de 7618 Bytes, el de P-Frames es 2665 Bytes y el de B-Frames es de 584 Bytes.

Ahora para la secuencia seq1_1024kb_max_i50.mp4 (ejecutada sobre UDP, pérdidas en la red y configuración CfgRed1), tenemos las siguientes gráficas:

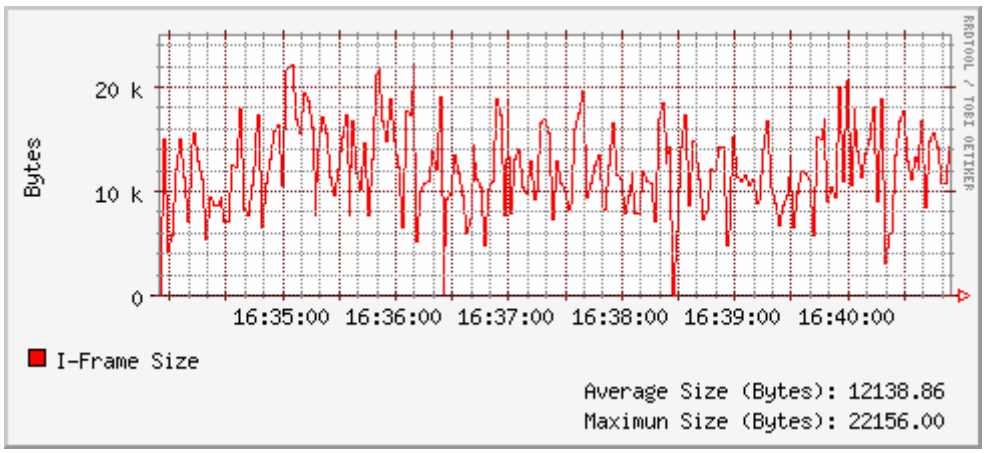


Figura 52: Tamaño Promedio de I-Frames

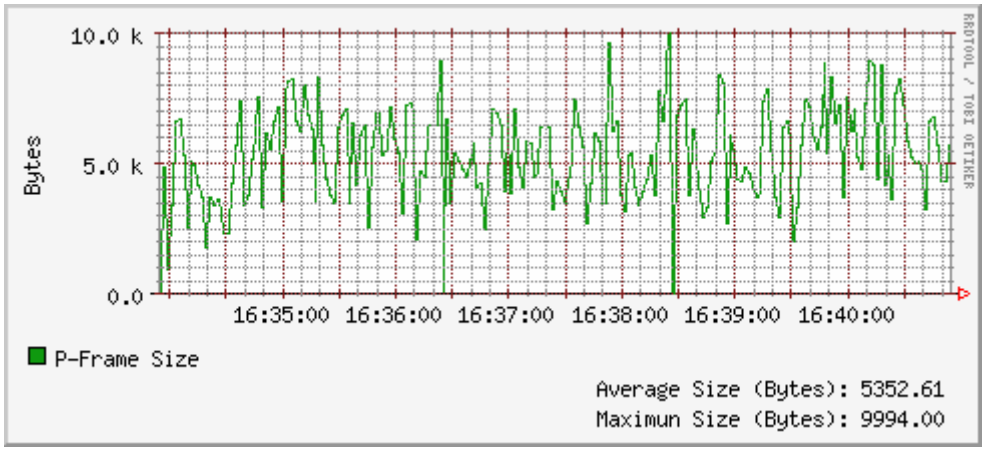


Figura 53: Tamaño Promedio de P-Frames

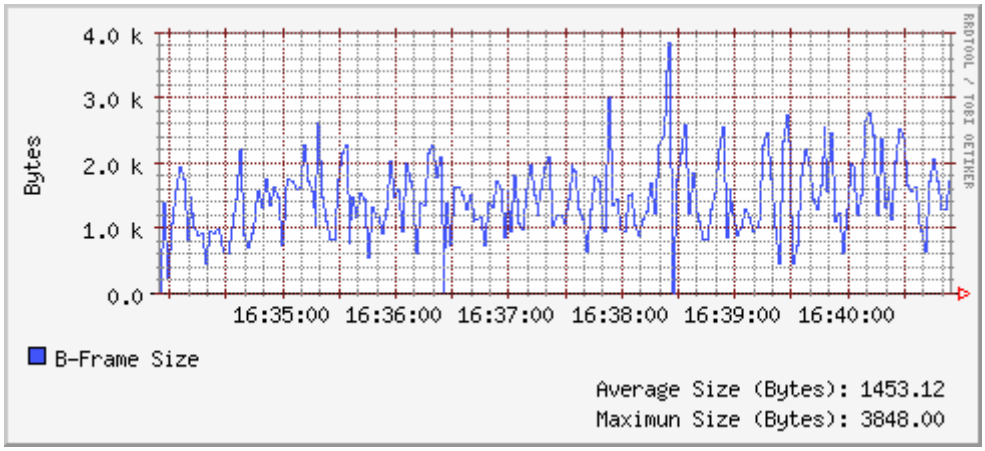


Figura 54: Tamaño Promedio de B-Frames

De las que podemos concluir en este caso que el tamaño promedio de I-Frames es de 12139 Bytes, el de P-Frames es 5353 Bytes y el de B-Frames es de 1453 Bytes.

Claramente podemos concluir que ante bitrates mayores aumenta el tamaño promedio por tipo de frame, lo cual nos lleva a la hipótesis de que ante mayor tamaño de frame tenemos menor porcentaje de pérdidas de estos.

Una posible explicación a esto es que el VLC pueda estar llevando a cabo alguna técnica de padding (rellenar los frames con información nula) en lugar de descartarlos.

MPEG2

Las pruebas realizadas sobre UDP causan la finalización anormal del VLC (Segmentation Fault), luego de un largo proceso de debugueo llegamos a la conclusión de que la falla escapa al código correspondiente con el proyecto AUDIT, dado que al repetir esta prueba sobre la versión 0.8.5 del VLC “pura” (es decir sin realizarle ninguna modificación) este se comporto de igual manera.

Debido a este problema no fue posible realizar las pruebas planificadas con pérdidas en la red sobre UDP. Las 2 tablas siguientes representan resultados obtenidos sobre HTTP.

Bitrate 512

Configuración	% de pérdida de paquetes	% pérdidas en I	% pérdidas en P	% pérdidas en B	% pérdidas Total de frames
CfgRed1	1	59.0	59.2	59.2	59.2
CfgRed2	3	59.2	59.3	59.4	59.3
CfgRed3	4	63.1	63.0	63.0	63.0
CfgRed4	5	59.9	59.9	59.9	59.9
CfgRed5	6	51.1	51.0	51.1	51.1

Tabla 7: Resultados pérdidas en Red – MPEG 2 – Bitrate 512

A continuación se presenta una gráfica que muestra la pérdida de frames en el tiempo. Como se puede observar, en los primeros instantes del video, no existe pérdida alguna (el video se ve correctamente). Llegado un punto el video queda completamente detenido hasta la finalización del mismo, lo cual claramente se puede apreciar en la Figura 56, en donde dado un instante en adelante solo se registran pérdidas de frames (frames expected = 100% y frames arrived = 0%).

Como es sabido en HTTP los paquetes perdidos son retransmitidos, y por tanto la aplicación (donde se realiza nuestra medida) recibe todos los paquetes. Los frames que vemos como pérdidas, realmente son frames que llegaron a destino pero en forma tardía, razón por la cual el VLC no los puede presentar en pantalla siendo así descartados.

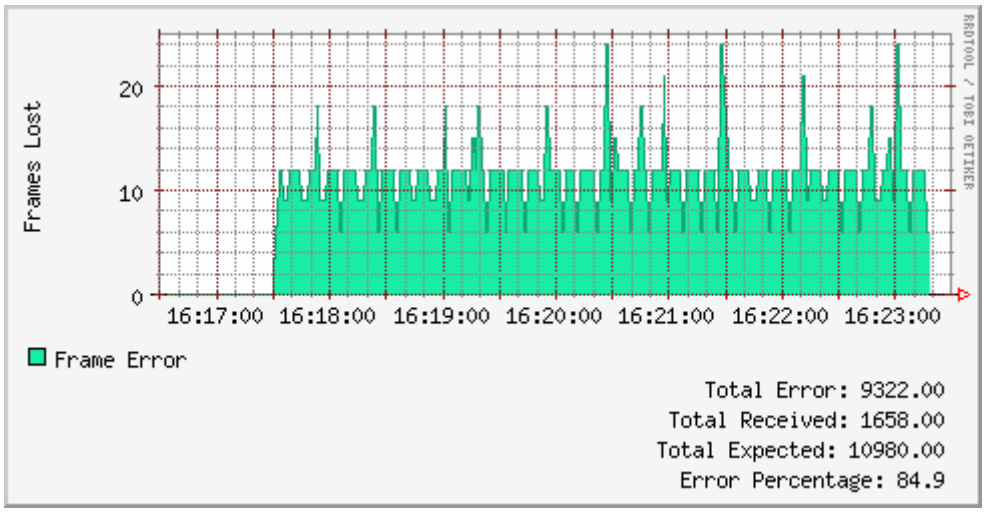


Figura 55: Frames Lost – MPEG2, http, Pérdidas en Red, BitRate 512

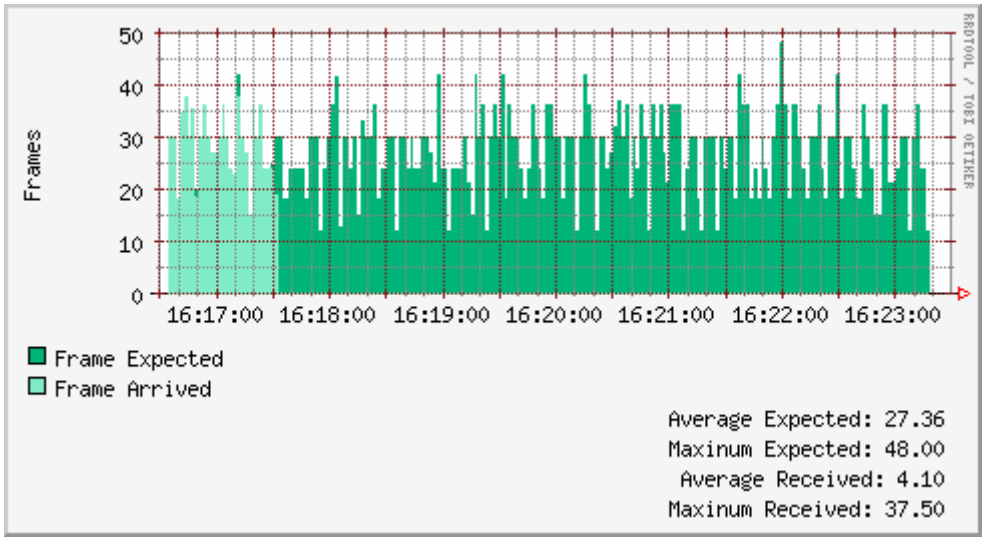


Figura 56: Frames – MPEG2, http, Pérdidas en Red, BitRate 512

Bitrate 1024

Configuración	% de pérdida de paquetes	% pérdidas en I	% pérdidas en P	% pérdidas en B	% pérdidas Total de frames
CfgRed1	1	31.3	31.4	31.4	31.4
CfgRed2	3	33.0	33.0	32.9	32.9
CfgRed3	4	34.6	34.5	34.6	34.6
CfgRed4	5	38.4	38.6	38.6	38.6
CfgRed5	6	48.8	48.7	48.7	48.7

Tabla 8: Resultados pérdidas en Red – MPEG 2 – Bitrate 1024

En la Figura 57 podemos ver un comportamiento similar al caso anterior, con la diferencia de que el tiempo en el que el video se detiene es posterior, es decir que el video se mantiene activo durante una mayor duración.

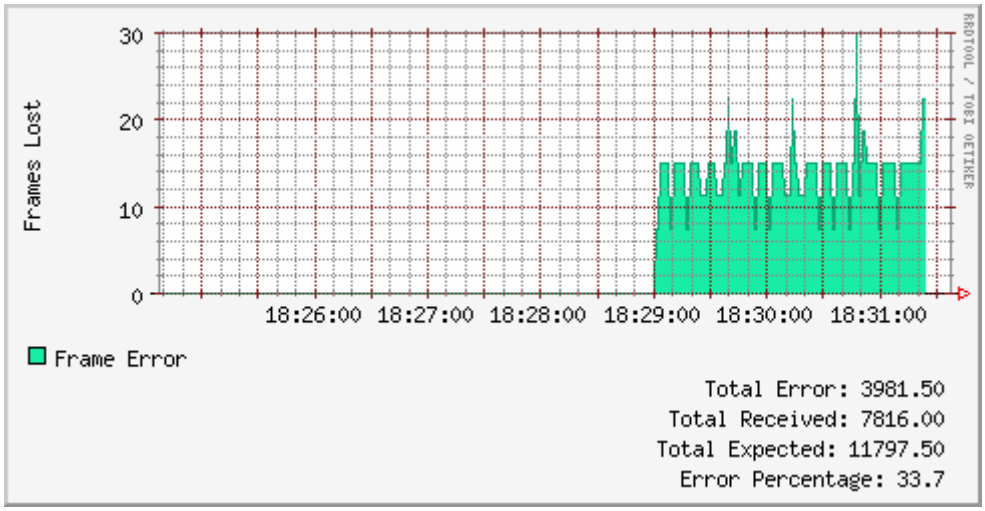


Figura 57: Frames Lost – MPEG2, http, Pérdidas en Red, BitRate 1024

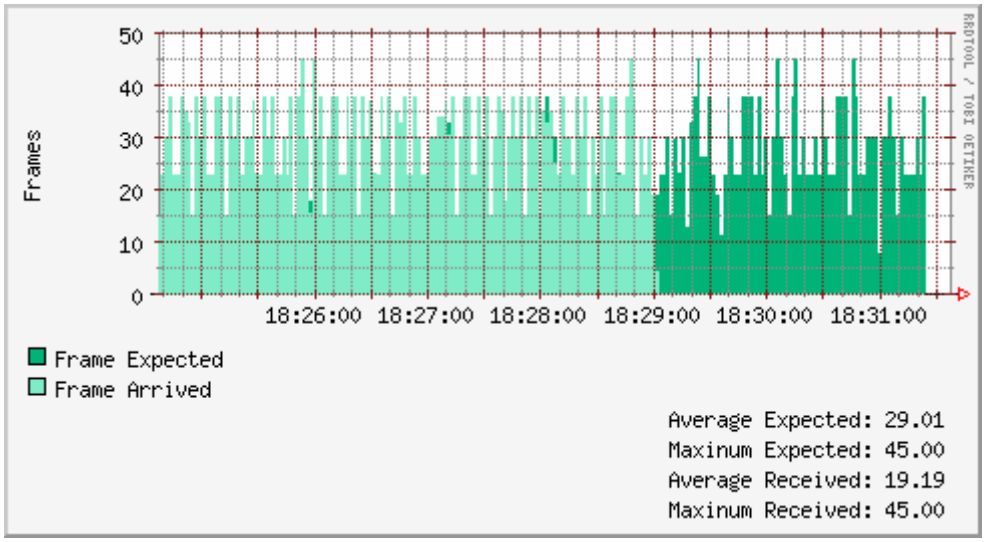


Figura 58: Frames – MPEG2, http, Pérdidas en Red, BitRate 1024

Al igual que en el caso de MPEG4 y los tamaños de frames podemos deducir que el tamaño promedio de un frame codificado a 1024 Kbps es mayor que el de uno codificado a 512 Kbps.

Lo que en este caso también podemos suponer que ante un escenario de pérdida, si el tamaño de los frames es mayor, el porcentaje de pérdidas de frames es menor por lo que la calidad del video es mejor.

4.2.2.2. Pérdidas en Servidor

MPEG4

Prot.	Config.	% perd. de frames generada	Ráfagas de perd. de frame prom. Gen.	% Total perd. de frame medido
udp	CfgSrv1	10	2	10.4
	CfgSrv2	30	2	29.1
	CfgSrv3	20	10	22.7
	CfgSrv4	30	30	31.6
http	CfgSrv1	10	2	10.6
	CfgSrv2	30	2	29.4
	CfgSrv3	20	10	22.7
	CfgSrv4	30	30	30.5

Tabla 9: Resultados pérdidas en Servidor – MPEG 4

MPEG2

Prot.	Config.	% perd. de frames generada	Ráfagas de perd. de frame prom. Gen.	% Total perd. de frame medido
udp	CfgSrv1	10	2	10.3
	CfgSrv2	30	2	29.3
	CfgSrv3	20	10	22.3
	CfgSrv4	30	30	32.4
http	CfgSrv1	10	2	10.4
	CfgSrv2	30	2	29.2
	CfgSrv3	20	10	22.5
	CfgSrv4	30	30	32.6

Tabla 10: Resultados pérdidas en Servidor – MPEG 2

Como podemos observar en la Tabla 9 y en la Tabla 10, los porcentajes de pérdida de frames son muy similares a los porcentajes de pérdida generados.

A su vez se puede observar que en las configuraciones CfgSrv1 y CfgSrv2 el error del cálculo no supera el 3% mientras que en las configuraciones CfgSrv3 y CfgSrv4 el error llega hasta el 11%. Esto se debe a que el tamaño promedio de las ráfagas en las configuraciones CfgSrv3 y CfgSrv4 es mucho mayor que en el caso de las configuraciones CfgSrv1 y CfgSrv2. Siempre existe un defasaje entre que el servidor comienza a stremear y el cliente comienza a leer dicho streaming, razón por la cual el cliente no recibe el 100% de los frames de las secuencias, es decir que pérdidas más uniformes (como en el caso de las configuraciones CfgSrv1 y CfgSrv2) existe mayor probabilidad de que el error relativo sea menor que en los casos de pérdidas esporádicas (debido a que si existe una ráfaga en el periodo en donde el cliente no esta leyendo, éste no registrara esta información).

Otro punto importante a observar es que los resultados no varían según sea el protocolo de streaming, la especificación y el bitrate, lo cual es lo esperable dado que las pérdidas son generadas en el servidor mismo.

A continuación se presenta un conjunto de gráficas mostrando las diferencias registradas entre las configuraciones CfgSrv1/CfgSrv2 y las CfgSrv3/CfgSrv4.

En la Figura 59, Figura 60 y Figura 61, se presentan las pérdidas registradas de frames I, P, B respectivamente en el siguiente caso de prueba:

Secuencia: seq1_512kb_max_i150.mp4

Protocolo de Streaming: HTTP

Escenario: Pérdidas en el servidor

Configuración: CfgSrv3 (20% de pérdidas, ráfagas de largo promedio de 10)

Como se puede observar en las figuras, las ráfagas son claramente identificables dado la existencia de picos muy definidos, los cuales se encuentran presentes tanto en la gráfica de pérdidas en frames I, como en pérdidas de frames P, como de frames B (por ejemplo en el instante 00:53:00).

En dicho instante, las pérdidas están en el entorno de los 200 frames. En una primera impresión, parecería una inconsistencia que se hayan generado pérdidas con ráfagas de 10 frames y se hallan obtenido picos de pérdidas de hasta 200 frames.

Primero hay que aclarar, que el tamaño de las ráfagas generadas es un promedio, por lo que puede haber ráfagas de distintos tamaños. Para este caso de prueba no existe ninguna ráfaga en todo el video que alcance los 200 frames. La ráfaga mayor alcanza los 89 frames.

La explicación es la siguiente. La información de las pérdidas se condensa por período, donde el período es la llegada entre un user data y el siguiente. Para el caso de prueba que estamos analizando, el período es de 150 frames. Pero cuando se pierde un paquete de tipo I, también se pierde el user data, por lo que el período es de 300 frames.

El sistema condensa toda la información de cada periodo, por lo que puede haber muchas ráfagas cortas, pero que en la gráfica se ve como una sola, provocando en este caso que en un período de 300 frames tengamos pérdidas de 200 frames.

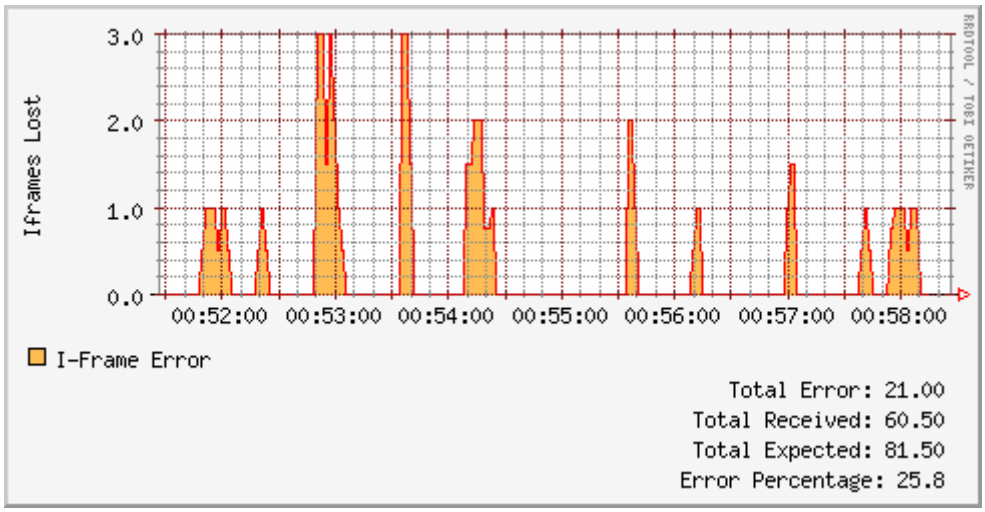


Figura 59: I Frames Lost – HTTP, seq1_512kb_max_i150.mp4, CfgSrv3

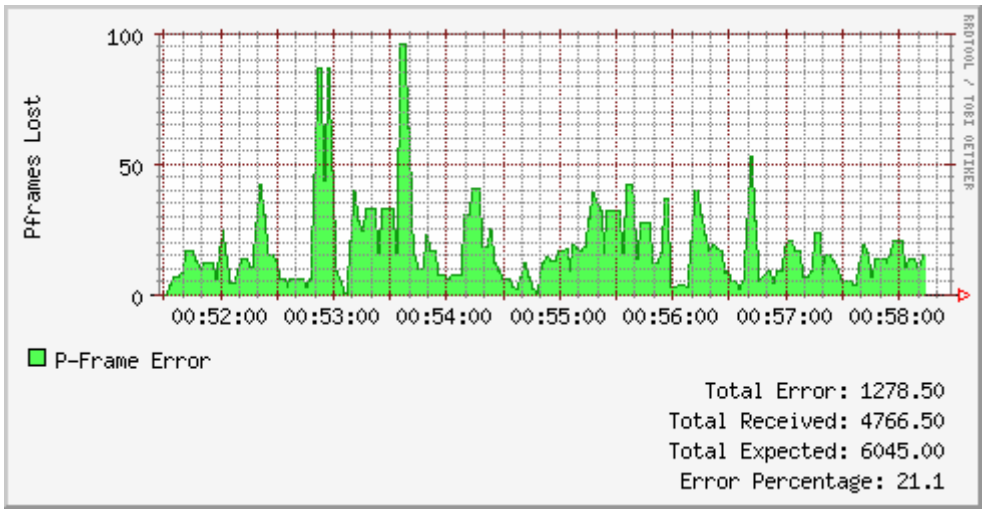


Figura 60: P Frames Lost – HTTP, seq1_512kb_max_i150.mp4, CfgSrv3

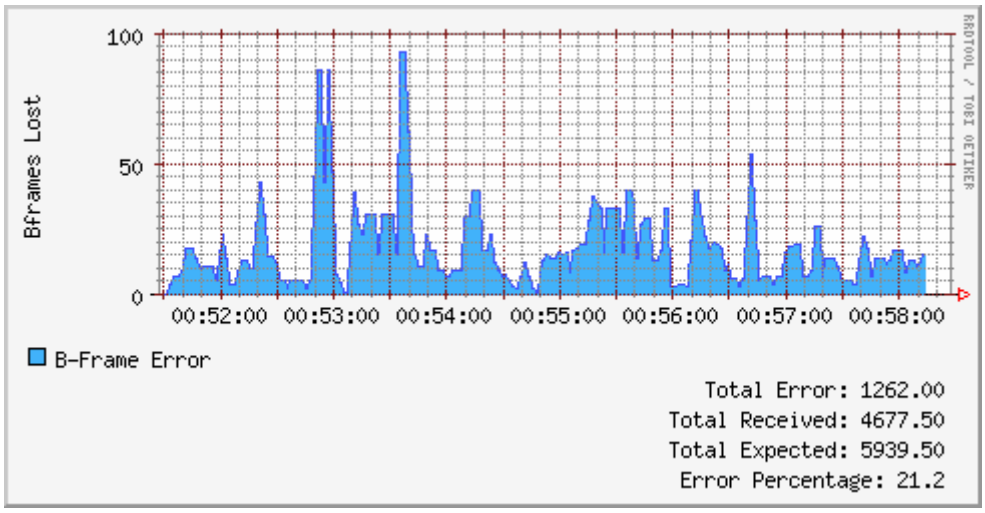


Figura 61: B Frames Lost – HTTP, seq1_512kb_max_i150.mp4, CfgSrv3

Ahora veremos estas mismas gráficas pero para el siguiente caso de prueba (Figura 62, Figura 63 y Figura 64):

Secuencia: seq1_512kb_max_i150.mp4
Protocolo de Streaming: HTTP
Escenario: Pérdidas en el servidor seq1_512kb_max_i150_http_CfgSrv1
Configuración: CfgSrv1 (10% de pérdidas, ráfagas de largo promedio de 2)

Como se puede apreciar en este caso los picos son más bajos que en el caso anterior (de 20 frames como máximo) y los mismo no trascienden a las 3 gráficas como pasaba en el caso anterior, lo cual se explica dado que el tamaño promedio de ráfagas es muy pequeño.

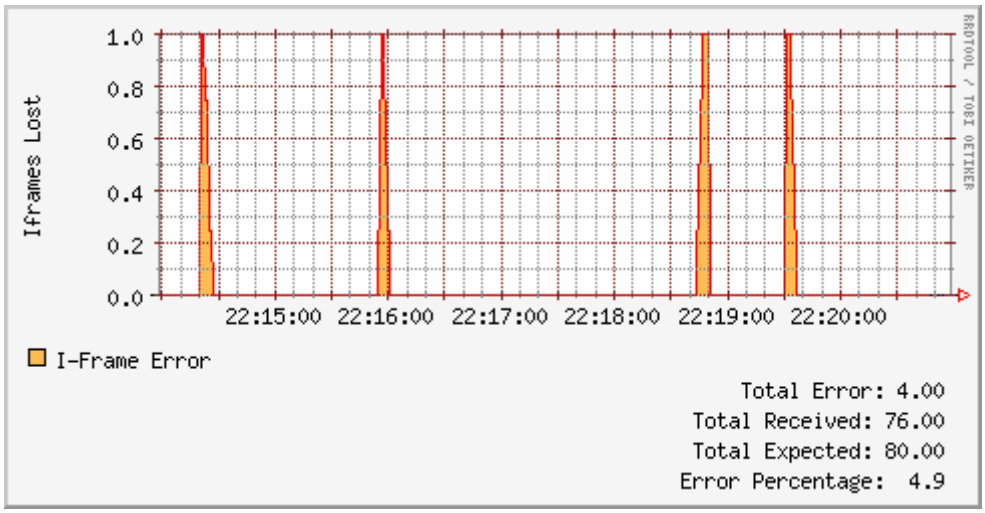


Figura 62: I Frames Lost – HTTP, seq1_512kb_max_i150.mp4, CfgSrv1

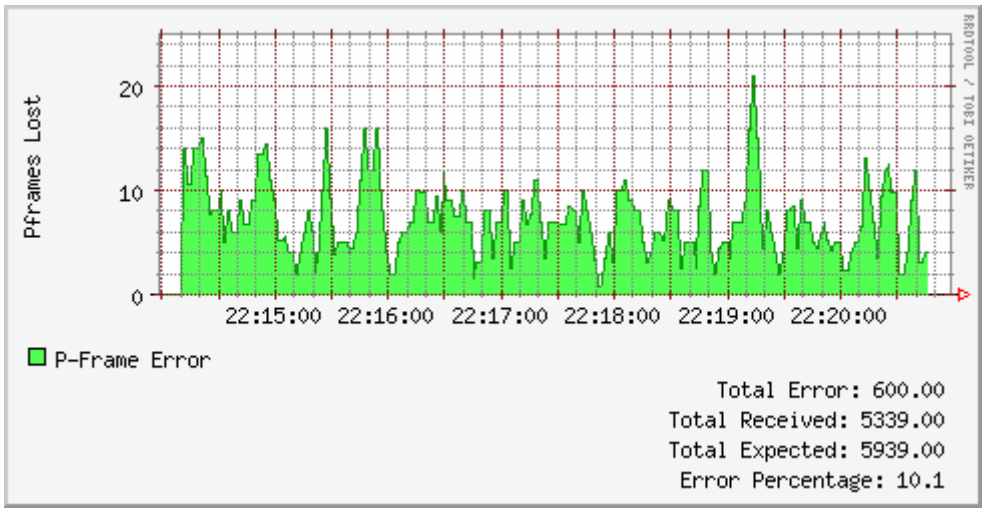


Figura 63: P Frames Lost – HTTP, seq1_512kb_max_i150.mp4, CfgSrv1

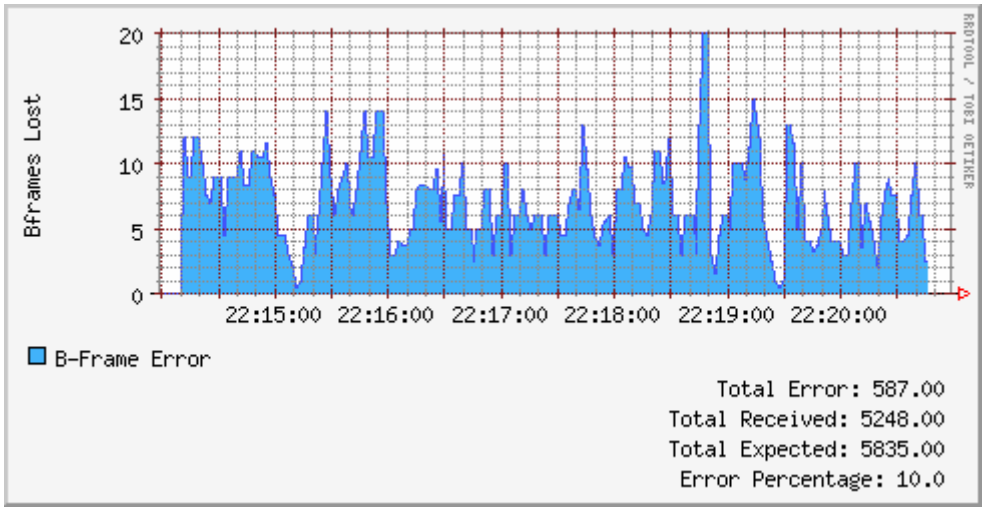


Figura 64: B Frames Lost – HTTP, seq1_512kb_max_i150.mp4, CfgSrv1

4.3. Pruebas Complejas

4.3.1. Pruebas Realizadas

Los casos de prueba que se describen en la sección 4.2 se realizaron con un Cliente con un único video. Debido a que los requerimientos son cada vez mayores en materia de streaming, se ejecutó un caso de prueba más real.

El caso de prueba tiene las siguientes características:

- Escenario de pérdida de red (Ver detalles en 4.3.1.2)
- 2 clientes
- Los clientes reproducen varios videos

4.3.1.1. Secuencias de prueba

A continuación se muestra la Tabla 11 donde se especifica que secuencias reproduce cada cliente, el protocolo usado y el tiempo de inicio de la reproducción. Para tener información detallada de las características de las secuencias ir a la sección 4.3.1.3.

Cliente	Video	Sequencia	Protocolo	Tiempo de inicio (mins)
1	1	seq1_512kb_max_i300.mp4	http	0:00
	2	seq1_1024kb_max_i50.mp4	udp	0:00
	3	seq1_1024kb_max_i350.mp4	http	0:00
	4	seq2_1024kb_max_i300.mp4	http	1:30
2	1	seq1_1024kb_max_i350.mp4	udp	0:00
	2	seq1_512kb_max_i150.mp4	udp	0:00
	3	seq1_1024kb_max_i50.mp4	http	0:00

Tabla 11: Secuencias reproducidas por los clientes

4.3.1.2. Escenarios de Pérdida

Debido a que contábamos con 2 equipos más el equipo que actúa como router (encargada de eliminar paquetes de red), los equipos tuvieron que actuar como servidor y cliente de streaming al mismo tiempo. Para poder ver más claramente, podemos ver la Figura 65.

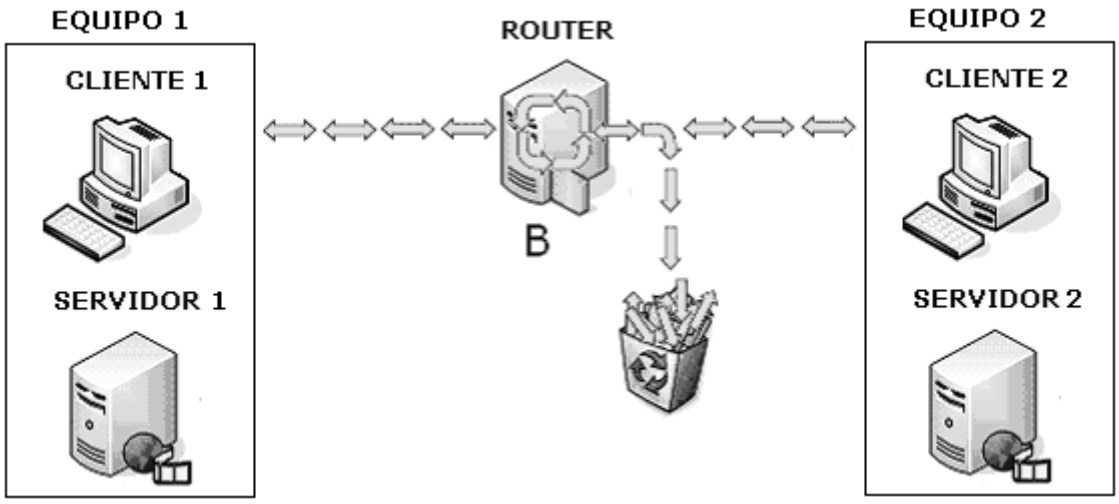


Figura 65: Arquitectura de Testing - Caso Complejo

El Cliente 1 reproduce los videos stremeados por el Servidor 2 y el Cliente 2 reproduce los videos stremeados por el Servidor 1.

4.3.1.3. Configuración de Pérdida

Los parámetros de configuración del router fueron los siguientes:

- Pérdida de red - 4%
- Ancho de banda – 3600Kbps
- Delay - 200ms

4.3.2. Resultados

Los resultados fueron los siguientes:

Cliente	Video	Sequencia	Protocolo	% de Pérdida de frames medido
1	1	seq1_1024kb_max_i350.mp4	udp	4.7
	2	seq1_512kb_max_i150.mp4	udp	4.2
	3	seq1_1024kb_max_i50.mp4	http	5.5
	4	seq2_1024kb_max_i150.mp4	udp	4.4
2	1	seq1_512kb_max_i300.mp4	http	4.1
	2	seq1_1024kb_max_i50.mp4	udp	3.8
	3	seq1_1024kb_max_i350.mp4	http	0

Tabla 12: Resultados - Caso Complejo

Como lo muestra la Figura 65 en toda la ejecución hubo 2 clientes.

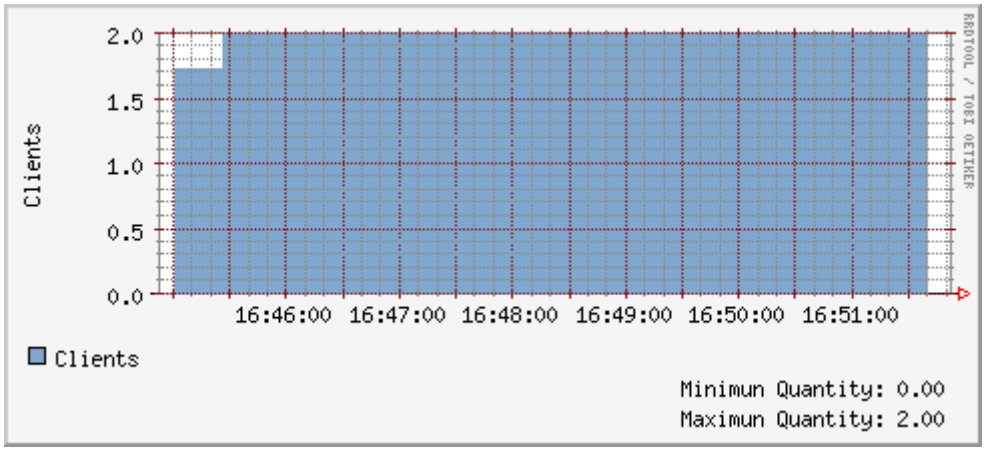


Figura 66: Clientes – Caso Complejo

4.3.2.1. Cliente 1

El Cliente 1 reproduce 3 videos que duran toda la ejecución del caso de prueba y otro video más, que comienza a reproducirse 1 minuto y medio después.
Esto lo podemos ver claramente en la siguiente figura.

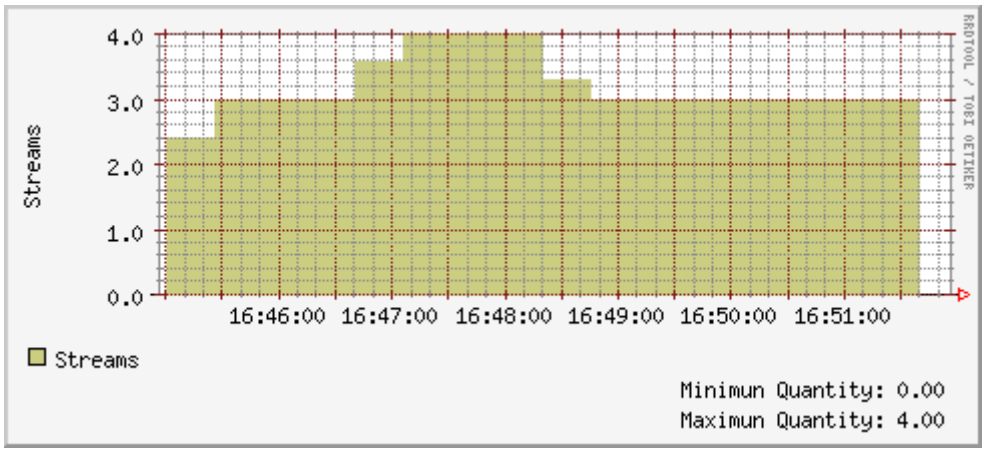


Figura 67: Streams Cliente 1 – Caso Complejo

Ahora vamos a ver las pérdidas globales discriminadas por tipo de frame en donde se condensa toda la información de todos los videos que está reproduciendo el Cliente 1.

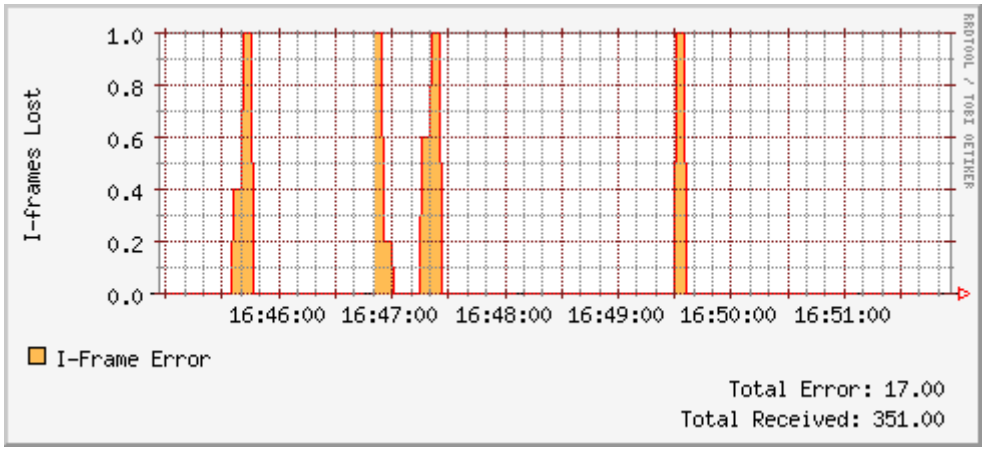


Figura 68: Frame I perdidos – Caso Complejo

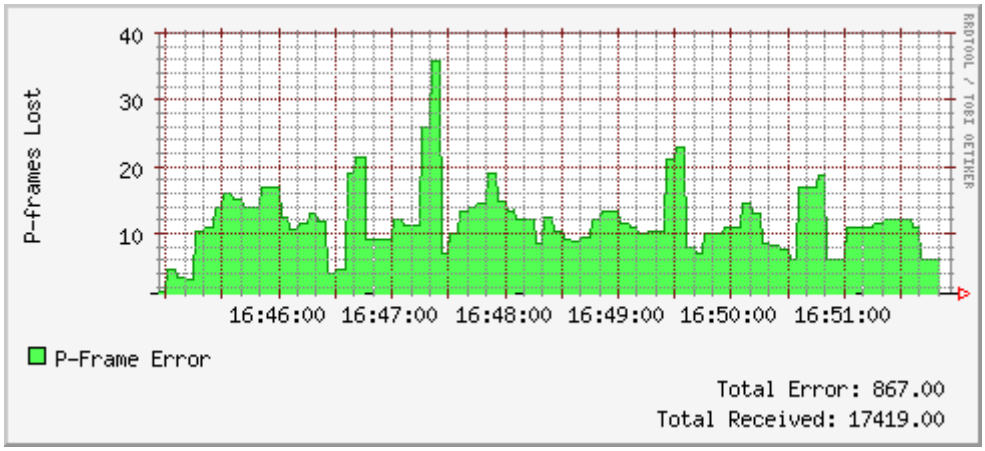


Figura 69: Frame P perdidos – Caso Complejo

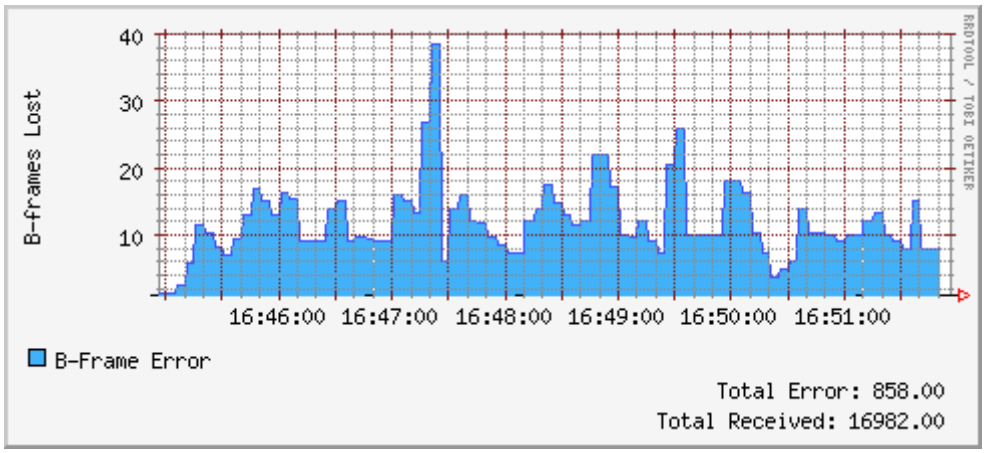


Figura 70: Frame B perdidos – Caso Complejo

Como podemos observar, durante la reproducción del video 4, se producen picos de pérdidas de frames. Esto se debe a que el consumo del ancho de banda con el

comienzo del cuarto video, superó el ancho de banda fijado en la red y las pérdidas crecieron en forma considerable en ese período.

También podemos observar en la gráfica del bitrate, que durante la duración del video 4, se observa un crecimiento del bitrate.

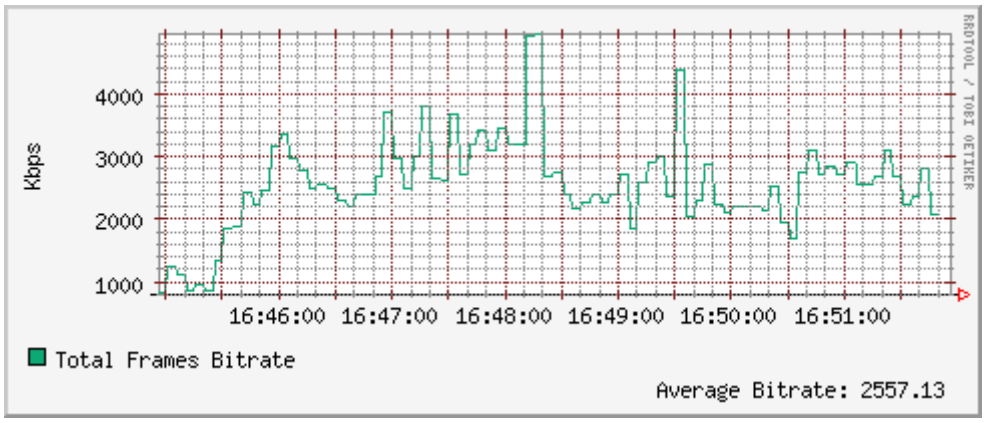


Figura 71: Bitrate total – Caso Complejo

4.3.2.2. Cliente 2

El Cliente 2 reproduce 3 videos en toda la ejecución. Esto lo podemos ver claramente en la siguiente gráfica.

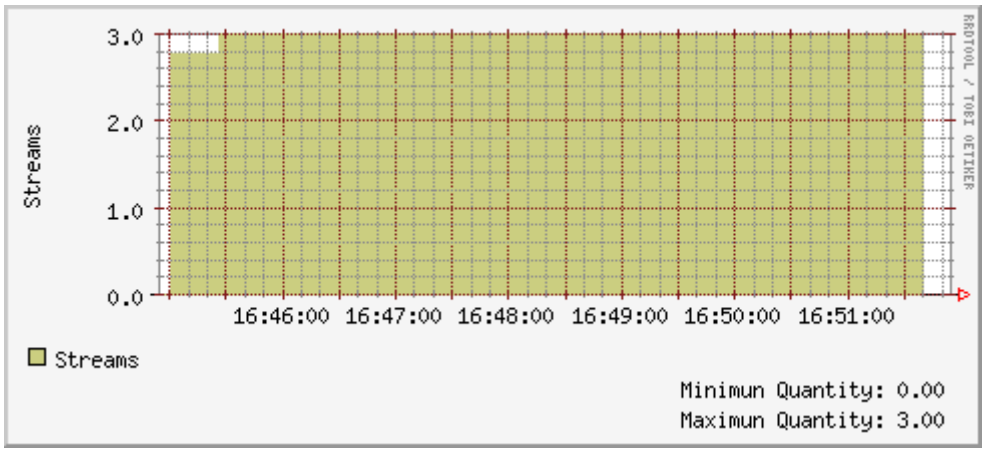


Figura 72: Streams Cliente 2 – Caso Complejo

Ahora vamos a ver las pérdidas globales discriminadas por tipo de frame en donde se condensa toda la información de todos los videos que está reproduciendo el Cliente 2.

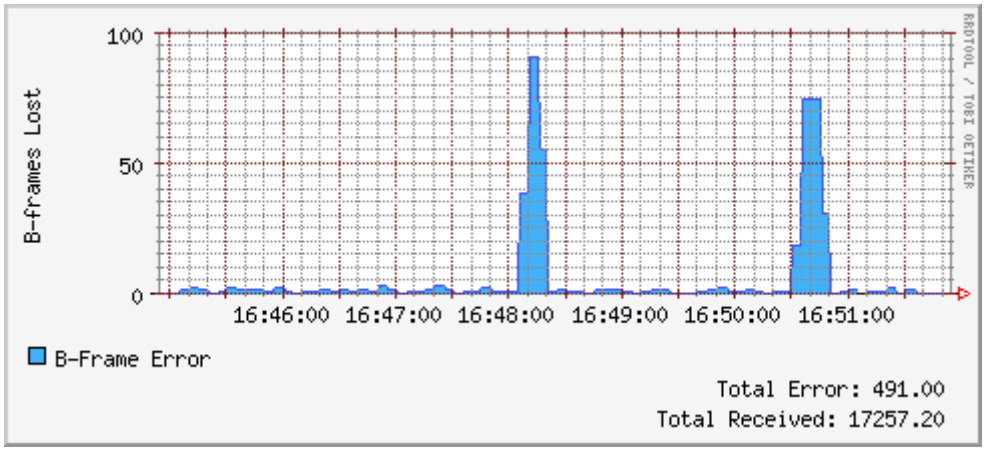


Figura 73: Frame B perdidos – Caso Complejo

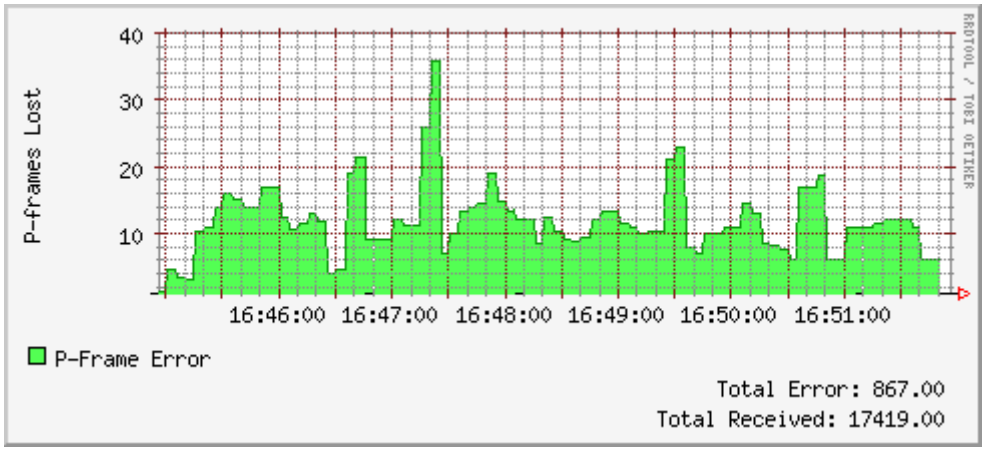


Figura 74: Frame P perdidos – Caso Complejo

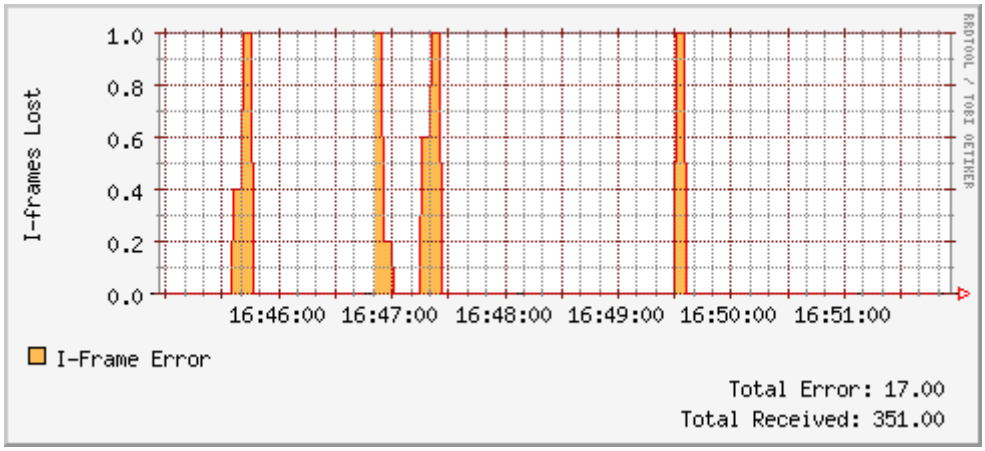


Figura 75: Frame I perdidos – Caso Complejo

Al igual que pasa con el Cliente 1, las pérdidas crecieron en forma considerable durante la reproducción del video 4 en el Cliente 1. Esto es debido a que si bien el

Cliente 1 es el que aumenta el consumo de ancho de banda, el medio físico es compartido y provoca aumento en las pérdidas de ambos cliente.

Además podemos observar otro pico que se da en el último tramo de las reproducciones. Investigando la posible causa de ese aumento considerable en las pérdidas, observamos que en el Cliente 2, en el video 3 se registró un pico importante al mismo tiempo.

A continuación veamos la gráfica de pérdidas de frames del Cliente 2 y el video 3.

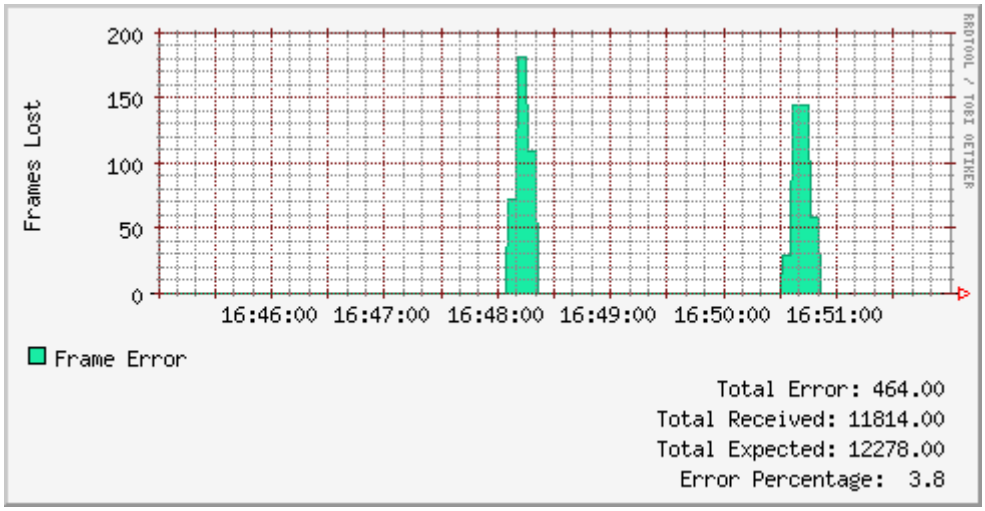


Figura 76: Frame perdidos – Cliente 2 – Video 3 – Caso Complejo

Este video fue reproducido en http. Como vimos en casos de prueba anteriores, las pérdidas en http se dan siempre con picos muy pronunciados. Y esto causa un pico de pérdidas en las gráficas globales del Cliente 2.

En el caso de la gráfica del bitrate en el Cliente 2, observamos un bitrate uniforme (a pesar de las curvas), debido a que el cliente reproduce 3 videos en toda su ejecución.

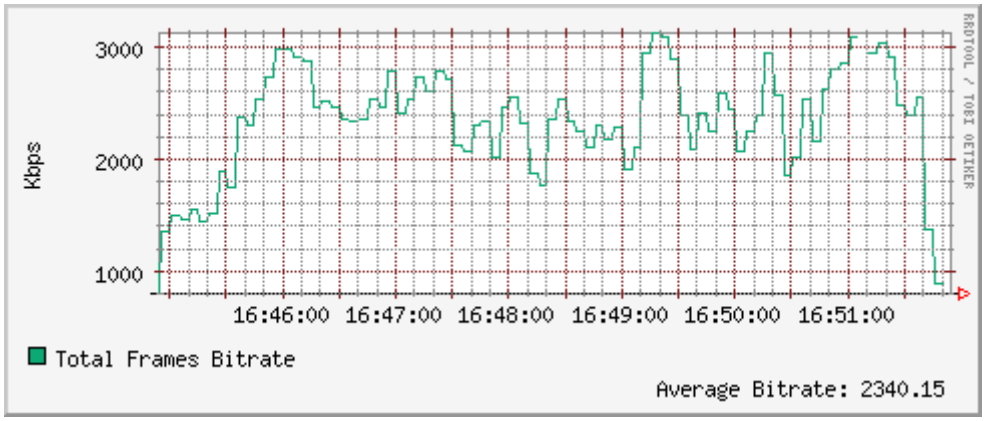


Figura 77: Bitrate total – Caso Complejo

4.3.2.3. Global

A continuación veamos las gráficas globales en donde se condensa toda la información de todos los videos que están reproduciendo todos los clientes.

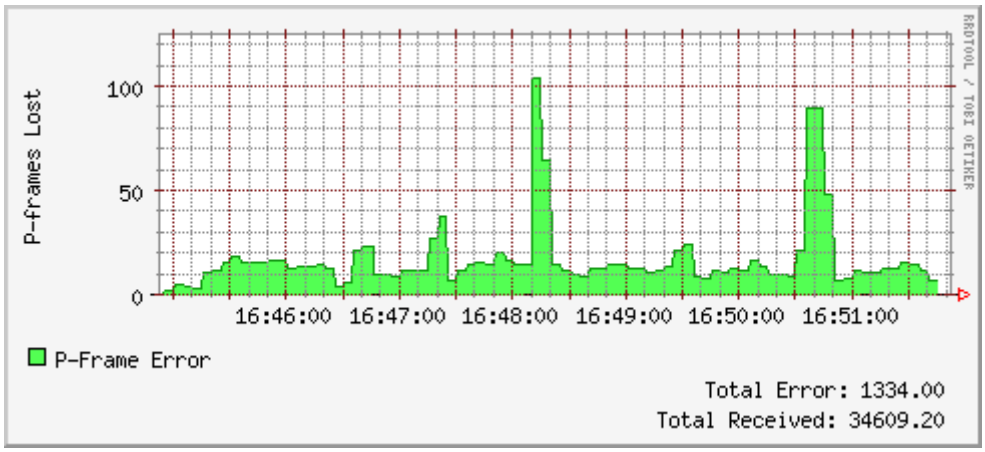


Figura 78: Global de Frames P perdidos – Caso Complejo

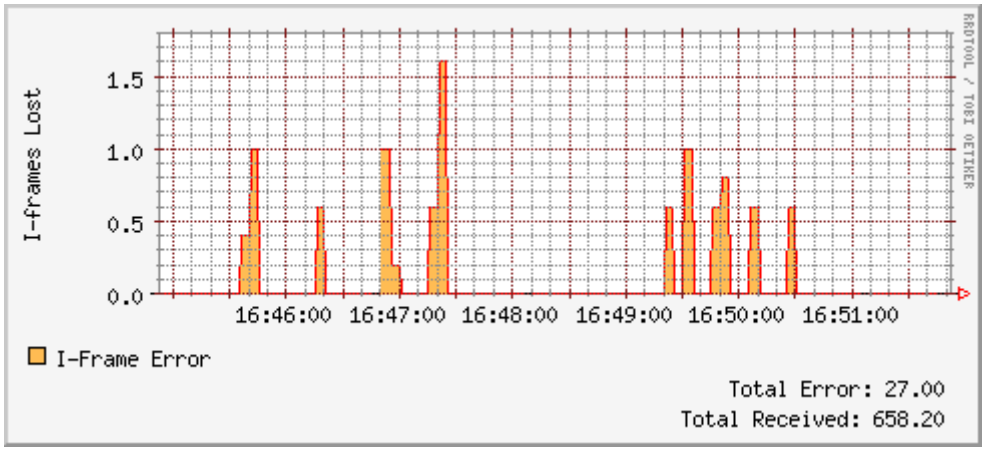


Figura 79: Global de Frames I perdidos – Caso Complejo

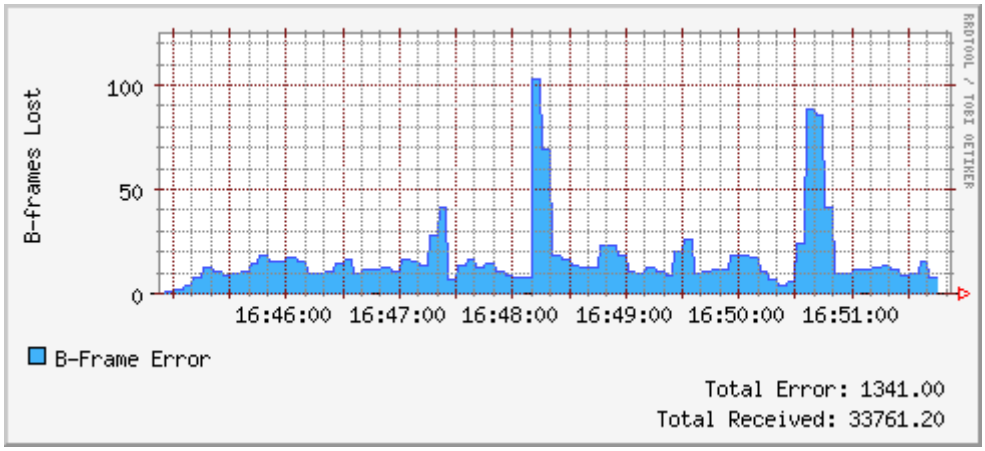


Figura 80: Global de Frames B perdidos – Caso Complejo

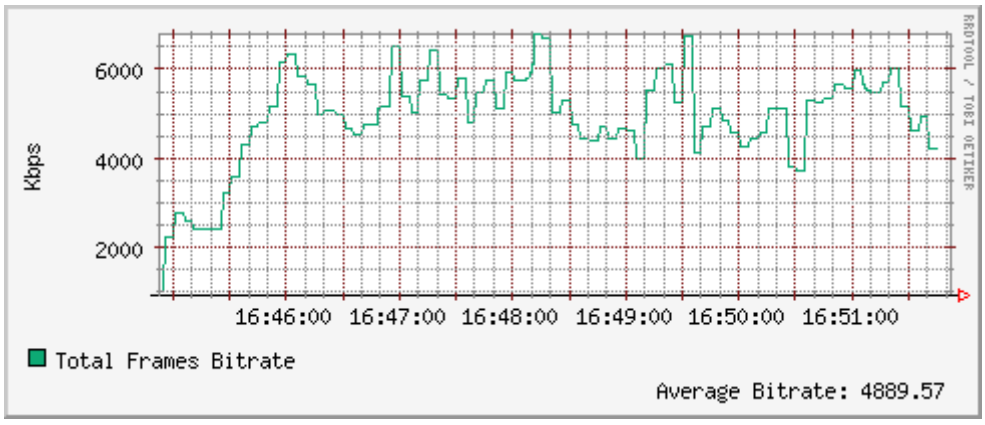


Figura 81: Global de Bitrate – Caso Complejo

Estás gráficas son muy importantes porque nos permiten ver el comportamiento general de la VDN mediante las distintas gráficas.

4.4. Conclusiones

En los escenarios de pérdida del servidor, donde nosotros sabíamos previamente el porcentaje de frames descartados, los cálculos realizados por el sistema fueron correctos, ya que el porcentaje de pérdida de paquetes calculados fue muy cercano a los porcentajes generados.

En los escenarios de pérdida de red, no pudimos establecer una relación entre el porcentaje de pérdida de red y el porcentaje de pérdida de frames, pero empíricamente podemos concluir que ante menores porcentajes de pérdida de red, la varianza en las pérdidas de frames es muy alta y ante mayores porcentajes de pérdida de red la varianza es baja.

La ejecución del caso de prueba complejo fue exitosa, debido a que el sistema respondió correctamente ante varios clientes y muchos videos.

El sistema nos permite monitorear la VDN desde lo general a lo particular. Esto es debido a que podemos monitorear desde la reproducción de un video particular de un Cliente particular hasta monitorear todos los videos de todos los Clientes en forma condensada.

5. Conclusión

El principal objetivo de este proyecto consiste en medir la calidad de video en clientes de una VDN y podemos concluir que dicho objetivo fue cumplido.

Para lograr el objetivo establecimos un conjunto de parámetros a medir sobre cada uno de los clientes. Dentro de este conjunto, los parámetros más relevantes a la hora de la medición de la calidad son los relacionadas con los frames, tales como pérdida de frames, bitrate y tamaño promedio.

Estás mediciones se podrían haber realizado a distintos niveles. A un nivel bajo se puede realizar mediciones a nivel de paquetes en la red. En principio, un determinado porcentaje de pérdida de paquetes de red no nos brinda una medida objetiva de que tan bien se está viendo el video, por lo que dicha alternativa fue descartada.

A un alto nivel se pueden medir frames. Las pérdidas de frames van a repercutir de forma mucho más directa en la calidad del video, ya que van a ser imágenes que no van a ser vistas por el usuario.

Nosotros hemos medido a nivel de frames lo cual no nos fue para nada sencillo pero de las opciones que se encontraban a nuestro alcance, sin dudas ésta es la mejor.

Este proyecto contó con una importante componente de investigación la cual sustenta la solución propuesta. Esta investigación corresponde tanto a elementos de la solución en si misma, como a investigación para el planteo del problema desde todas sus dimensiones. Se estudiaron diferentes protocolos, estándares, rfc's y otros tipo de información complementaria que nos sirvió de base para plantear las diferentes opciones.

En cuanto a la solución implementada, podemos asegurar que la arquitectura posee una alta cohesión y un bajo acoplamiento entre sus componentes, por lo que el impacto ante cambios se reduce considerablemente.

A su vez se llevó a cabo una verificación que incluyo más de 240 casos de pruebas en diferentes escenarios, usando diferentes protocolos, diferentes secuencias de video, CODECs, porcentajes de pérdidas, etc. Lo cual sumado al caso de prueba complejo (en donde existieran más de un cliente y un servidor stremeando y recibiendo más de un stream cada uno) nos garantiza de buena manera la correctitud del sistema.

Con el fin de poder brindar una clara visión sobre los resultados de la pruebas, se construyó el servidor de estadísticas, lo cual desde un principio no se encontraba completamente incluido en el alcance del proyecto. Si bien en un principio se analizó el uso de diferentes alternativas ya desarrolladas (tal como el uso de CACTI o MRTG) se optó por desarrollar una aplicación propia a nuestro proyecto la cual se adapte completamente a nuestros requerimientos.

Siglas y Abreviaturas

3GP	3rd Generation Partnership (simplicación de ISO 14496-1 Media Format)

A

AES	Audio Elementary Streams
ARP	Address Resolution Protocol (RFC 826 – http://www.ietf.org/rfc/rfc826.txt)

B

BOOTP	Bootstrap Protocol (RFC 951 – http://www.ietf.org/rfc/rfc951.txt)

C

CAT	Conditional Access Table

D

DTS	Decoding Time Stamp

E

ES	Elementary Streams

F

FIFO	First In First Out

G

GOP	Group Of Pictures – Grupo de Figuras
GSM	Global System for Mobile Communications

H

HTTP	HyperText Transfer Protocol

I

ICM	Internet Control Message Protocol (RFC 792 - http://www.ietf.org/rfc/rfc792.txt)
IGMP	Internet Group Management Protocol
IP	Internet Protocol
ISO	International Standarts Organization

J

JPEG	Joint Photographic Experts Group

L

LAN	Local Area Network (Red de área local)

M

MPEG	Moving Picture Experts Group

N

NetBEUI	NetBIOS Extended User Interface
NIT	Network Information Table

O

OSI	Open System Interface

P

PAT	Program Association Table
PC	Personal Computer
PCR	Program Clock Reference
PES	Packetized Elementary Streams
PID	Packet Identifier
PMT	Program Map Table
PS	Program Stream
PSI	Program Specific Information
PTS	Presentation Time Stamp

R

RARP	Reverse Address Resolution Protocol (RFC 903 – http://www.ietf.org/rfc/rfc903.txt)
RFC	Request For Comment
RTCP	Real-Time Control Protocol (RFC 1889 – http://www.ietf.org/rfc/rfc1889.txt)
RTP	Real-time Transport Protocol (RFC 1889 - http://www.ietf.org/rfc/rfc1889.txt)
RTSP	Real Time Streaming Protocol (RFC 2326 – http://www.ietf.org/rfc/rfc2326.txt)

S

STC	System Time Clock

T

TCP	Transport Control Protocol (RFC 793 – http://www.ietf.org/rfc/rfc793.txt)
TS	Transport Stream
TV	Televisión

U

UDP	User Datagram Protocol (RFC 768 – http://www.ietf.org/rfc/rfc0768.txt)

V

VES Video Elementary Streams

Bibliografía

[Ref1] : Wayne Brete, Mark Fimoff. *Tutorial MPEG*. [En línea]. <http://www.zenith.com/sub_hdtv/mpeg_tutorial/>. [3 de junio de 2006]

[Ref2] : David Austerberry. *The Technology of Video and Audio Streaming*. Segunda Edicion. Ed. Elsevier. 200 Wheeler Road, Burlington, MA 01803, USA.
ISBN: 0240805801.

[Ref3] : A Guide to MPEG Fundamentals and Protocol Analysis (Including DVB and ATSC). [En línea].
<http://www.broadcastpapers.com/whitepapers/paper_loader.cfm?pid=187>
[3 de junio de 2006]

[Ref4] : Protocolo de red – Wikipedia, la enciclopedia libre. [En línea].
<http://es.wikipedia.org/wiki/Protocolo_de_red>
[7 de junio de 2006]

[Ref5] : IP packet Structure. [En línea].
<http://www.freesoft.org/CIE/Course/Section3/7.htm>
[7 de junio de 2006]

[Ref6] : SCTP for Beginners. [En línea].
<http://tdrwww.exp-math.uni-essen.de/inhalt/forschung/sctp_fb/index.html>
[10 de junio de 2006]

[Ref7] : Teoría sobre los Métodos de Transmisión de Audio y Video por Internet. [En línea].
<http://www.internetmultimedia.com.mx/bloques/transmision.htm>
[10 de junio de 2006]

[Ref8] : Kevin Jeffay. *COMP 249 Advanced Distributed Systems*. [En Linea].
5/10/1999. Department of Computer Science, University of North Carolina at Chapel Hill, USA.
<http://www.cs.odu.edu/~cs778/jeffay/Lecture6.pdf>
[10 de junio de 2006]

[Ref9] : MMS streaming protocol. [En línea].
<http://www.softbytelabs.com/Support/files/MMSprotocol.doc>
[10 de junio de 2006]

[Ref10] : VLC - Features. [En línea].
<http://www.videolan.org/vlc/features.html>
[17 de diciembre de 2006]

[Ref11] : VLC media player – Wikipedia, la enciclopedia libre. [En línea].
<http://es.wikipedia.org/wiki/VLC_media_player>
[17 de diciembre de 2006]

[Ref12] : VideoLAN Streaming – Features list. [En línea].
<http://www.videolan.org/streaming-features.html>
[17 de diciembre de 2006]

[Ref13] : Wayne Brete, Mark Fimoff. *Information technology — Generic coding of moving pictures and associated audio information: Systems. Ed. Elsevier B.V.*
Second edition. [2000-12-01]

[Ref14] : estándar ISO/IEC 13818-1. [En línea].
<http://neuron2.net/library/mpeg2/iso13818-1.pdf>
[04 de enero de 2007]

[Ref15] : smi.pdf. [En línea]
<http://www.simpleweb.org/tutorials/smi/smi.pdf>
[12 de Junio de 2006]

[Ref16] : The simpleweb - MIB module validation . [En línea]
<http://www.simpleweb.org/ietf/mibs/validate/>
[12 de Junio de 2006]

[Ref17] : NET-snmp. [En línea]
<http://www.net-snmp.org/tutorial/tutorial-5/>
[12 de Junio de 2006]

[Ref18] : Comparison of media players – Wikipedia, la enciclopedia libre. [En línea].
<http://en.wikipedia.org/wiki/Comparison_of_media_players>
[03 de Enero de 2007]

[Ref19] : Métricas para el Análisis de Calidad en Servicios de Vídeo-Bajo-Demanda Reales. [En línea].
<http://cita2003.fing.edu.uy/articulosvf/19.pdf>
[04 de Enero de 2007]

[Ref20] : *IPROUTE2 Utility Suite Howto*. [En linea].
<http://www.policyrouting.org/iproute2-toc.html>. [10 de junio de 2006]

[Ref21] : Mark Lamb. *iproute2+tc notes*. [En linea].
< http://snafu.freedom.org/linux2.2/iproute-notes.htm>.
[10 de junio de 2006]

[Ref22] : Martin Devera. *HTB Linux queuing discipline manual - user guide*.
[En linea].
< http://luxik.cdi.cz/~devik/qos/htb/manual/userg.htm >.
[10 de junio de 2006]

[Ref23] : Bert Hubert. *Linux Advanced Routing & Traffic Control HOWTO*.
[En linea].
<http://lartc.org/howto/index.html>.
[10 de junio de 2006]

[Ref24] : *Netem*. [En linea].
<http://linux-net.osdl.org/index.php/Netem>.
[10 de junio de 2006]

[Ref25] : Kishor Panth. *Linux Routing*. [En linea].
<http://www.apricot.net/apricot2004/doc/IPRoute%202Basics.PDF>.
[10 de junio de 2006]

[Ref26] : Stephen Hemminger. *Network Emulation with NetEm*. [En linea].
<http://developer.osdl.org/shemminger/netem/LCA2005_paper.pdf>
[15 de Abril de 2005]

[Ref27] : Estándar MPEG-4. [En linea].
<http://pub.ufasta.edu.ar/SISD/mpeg/mpeg4.htm>.
[13 de enero de 2007]

[Ref28] : MPEG-4. [En linea].
<http://www.chiariglione.org/mpeg/standards/mpeg-4/mpeg-4.htm>.
[13 de enero de 2007]

[Ref29] : MPEG-4. [En linea].
<http://ampere.iie.edu.uy/ense/asign/codif/material/transparencias/11_mpeg4.pdf
>.
[13 de enero de 2007]

[Ref30] : Helix Service Delivery Suite. [En linea].
<http://docs.real.com/docs/rn/datasheet/Helix_SDS_S306.pdf>
[13 de enero de 2007]

[Ref31] : Helix Universal Server Administration Guide. [En linea].
<http://service.real.com/help/library/guides/helixuniversalserver/realsrvr.htm>
[13 de enero de 2007]

[Ref32] : Open Source – Server – Streaming Server. [En linea].
<http://developer.apple.com/opensource/server/streaming/index.html>
[13 de enero de 2007]

[Ref33] : QuickTime Streaming Server - Darwin Streaming Server Administrator's Guide. [En linea].
<http://developer.apple.com/opensource/server/streaming/qtss_admin_guide.pdf>
[13 de enero de 2007]

[Ref34] : Windows Media Services 9 Series New Features & Benefits. [En linea].
<http://www.microsoft.com/windows/windowsmedia/forpros/serve/features.aspx>
[13 de enero de 2007]

[Ref35] : E. Gilbert. Capacity of a burst–loss channel. Bell Systems Technical Journal, vol. 39, pp. 1253–1265.
[01 de Setiembre de 1960]

[Ref36] : VideoLAN developers – VLC Media Player. [En linea].
<http://www.videolan.org/developers/vlc.html>
[01 de febrero de 2007]

[Ref37] : Net – SNMP. [En linea].
<http://www.net-snmp.org/>
[01 de febrero de 2007]

Figuras

Tablas

ANEXOS

ANEXO I

1. Arquitectura del VLC

1.1. Introducción

Esta documentación pretende presentar una visión a alto nivel de la arquitectura interna del VLC, junto con las principales secuencias de interacción que ocurren entre los diferentes componentes del mismo.

1.2. Conceptos Generales

El VLC es una aplicación la cual se basa firmemente en las funcionalidades provistas por la librería LibVLC. Esta última y como veremos a continuación brinda múltiples funcionalidades entre las que se encuentra la carga dinámica de módulos, el manejo genérico de buffers, las primitivas de acceso a la red, etc.

Figura 82: Principales componentes

A su vez el VLC esta compuesto por una numerosa cantidad de módulos (muchos de los cuales son cargados en forma dinámica con la LibVLC), entre los que se encuentran las implementaciones de los diferentes demuxers, tales como MPEG-TS, MPEG-PS y OGG entre otras, implementaciones de los diferentes codecs de audio y video, implementaciones de las diferentes interfaces graficas, etc.

Muchos módulos del VLC son construidos en base a librerías externas, tales como las implementaciones de varios codecs, la interfaz grafica, etc.

En resumen VLC es la combinación de las funcionalidades provistas por la LibVLC, junto con las funcionalidades específicas provistas por cada uno de los módulos integrados, muchos de los cuales se construyen en base a librerías externas.

1.3. Librería LibVLC

El código fuente de esta librería se encuentra dentro de include/ donde se encuentran los archivos .h y en src/ donde se encuentra la implementación de dichos .h (los archivos .c).

En la Figura 83 se muestra de la estructura de directorios bajo src/, para algunos de los subdirectorios dentro de este mismo se presenta una breve reseña de las funcionalidades provistas dentro de éstos subdirectorios.

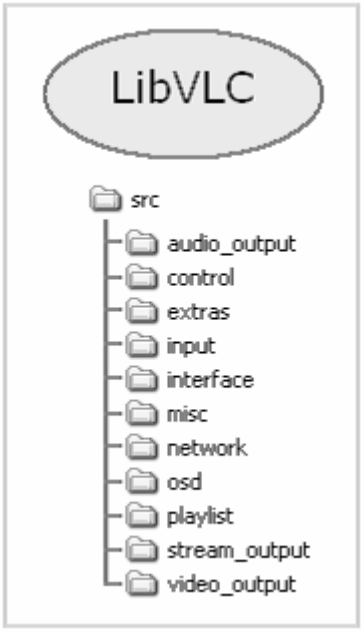


Figura 83: Estructura de directorios del LibVLC

- interface/: aquí se encuentra el código referente a la gestión de eventos generados por el usuario como puede ser el presionar un botón en la interfaz grafica, etc.

- playlist/: aquí se encuentra el código referente a la gestión la lista de reproducción, se encarga de responder ante pedidos de tipo: play, stop, etc.

- input/: aquí se encuentra el código referente a la obtención del streaming, ya sea desde un medio físico o desde la red. Es aquí donde se parsean los paquetes, se generan los ESs y se los pasa al decoder. En estas funcionalidades se profundizará en los próximos capítulos.

- video_output/: aquí se encuentra el código encargado de realizar el display del video, se tiene como entrada la salida del decoder de video, se le puede realizar alguna transformación y finalmente se despliega en pantalla.

- audio_output/: aquí se encuentra el código que procesa la salida del decorder de audio, generando el sonido de streaming.

- stream_output/: aquí es donde se genera el stream de salida en caso de que entre las opciones de volcado de salida del VLC se encuentre seleccionado el streamear por la red. En esta sección se paquetiza el stream generando así los paquetes PES.

- misc/: aquí se encuentran implementadas muchas funcionalidades de diferente tipo, como por ejemplo la carga dinámica de módulos, manejo de threads, la detección de CPU, manejo de la cola de mensajes, manejo de buffer, etc.

1.4. Módulos

Todos los módulos que están integrados dentro del VLC se encuentran bajo el directorio `modules/`.

Cada módulo contiene una sección en el código donde define su capacidad, define sus callback functions, su descripción, entre otras cosas. A continuación se presenta un ejemplo de definición de un módulo, el mismo pertenece al módulo de demux, TS.

```
vlc_module_begin();
    set_description( _("MPEG Transport Stream demuxer") );
    set_shortname ( "MPEG-TS" );
    set_category( CAT_INPUT );
    set_subcategory( SUBCAT_INPUT_DEMUX );

    add_string( "ts-extra-pmt", NULL, NULL, PMT_TEXT, PMT_LONGTEXT, VLC_TRUE );
    add_bool( "ts-es-id-pid", 1, NULL, PID_TEXT, PID_LONGTEXT, VLC_TRUE );
    add_string( "ts-out", NULL, NULL, TSOUT_TEXT, TSOUT_LONGTEXT, VLC_TRUE );
    add_integer( "ts-out-mtu", 1500, NULL, MTUOUT_TEXT,
                 MTUOUT_LONGTEXT, VLC_TRUE );
    add_string( "ts-csa-ck", NULL, NULL, CSA_TEXT, CSA_LONGTEXT, VLC_TRUE );
    add_integer( "ts-csa-pkt", 188, NULL, CPKT_TEXT, CPKT_LONGTEXT, VLC_TRUE );
    add_bool( "ts-silent", 0, NULL, SILENT_TEXT, SILENT_LONGTEXT, VLC_TRUE );

    add_file( "ts-dump-file", NULL, NULL, TSDUMP_TEXT, TSDUMP_LONGTEXT, VLC_FALSE );
    add_bool( "ts-dump-append", 0, NULL, APPEND_TEXT, APPEND_LONGTEXT, VLC_FALSE );
    add_integer( "ts-dump-size", 16384, NULL, DUMPSIZE_TEXT,
                 DUMPSIZE_LONGTEXT, VLC_TRUE );

    set_capability( "demux2", 10 );
    set_callbacks( Open, Close );
    add_shortcut( "ts" );
vlc_module_end();
```

Como se puede ver en el ejemplo, se ha resaltado 2 acciones en particular: el seteo de la capacidad del módulo, y el seteo de las callback functions.

Conceptualmente la capacidad (capability) de cada módulo se corresponde con la categoría a la que éste pertenece. Desde dentro de la LibVLC cuando es necesario cargar un módulo se llama a la función module_Need() pasándole por parámetro el

nombre de la capacidad del módulo el cual se esta necesitando (en el ejemplo seria “demux2”), dentro de esta función se busca en el banco de Módulos del VLC y se carga dinámicamente el módulo perteneciente a esa categoría (que cumpla con la capacidad buscada) que mejor se adapte a cada caso.

Cada vez que un módulo es cargado en forma dinámica, se le invoca al mismo la función definida en el primer parámetro del set_callbacks() (en el ejemplo Open()), esta función recibe como parámetro un puntero a un vlc_object_t (objeto genérico) en donde se debe inicializar el mismo. Finalmente cuando el módulo se da de baja se llama a la segunda función definida en el set_callbacks() (en el ejemplo Close()) con el fin de liberar los recursos usados por dicho módulo.

En la Figura 84 se presentan la estructura de directorios que se encuentra bajo `modules/`, comentando las de mayor relevancia para nuestro proyecto.

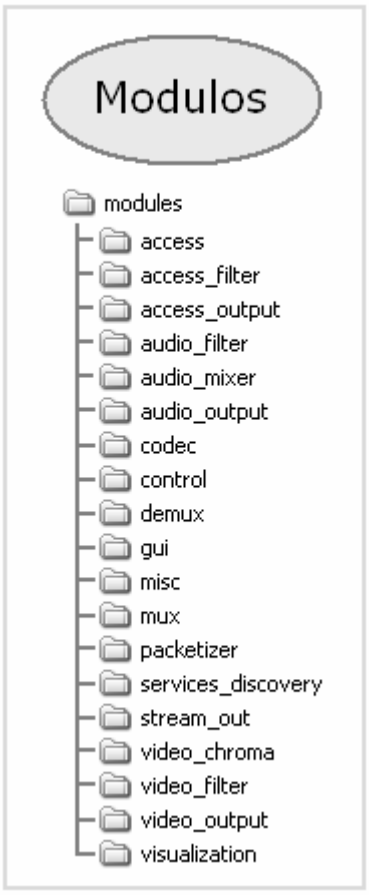


Figura 84: Estructura de directorios de los Módulos

- access/: aquí se encuentra los diferentes módulos de acceso a los datos, como por ejemplo dvdread.c (que lee desde un DVD), file.c (que lee un archivo ubicado en el disco), http.c (que lee un stream via HTTP), udp.c (que lee un stream via UDP o RTP)

- codec/: aquí se encuentran los diferentes decoders y encoders, como por ejemplo mpeg_audio.c (MPEG audio layer I/II/III decoder), libmpeg2.c (MPEG I/II video decoder), entre otros.

- demux/: aquí se encuentra los diferentes demuxers, como por ejemplo ts.c (quien maneja paquetes MPEG-TS), ps.c (quien maneja paquetes MPEG-PS), ogg.c (quien se encarga de demultiplexar paquetes OGG)

- packetizer/: algunos decoders no tienen integrado dentro de si mismo la funcionalidad de despaquetización de PES (el obtener los frames I, P, B desde dentro de un PES), por lo que aquí se encuentran módulos que realizan dicha tarea.

1.5. Secuencia de Ejecución

A continuación se presenta una secuencia de ejecución de un streaming H.264/TS-MPEG2 sobre HTTP, desde la creación de un input thread, quien será el responsable de controlar todo el proceso de streaming. Se brindará una visión a alto nivel de dicha secuencia, mostrando las invocaciones a las funciones consideradas de mayor relevancia para el desarrollo de nuestro proyecto.

1.5.1. Proceso de Inicialización

Al abrir un streaming desde la función playlist_PlayItem() se invoca la función input_CreateThread() definida en include/vlc_input.h mediante:

```
#define input_CreateThread(a,b) __input_CreateThread(VLC_OBJECT(a),b)
```

Dicha función se encuentra implementada en src/input/input.c, la misma crea un nuevo thread quien se encargara de llevar a cabo la lectura y el display del streaming a ejecutar.

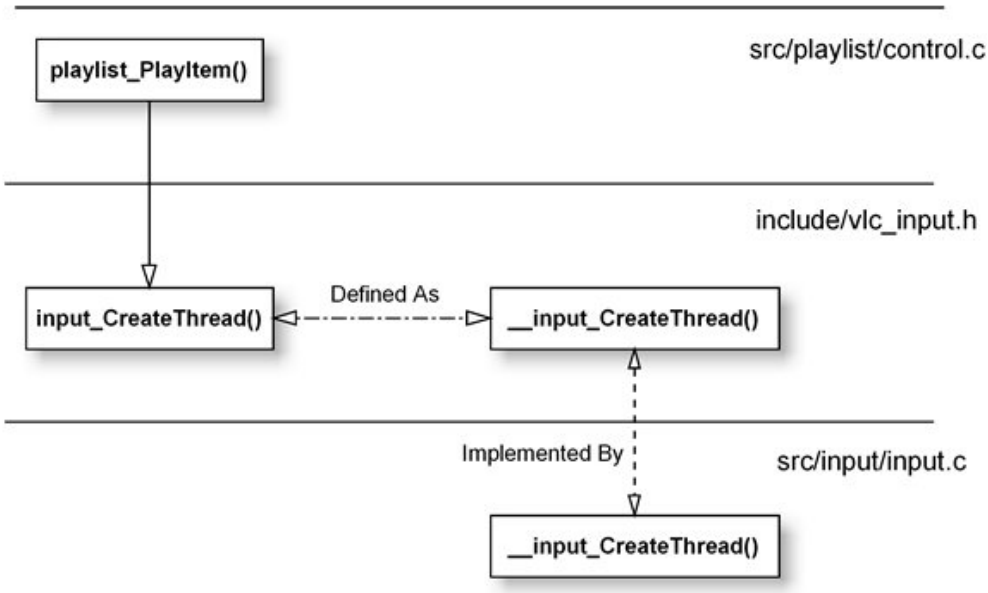


Figura 85: Iniciación de Streaming

A continuación veremos como se inicializan los diferentes módulos y estructuras.

La función __input_CreateThread() invoca a la función __input_CreateThread2() como se puede observar en la Figura 86, en donde se inicializará la estructura input_thread_t (esto en la función Create) y se creará un thread pasándole dicha estructura como parámetro. Dicho hilo ejecutará la función Run() y como se verá a continuación será el hilo encargado de leer el streaming desde la red y pasárselo al decoder para que éste genere el audio y video correspondiente.

La estructura input_thread_t será pasada en todas las funciones dentro de este módulo y allí se guardaran diferentes tipos de información, en particular se guardaran los punteros al demux y a los decoders.

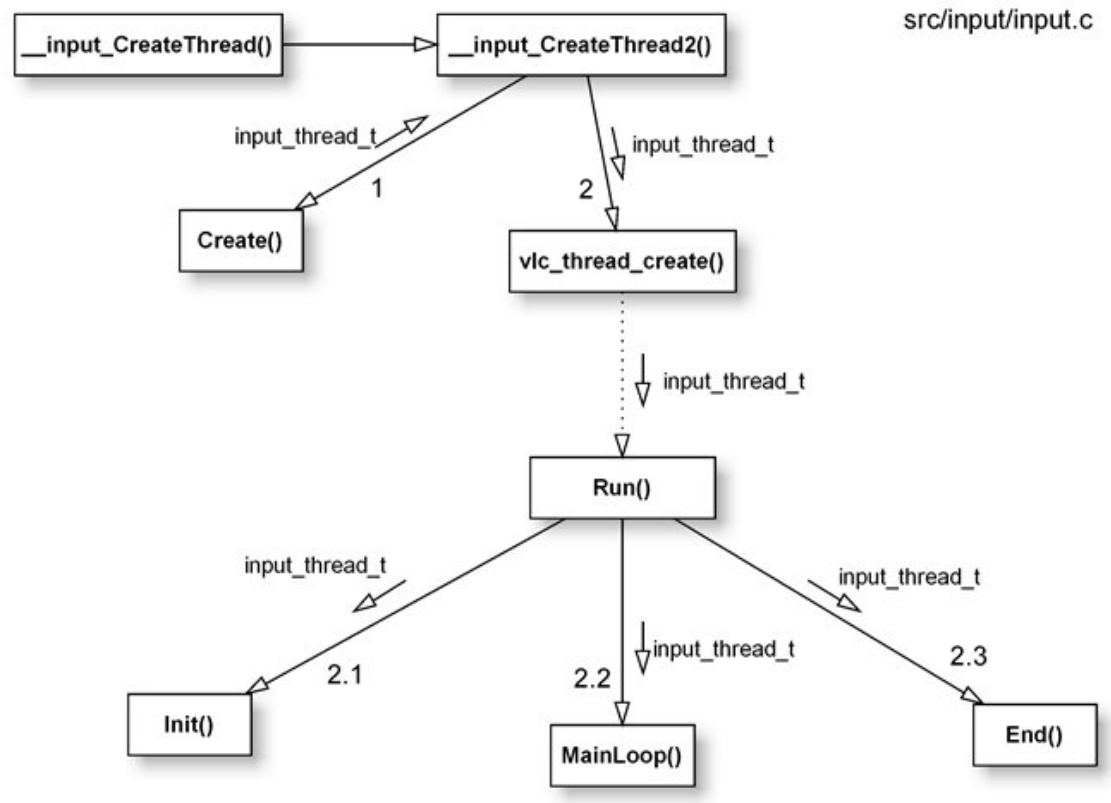


Figura 86: Creación del thread de input

Dentro de la función Run() primero se llama a la función Init() donde se setean ciertas propiedades, como veremos a continuación, luego se entra en la principal iteración (función MainLoop()) de la secuencia del display de un streaming, donde se realiza la lectura desde la red, procesamiento de dichos datos y el display propiamente dicho del audio y video.

A continuación profundizaremos en las diferentes interacciones e inicializaciones realizadas dentro de la función Init().

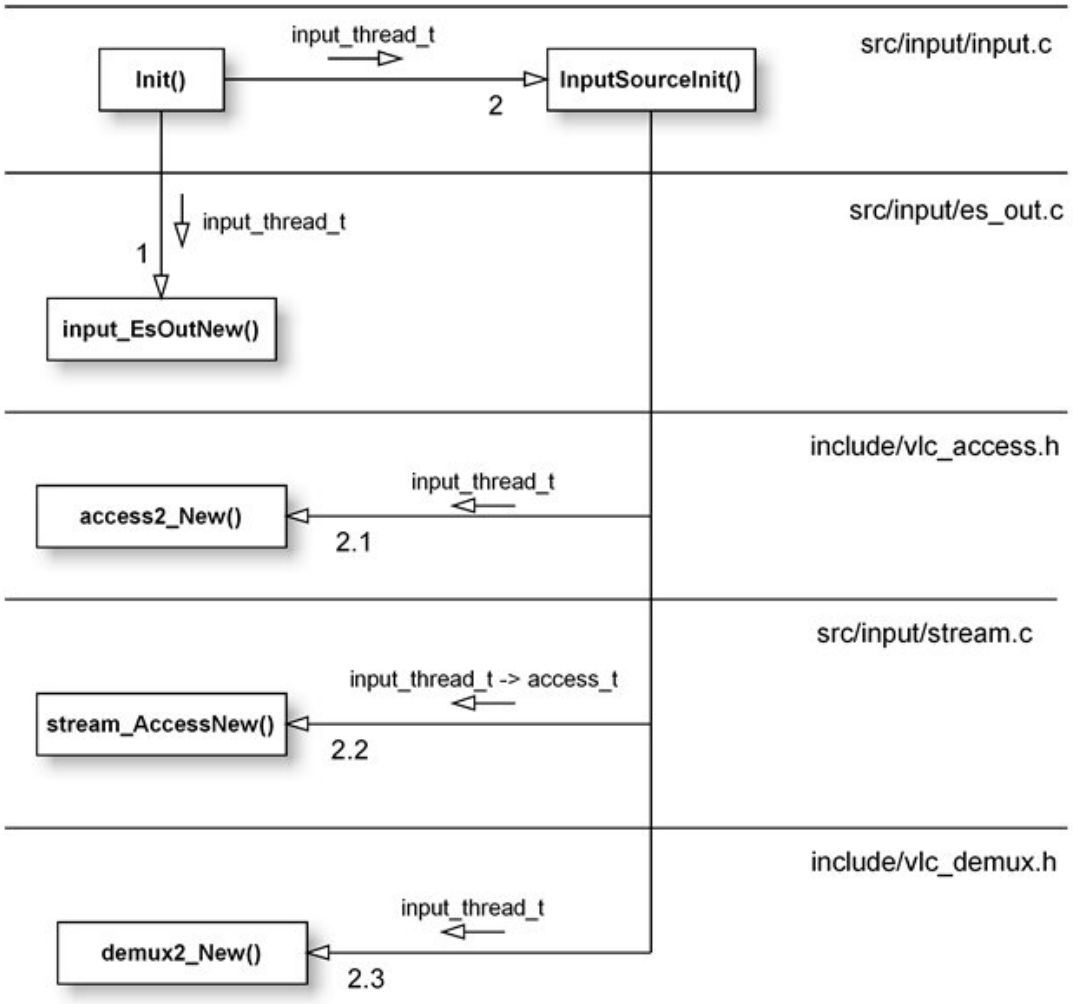


Figura 87: Funcion init – input thread

Como se puede observar en la Figura 87, dentro de la función Init(), se inicializa el módulo de ESOut (Elementary Stream Output), quien será el responsable de la entrega de los paquetes PES al decoder, el módulo de Access, módulo de lectura desde la red, el módulo de Stream, módulo de encargado de realizar el buffering del streaming y el módulo de demux, módulo encargado de demultiplexar el streaming.

1.5.2. Creación de módulo EsOut

La función input_EsOutNew() básicamente se dedica a setear varias propiedades del es_out_t, entre las que destacamos los punteros a las siguientes funciones:

```
es_out_t     *out = malloc( sizeof( es_out_t ) );
.. .. ..
out->pf_add     = EsOutAdd;
out->pf_send    = EsOutSend;
out->pf_del     = EsOutDel;
out->pf_control = EsOutControl;
.. .. ..
```

```
return out;
```

Esta función es invocada desde el InputSourceInit(), con la siguiente invocación:

```
/* Create es out */
p_input->p_es_out = input_EsOutNew( p_input );
```

1.5.3. Creación de módulo de Access

A continuación profundizaremos en la función access2_New() responsable de cargar el módulo de acceso.

Como se puede observar en la Figura 88, esta función invoca a access2_InternalNew() donde se crea la estructura access_t, seteando las propiedades por defecto, entre ellas podemos destacar los punteros a los siguientes funciones:

```
access_t *p_access = vlc_object_create( p_obj, VLC_OBJECT_ACCESS );
.. .. ..
p_access->pf_read    = NULL;  // Función de lectura
p_access->pf_block   = NULL;  // Función de lectura de bloques
p_access->pf_seek    = NULL;  // Función de seek
p_access->pf_control = NULL;  // Función de control
```

Luego dentro de esta función se carga el módulo de acceso (a través de la función module_Need()) que corresponda. Para esto se usa la siguiente capacidad: “access2”:

```
p_access->p_module = module_Need( p_access, "access2", p_access->psz_access,
                                  b_quick ? VLC_TRUE : VLC_FALSE );
```

Dado que el ejemplo que estamos manejando es el de un streaming sobre HTTP, el módulo que será cargado dinámicamente será el modules/access/http.c.

Una ves cargado (y como se comentó anteriormente) a este se le llamará a la función especificada para la inicialización del módulo, en este caso la Open().

En la función Open(), se setéan dentro de la estructura access_t anteriormente creada las propiedades especificas al módulo cargado, dentro de las que se encuentran los punteros a las funciones concretas a usar:

```
/* Set up p_access */
p_access->pf_read = Read;
p_access->pf_block = NULL;
p_access->pf_control = Control;
p_access->pf_seek = Seek;
```

Una vez que se setean estas propiedades se invoca a la función Connect() quien a través de __net_ConnectTCP() (que se encuentra en src/network/tcp.c) y finalmente con net_Socket() (que se encuentra en src/network/io.c) crean una conexión.

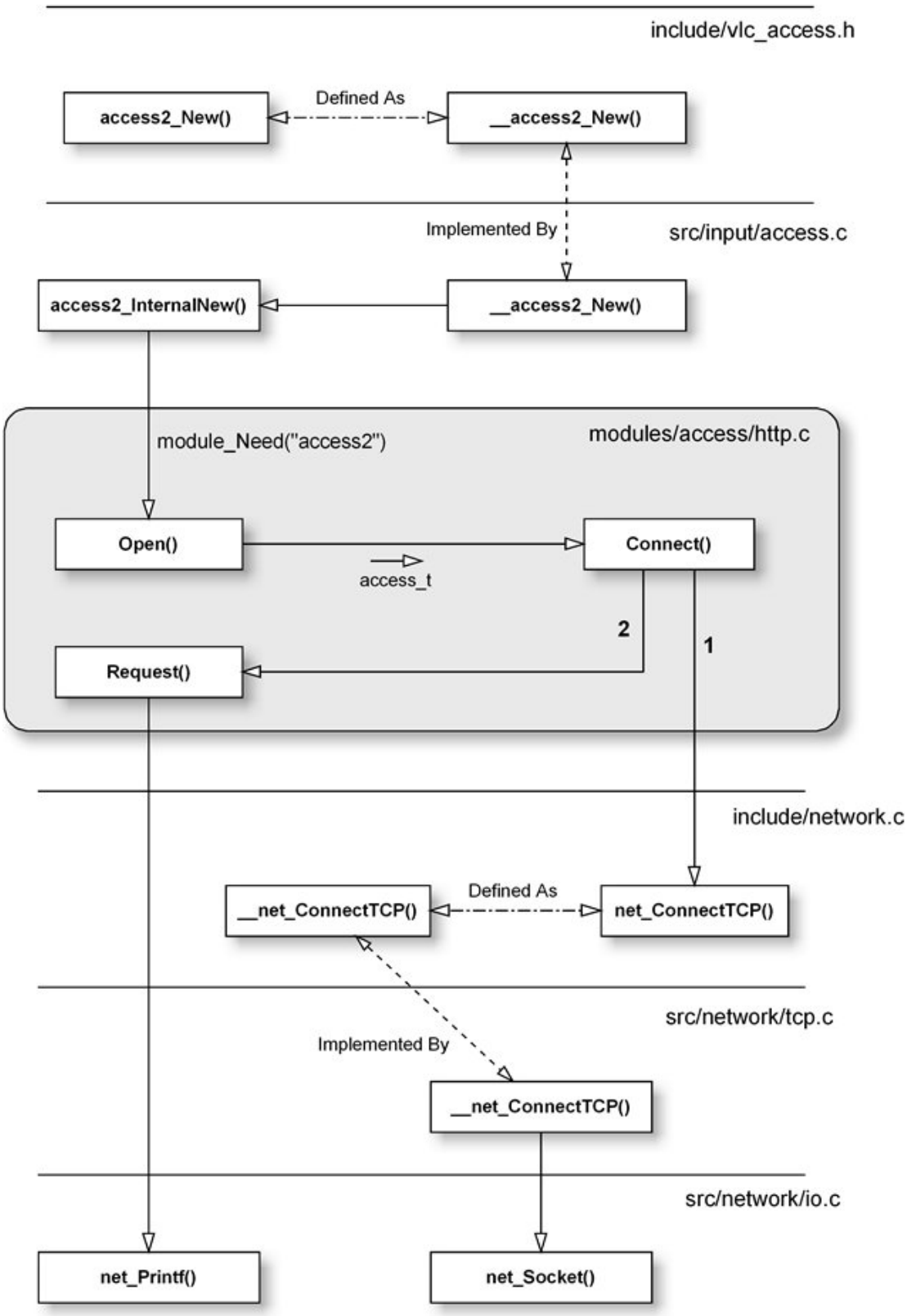


Figura 88: Conexión a través de la capa de acceso

```
// Dentro de Connect() en modules/access/http.c
access_sys_t   *p_sys = p_access->p_sys;
.. .. ..
p_sys->fd = net_ConnectTCP( p_access, srv.psz_host, srv.i_port );

-------------------------------------------------------------------------------

// Dentro de __net_ConnectTCP() en src/network/tcp.c
int fd = net_Socket( p_this, ptr->ai_family, ptr->ai_socktype,
                             ptr->ai_protocol );
.. .. ..
return fd;
-------------------------------------------------------------------------------

// Dentro de net_Socket() en src/network/io.c
fd = socket( i_family, i_socktype, i_protocol );
.. .. ..
return fd;
```

Luego de crear la conexión se llama a la función Request() donde se realizan las primeras comunicaciones con el Servidor de Streaming:

```
 // Dentro de Request() en modules/access/http.c
net_Printf( VLC_OBJECT(p_access), p_sys->fd, NULL,
                        "GET http://%s:%d%s HTTP/1.%d\r\n",
                        p_sys->url.psz_host, p_sys->url.i_port,
                        p_sys->url.psz_path, p_sys->i_version );
```

Dentro de src/network/io.c se encuentran todas las primitivas de acceso a la red, es aquí donde se manejas las funciones de manejo de sockects.

1.5.4. Creación de módulo de Stream

Con el fin de crear el módulo de Streaming se llama a la función stream_AccessNew() (observar la Figura 87) pasándole al mismo el módulo de acceso recién creado (el access_t). Como se puede observar en el siguiente código, se crea un stream_t y se lo agrega al access_t recibido.

```
stream_t *s = vlc_object_create( p_access, VLC_OBJECT_STREAM );
stream_sys_t *p_sys;
.. .. ..
s->pf_block  = NULL;
s->pf_read   = NULL;    /* Set up later */
s->pf_peek   = NULL;
s->pf_control= AStreamControl;
s->pf_destroy = AStreamDestroy;
s->p_sys = p_sys = malloc( sizeof( stream_sys_t ) );
.. .. ..
```

```
p_sys->p_access = p_access;
.. .. ..
s->pf_read = AStreamReadStream;
s->pf_peek = AStreamPeekStream;
```

A su vez también se setéan los punteros a las funciones de lectura, control, etc. del stream.

Como se puede observar en el la Figura 89 la función stream_AccessNew() invoca a la función AStreamPrebufferBlock(), para realizar el prebuffering del stream.

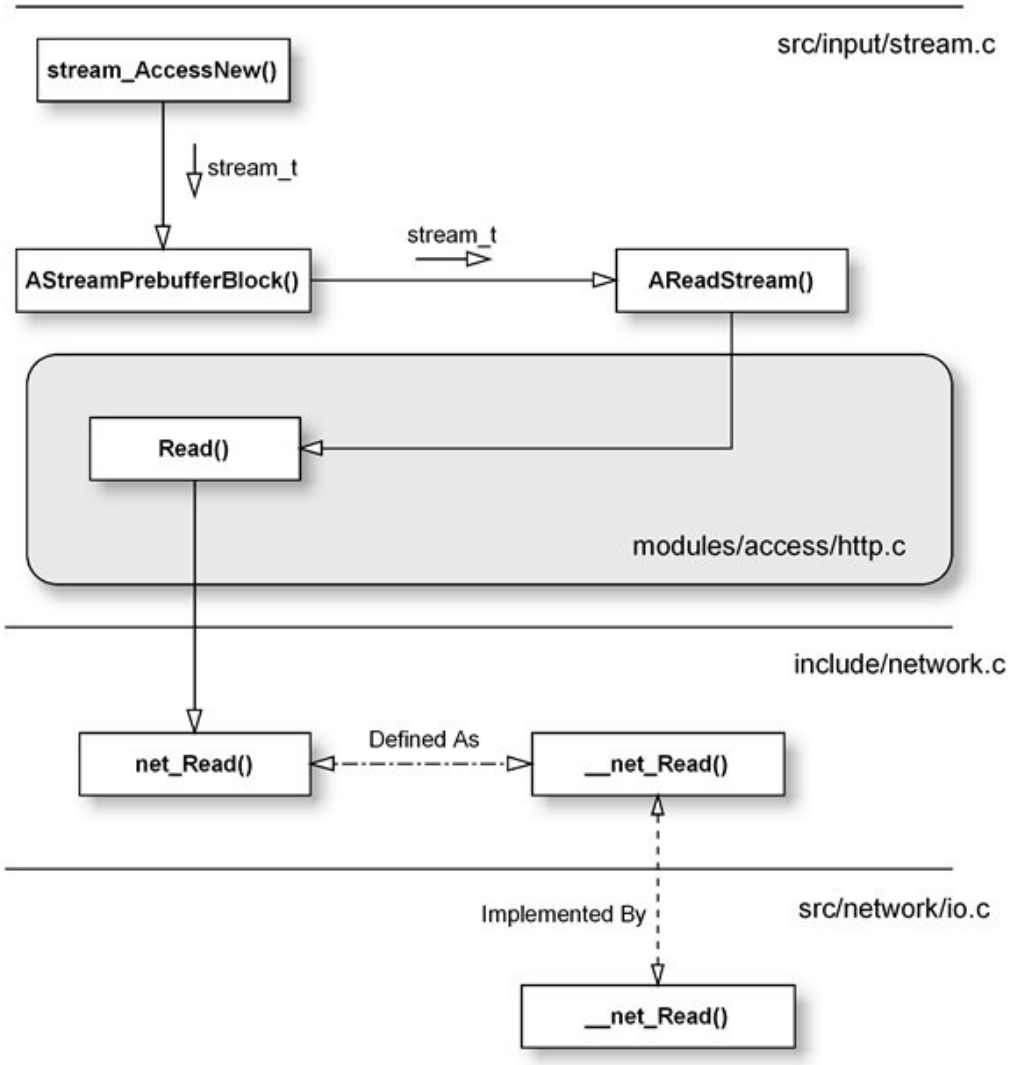


Figura 89: Creación del Modulo de Stream

La función AStreamPrebufferBlock() lleva a cabo el prebuffering llamando a la función AReadStream(), la cual usando el módulo de acceso cargado en la sección anterior, realiza la lectura de datos:

```
i_read = p_access->pf_read( p_access, p_read, i_read );
.. .. ..
return i_read;
```

1.5.5. Creación de módulo de Demux

Antes de profundizar en la implementación de esta funcionalidad veremos una breve introducción a los conceptos manejados dentro la definición del estándar MPEG-TS.

Un Transport Stream, esta compuesto por uno o más programas (un programa se podría considerar como un canal de televisión, es decir dentro de un mismo Stream se pueden transmitir varios canales en simultáneo).

El audio y el video son separados entre si, codificados cada uno de ellos, generando así los llamados Elementary Streams de audio y video respectivamente. Estos ES son paquetizados generando el llamado Packetized Elementary Stream, formado por una secuencia de paquetes PES. A su vez estos paquetes PES, son insertados en paquetes TS. Por lo general un PES es transportado en varios paquetes TS.

Dentro del header de los paquetes TS se encuentra un campo llamado Packet Identifier (PID) el cual mediante la información transmitida en las tablas pertenecientes a la Program Specific Information (PSI) identifican el contenido proveniente dentro de estos paquetes.

La PSI esta compuesta por cuatro tablas diferentes, la Program Association Table (PAT), la Program Map Table (PMT), la Conditional Access Table (CAT) y la Network Information Table (NIT). Dentro de estas tablas se encuentra toda la información necesaria para demultiplexar y presentar los diferentes programas. Las tablas PAT y PMT siempre están presentes dentro del Stream, mientras que las otras 2 pueden depender del Stream que se esta transmitiendo.

En la tabla PAT, se asocia el número de programa con el PID en el que vendrán la PMT para dicho programa, por ejemplo en la Figura 9 se muestra que en los paquetes con PID número 15 van a ser transmitidos los paquetes correspondientes a la PMT del programa 1. A su vez dentro de la tabla PMT y entre otras cosas se encuentra que PIDs y por lo tanto que ES están asociados a que programas, en nuestro ejemplo se define que el PID numero 64 es un ES de audio correspondiente al programa número 1.

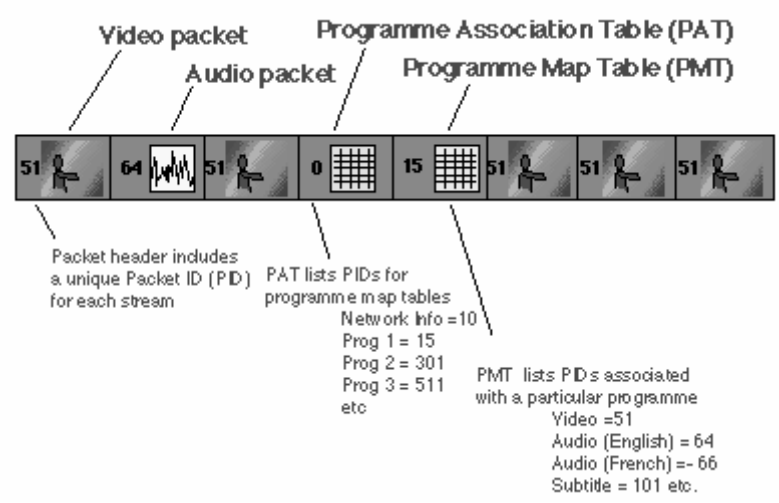


Figura 90: Ejemplo de Stream MPEG-TS

Una vez presentados estos conceptos pasaremos profundizar en la implementación de la función demux2_New() responsable de la creación del módulo de Demux (observar Figura 87).

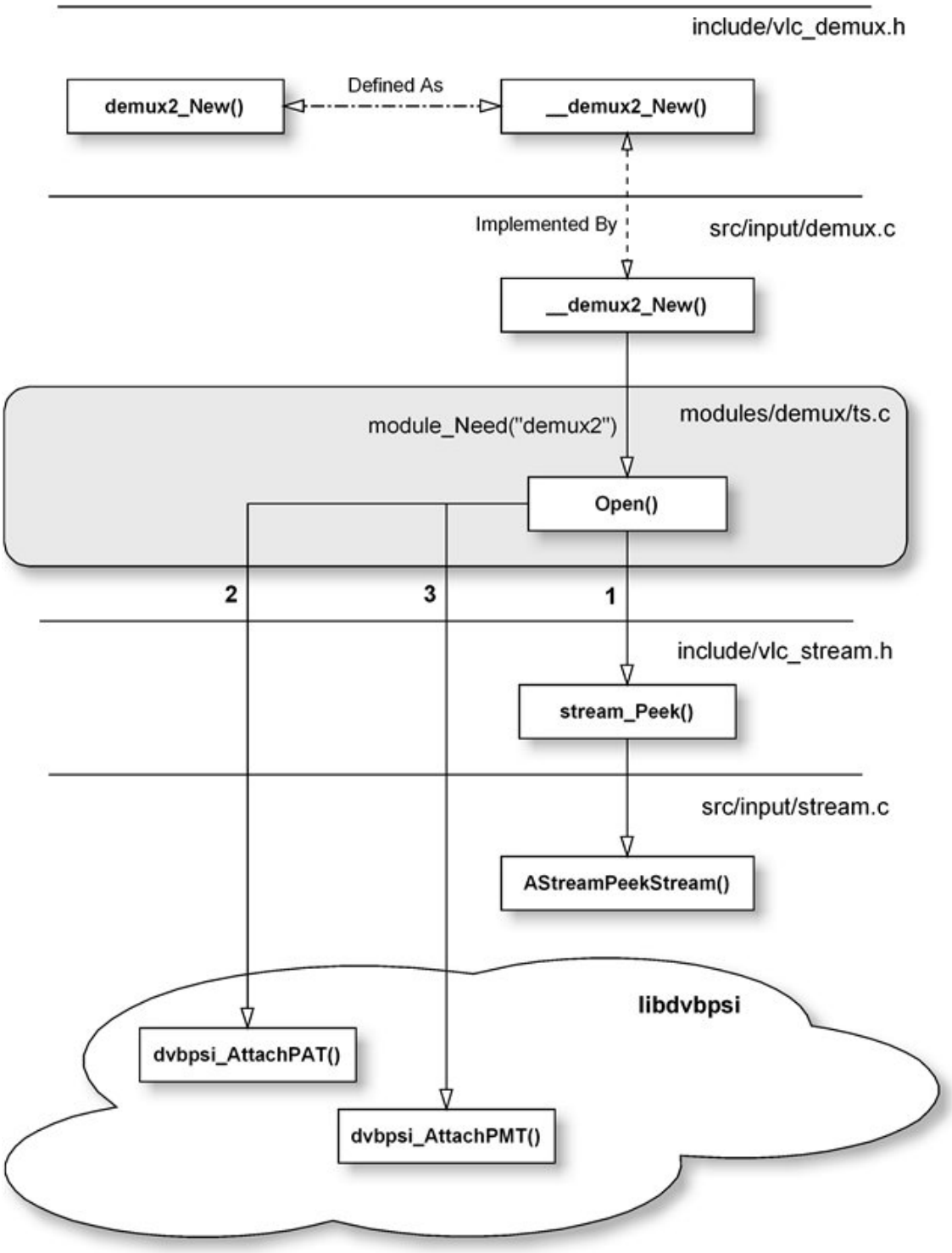


Figura 91: Creación del Modulo de Demux

Esta función crea la estructura demux_t, seteando las propiedades por defecto, entre ellas podemos destacar los punteros a los siguientes funciones:

```
demux_t *p_demux = vlc_object_create( p_obj, VLC_OBJECT_DEMUX );
.. .. ..
```

```
p_demux->pf_demux   = NULL; // Función de Demux
p_demux->pf_control = NULL; // Función de Control
```

Luego dentro de esta función se carga el módulo de demux (a través de la función module_Need()) que corresponda. Para esto se usa la siguiente capacidad: “demux2”:

```
p_demux->p_module =  module_Need( p_demux, "demux2", psz_module,
                                  !strcmp( psz_module, p_demux->psz_demux ) ?
                                  VLC_TRUE : VLC_FALSE );
```

Teniendo en cuenta que el ejemplo que estamos presentando es el de un stream MPEG TS, el módulo que será cargado dinámicamente será el modules/demux/ts.c.

Una ves cargado (y como se comentó anteriormente) a este se le llamará a la función especificada para la inicialización del módulo, en este caso la Open().

Dentro de la función Open(), se setéa la estructura demux_t anteriormente creada en la cual se asignan las propiedades especificas al módulo cargado, dentro de las que se encuentran los punteros a las funciones concretas a usar:

```
p_demux->pf_demux = Demux;
p_demux->pf_control = Control;
```

Como se puede observar en la Figura 91 la función Open() llama a la función stream_Peek() (función usada con el observar el buffer), con el fin de obtener el tamaño de las paquetes TS a manejar dentro de este Stream.

La función Open(), también invoca a funciones propias de la librería libdvbpsi, librería que se va a encargar del manejo de las diferentes tablas del PSI. A las funciones dvbpsi_AttachPAT() y dvbpsi_AttachPMT() se les pasa por parámetro el callback function a ser llamada cuando estas reciban información acerca de cada tabla.

```
/* Init PAT handler */
pat = &p_sys->pid[0];
PIDInit( pat, VLC_TRUE, NULL );
pat->psi->handle = dvbpsi_AttachPAT( (dvbpsi_pat_callback)PATCallBack,
                                     p_demux );
.. .. ..

/* Init PMT array */
p_sys->i_pmt = 0;
p_sys->pmt   = NULL;
pmt->psi->prg[0]->handle = dvbpsi_AttachPMT( 1, (dvbpsi_pmt_callback)PMTCallBack,
                                   p_demux );
```

1.5.6. Creación de módulos de Decoding

Como se vio anteriormente entre los primeros paquetes recibidos desde el Servidor se encuentran los paquetes con información correspondiente con la PSI, en particular información sobre las tablas PAT y PMT. A su vez en la sección anterior vimos que en la creación del módulo de demux, se setearon los callback functions para estas tablas. Esto funciona de la siguiente manera, cuando la librería libdvbpsi recibe información sobre estas tablas (cuya información será enviada desde el VLC a dicha librería durante el proceso de lectura), esta recoge esta información y en caso de haber completado una de estas tablas realiza un llamado a las callback functions pasadas por parámetro en la inicialización del módulo de demux.

La primera tabla que se recibirá será la PAT, dado que en esta misma se encuentra la asociación entre los programas y los PIDs de las tablas PMT de estos mismos. Los paquetes TS con los cuales se conforma la tabla PAT son identificados dado que su PID es el 0x0000.

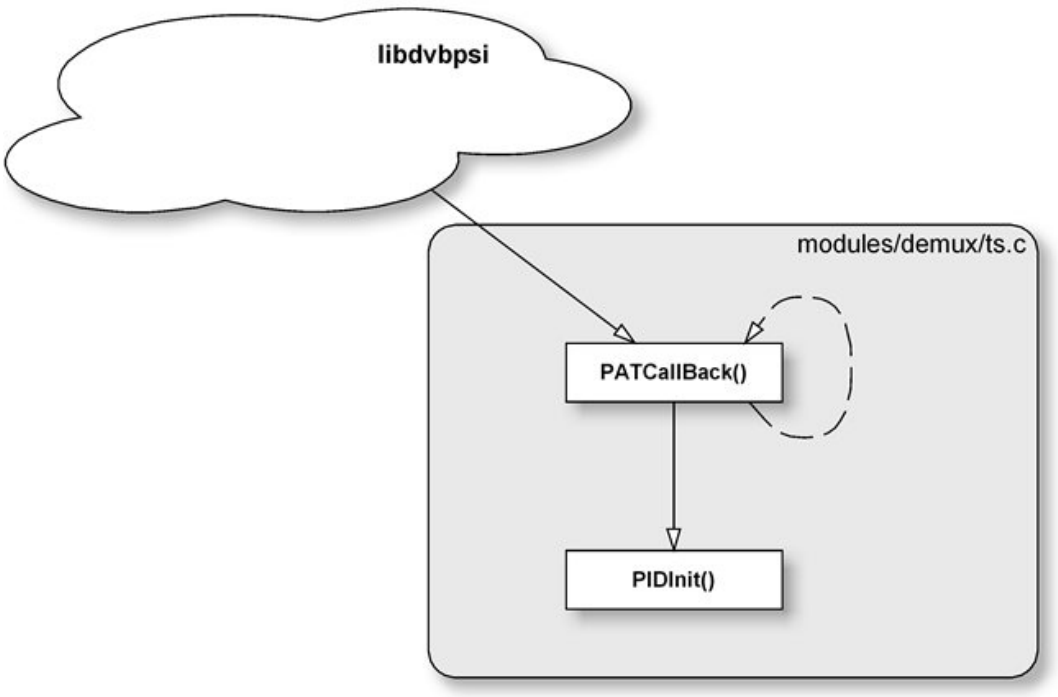


Figura 92: PATCallBack de ts

Inicialmente la función PATCallBack(), borra toda la información referente con los PIDs en caso de que esta exista, luego y como se puede observar en la Figura 92, esta realiza una iteración sobre los programas definidos en la tabla PAT recibida y inicializa los PID de estos mismos:

```
 /* now create programs */
for( p_program = p_pat->p_first_program; p_program != NULL;
     p_program = p_program->p_next )
{
```

```
        .. .. ..
        ts_pid_t *pmt = &p_sys->pid[p_program->i_pid];
        .. .. ..
        PIDInit( pmt, VLC_TRUE, pat->psi );
}
```

Vale observar que el parámetro VLC_TRUE esta indicando que este PID corresponde con un PID de PSI, el mismo será consultado durante el desarrollo del proceso de streaming.

Una vez que tiene conocimiento de esta información se podrá comenzar a recibir las tablas PMT de los programas y a medida que estas se vayan completando será invocada la función PMTCallBack (esto último, desde dentro de la librería libdvbpsi).

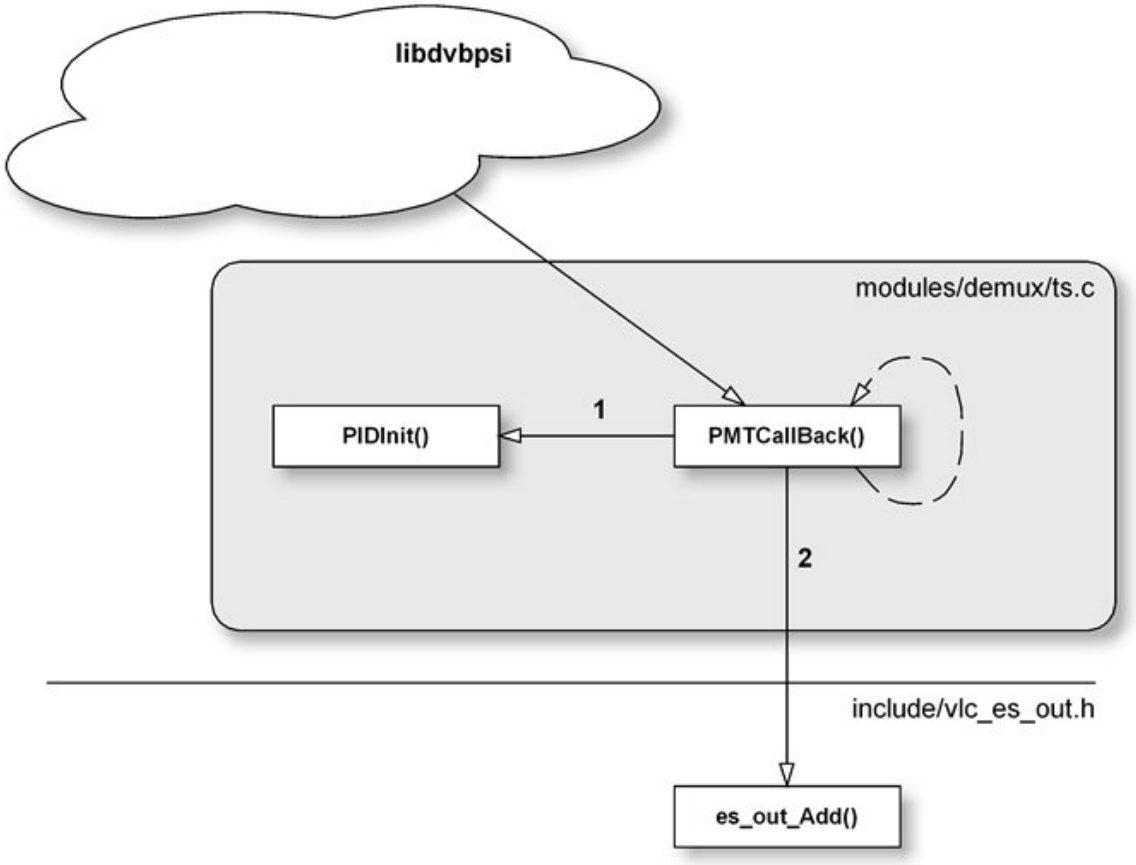


Figura 93: PMTCallBack en ts

En este caso la función PMTCallBack() y al igual que la función PATCallBAck(), esta borra toda la información sobre los ES en caso de que la misma exista.
Como se puede observar en la figura 12, se itera sobre los ES definidos dentro de la tabla PMT recibida, iniciando estos ES a través de PIDInit() y creando un es_out para cada uno de estos ES (función en la que se profundizará a continuación).

```
for( p_es = p_pmt->p_first_es; p_es != NULL; p_es = p_es->p_next )
{
        .. .. ..
        PIDInit( pid, VLC_FALSE, pmt->psi );
```

```
        .. .. ..
        pid->es->id = es_out_Add( p_demux->out, &pid->es->fmt );
        .. .. ..
}
```

Como se puede observar en el código anteriormente presentado, el PID es inicializado con VLC_FALSE, dado que en este caso el PID se corresponde con un ES y no con una tabla de PSI.

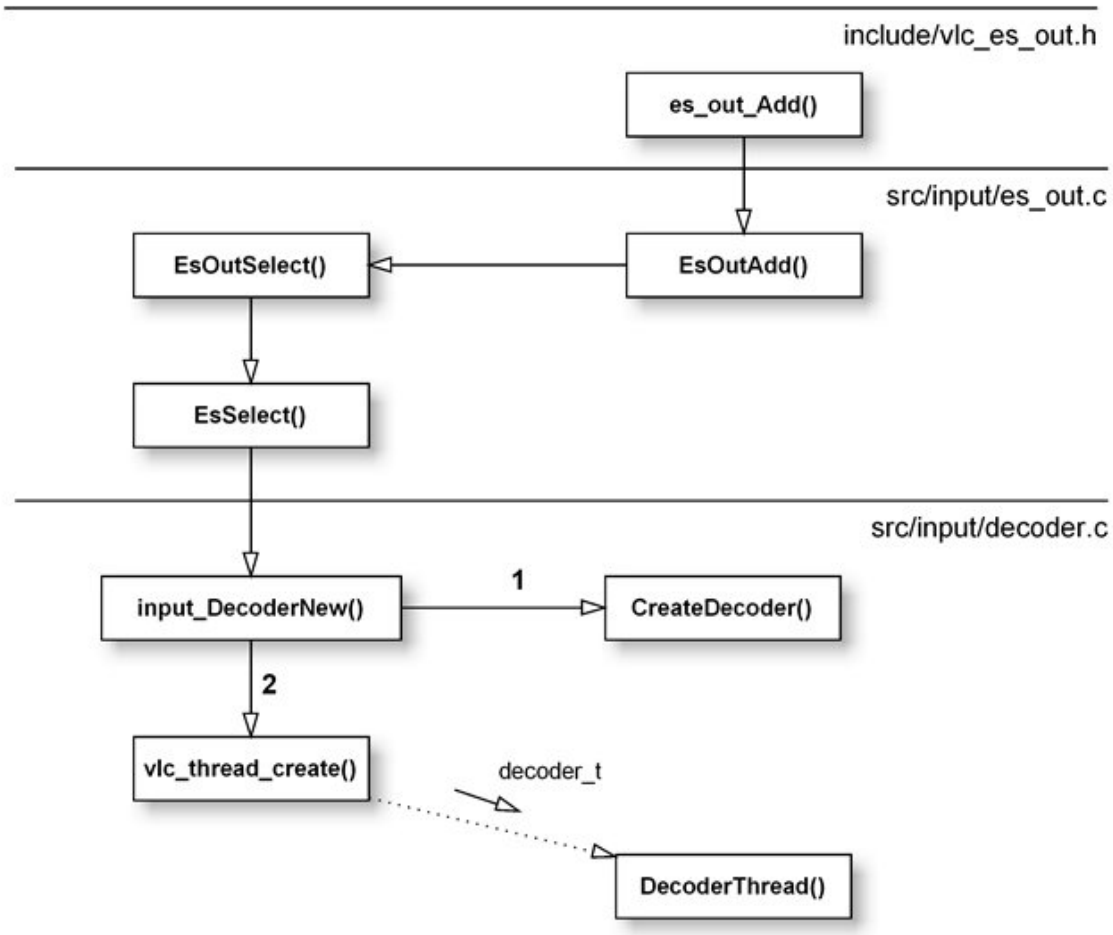


Figura 94: función es_out_Add

Básicamente la responsabilidad de la función es_out_Add() es la de cargar un módulo de decoding y dependiendo del caso, la de la creación de un hilo para que en él, se ejecute el proceso de decoding.

Como se presenta en la Figura 94 la función es_out_Add() llama a EsOutAdd(), en donde se invoca a EsOutSelect() que a su vez ésta llama a EsSelect() quien será la responsable de cargar el módulo de decoding a través de la función input_DecoderNew() y dado nuestro caso particular (él de un streaming MPEG-TS H264) la de crear un nuevo hilo para llevar a adelante dicha tarea.

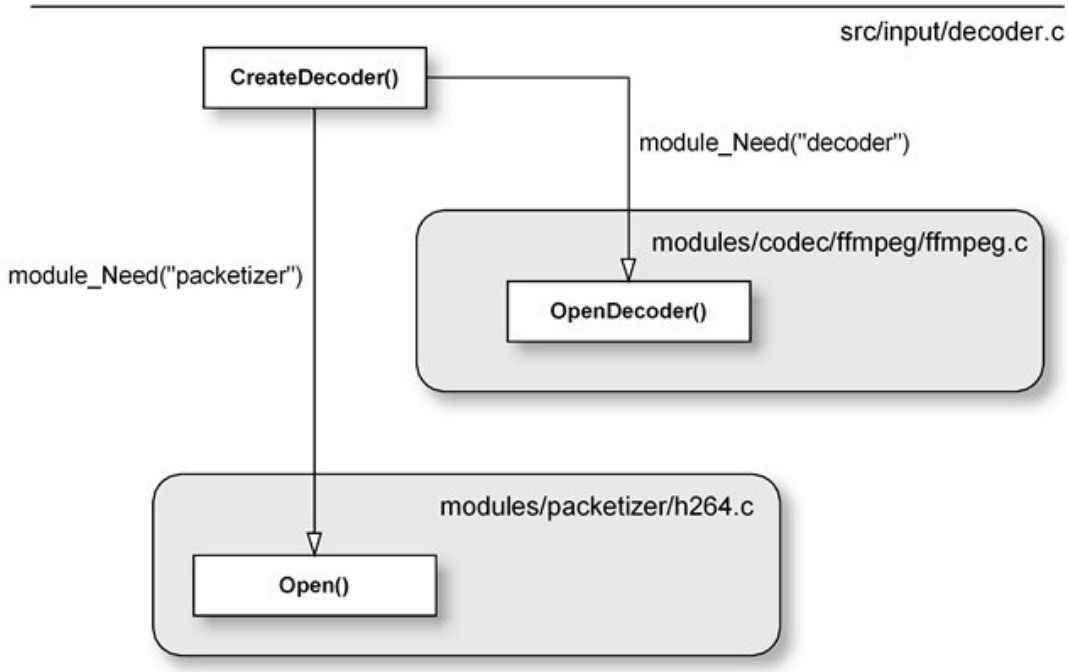


Figura 95: Creación del Modulo Decoder

Como se puede observar en la Figura 95, una vez que la función CreateDecoder() es invocada esta carga mediante module_Need() un módulo de tipo packetizer y uno de tipo decoder. En nuestro ejemplo estos corresponden con los mostrados en la figura anterior, el modules/codec/ffmpeg/ffmpeg.c y el módulos/packetizer/h264.c respectivamente.

Como de costumbre dentro de estos módulos se setéan los diferentes punteros a las funciones, las cuales serán usadas durante la ejecución del proceso de streaming.

```
// En ffmpeg.c
p_dec->pf_decode_video = E_(DecodeVideo);

// En h264.c
p_dec->pf_packetize = Packetize;
```

Cabe aclarar que este proceso de inicialización de módulos de decoding (junto con el módulo de packetizer) ocurrirá para cada uno de los ES existentes dentro de los diferentes programas incluidos en el Stream (recordar que como se vio anteriormente, dentro de un Stream MPEG-TS pueden existir múltiples programas y que para cada uno de estos programas han de existir al menos 2 ESs, uno de audio y otro de video). Por lo que para cada ES dentro de cada uno de los programas del Stream ocurrirá una diferente llamada a es_out_Add() (observar la figura 12), con todos los consiguientes pasos.

Para finalizar con la sección correspondiente con la carga de los decoders, vale aclarar que lo presentado en la Figura 95 se corresponde con un ES de video. Se opto por este mismo, dado las mediciones a realizar dentro de nuestro proyecto se basarán mayormente en atributos de esta categoría.

1.6. Proceso de ejecución del Streaming

Lo que acabamos de ver corresponde con el proceso de inicialización de los diferentes módulos, esto corresponde con todo lo realizado dentro de la función de Init() (observar la Figura 86), excepto por la inicialización de los decoder los cuales, como anteriormente se menciono, se realizan en base a la información provista por la tabla PMT.

A continuación profundizaremos en el proceso de ejecución del Streaming, el cual se encuentra implementado dentro de la función MainLoop().

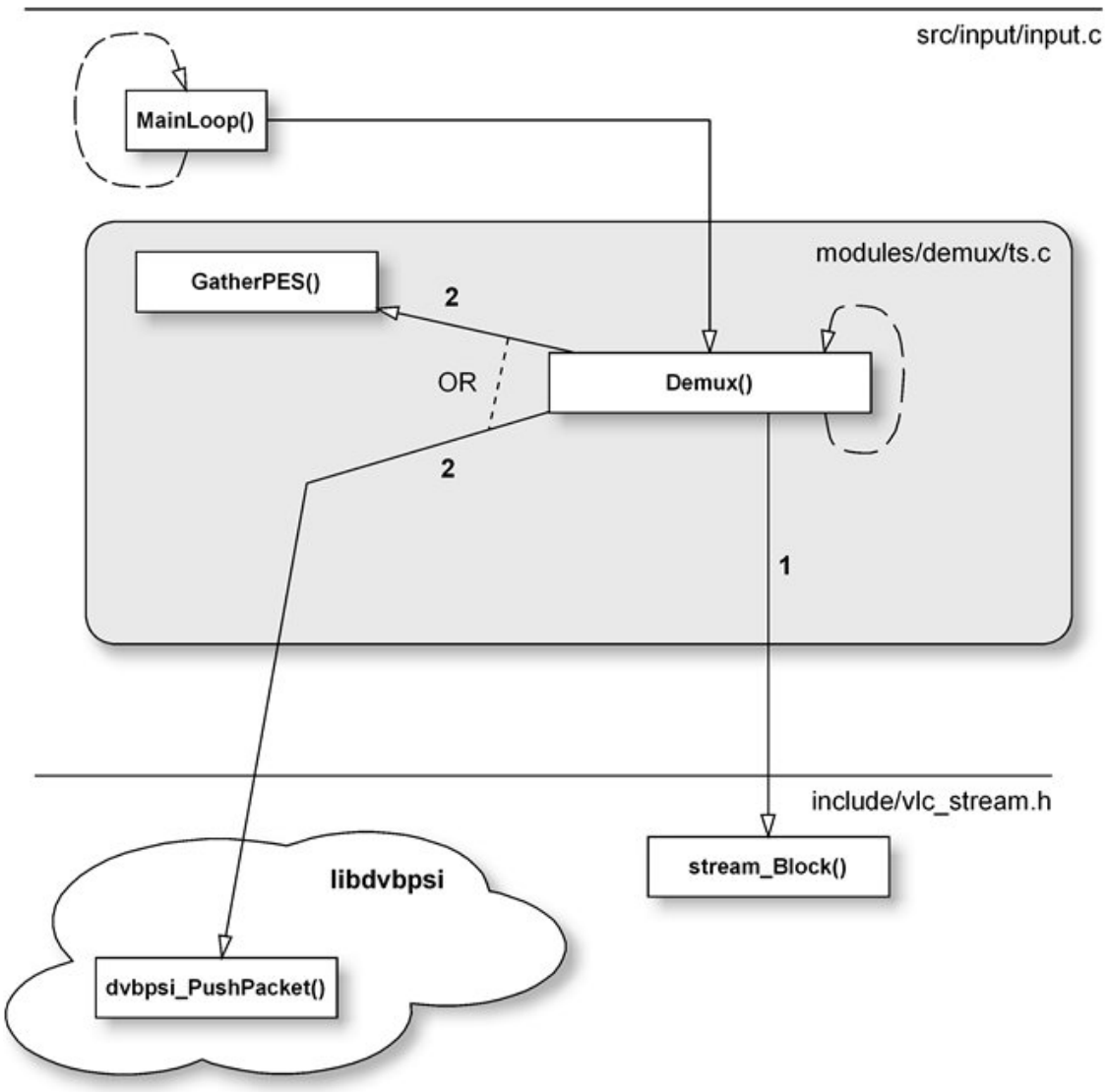


Figura 96: Ejecución de un Streaming

Dentro del MainLoop() se encuentra una iteración la cual se ejecuta mientras no haya errores, ni el thread de ejecución haya sido terminado, ni el Stream que estamos leyendo se haya acabado:

```
while( !p_input->b_die && !p_input->b_error && !p_input->input.b_eof )
{
    .. .. ..
    .. .. ..
    .. .. ..
}
```

Dentro de esta iteración y como se puede observar en la figura 15, se realiza un llamado a la función Demux() del módulo TS:

```
// pf_demux fue seteado anteriormente en la inicialización del módulo de demux
// y se corresponde con la función Demux()
//
i_ret=p_input->input.p_demux->pf_demux(p_input->input.p_demux);
```

A su vez dentro de la función Demux() existe otra iteración la cual se detiene al leer 100 paquetes TS o al completar un paquete PES:

```
/* We read at most 100 TS packet or until a frame is completed */
for( i_pkt = 0; i_pkt < p_sys->i_ts_read; i_pkt++ )
{
    .. .. ..
    if( p_pid->psi )
    {
        .. .. ..
        dvbpsi_PushPacket( p_pid->psi->handle, p_pkt->p_buffer );
    }
    else
    {
        b_frame = GatherPES( p_demux, p_pid, p_pkt );
     }
    .. .. ..
}
```

Básicamente dentro de esta iteración, lo que se realiza es la lectura de paquetes del buffer, los cuales en el caso de los paquetes con PID correspondiente a un ES son demultiplexados y pasados a los decoders correspondientes (mediante GatherPES()) y en el caso de paquetes con PID correspondiente con PSI son enviados a la librería libdvbpsi (mediante dvbpsi_PushPacket()) para que esta forme la tabla correspondiente con dicha PSI.

Para llevar a cabo la lectura de paquetes, se usa la función stream_Block(), la cual veremos a continuación.

1.6.1. Manejo de buffer de paquetes (stream_Block())

Antes de profundizar en la desarrollo de la función stream_Block(), veremos como es que se realiza el manejo de buffer. Para llevar a cabo el mismo, se almacenan los paquetes recibidos en una estructura con forma de anillo, definida de la siguiente manera:

```
typedef struct
{
    int64_t i_date;
```

```
    int64_t i_start;
    int64_t i_end;

    uint8_t *p_buffer;

} stream_track_t;


struct
{
    int i_offset;   /* Buffer offset in the current track */
    int i_tk;       /* Current track */
    stream_track_t tk[STREAM_CACHE_TRACK];

    /* Global buffer */
    uint8_t *p_buffer;

    /* */
    int i_used; /* Used since last read */
    int i_read_size;

} stream;
```

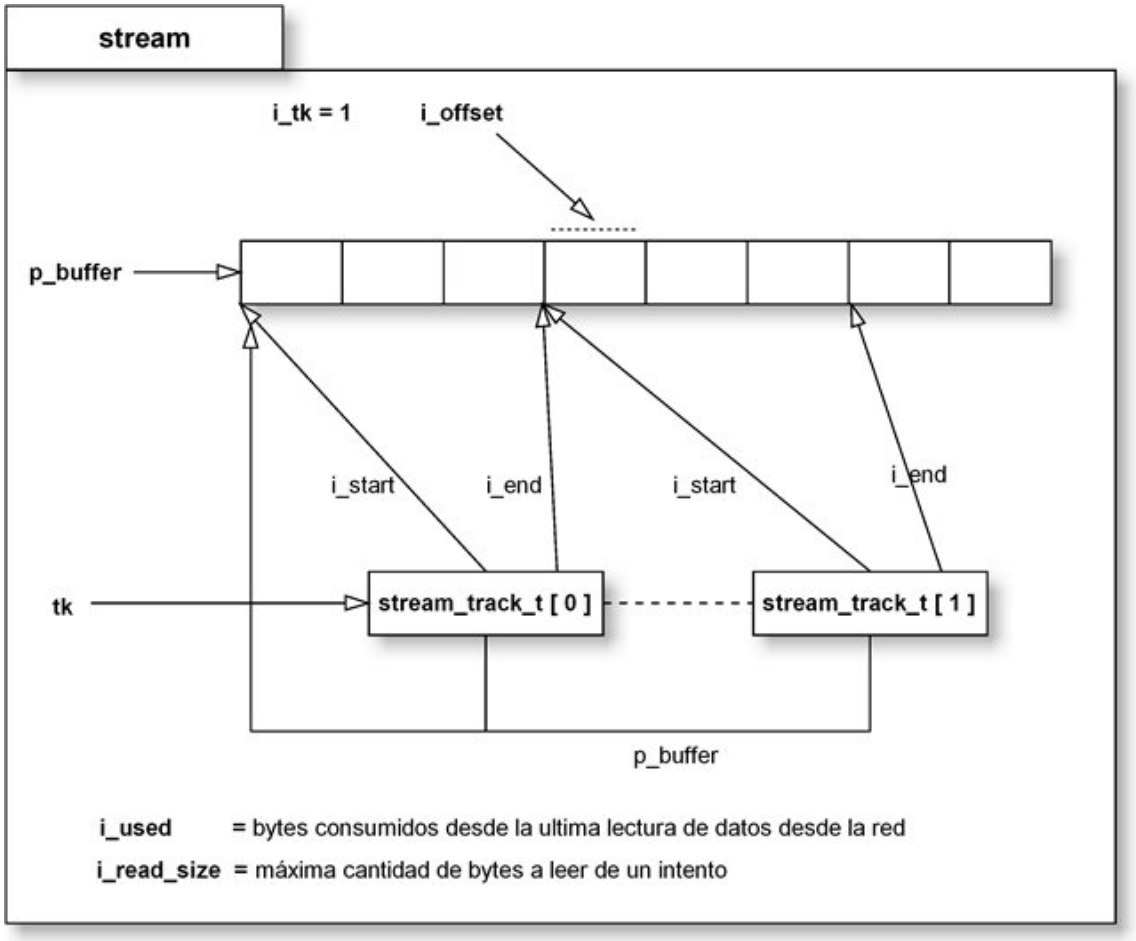


Figura 97: Manejo del buffer de paquetes

Dentro de la estructura stream, como se puede observar en la Figura 97, se tiene un puntero al buffer (p_buffer) y un array de anillos (tk), a su vez se almacena el anillo actual (i_tk), el desplazamiento dentro de dicho anillo (i_offset), la máxima

cantidad de bytes a leer de un intento (i_read_size) y los bytes consumidos desde la última lectura de datos desde la red (i_used).

Dentro de cada anillo se almacena una referencia al buffer (p_buffer), el comienzo del mismo dentro del buffer (i_start) y el final del mismo (i_end).

En nuestro caso particular, solo se maneja un único anillo (dada la implementación propia del VLC) para llevar a cabo la administración del buffer (observar la Figura 98) y se tienen las siguientes restricciones:

```
// Numero de anillos
#   define STREAM_CACHE_TRACK 1

// Máximo tamaño del buffer: 128 KB
#   define STREAM_CACHE_SIZE  (STREAM_CACHE_TRACK*1024*128)

// Máximo tamaño de anillo: 128 KB
#define STREAM_CACHE_TRACK_SIZE (STREAM_CACHE_SIZE/STREAM_CACHE_TRACK)

// Máxima cantidad de bytes a leer de un intento
#define STREAM_READ_ATONCE 32767
```

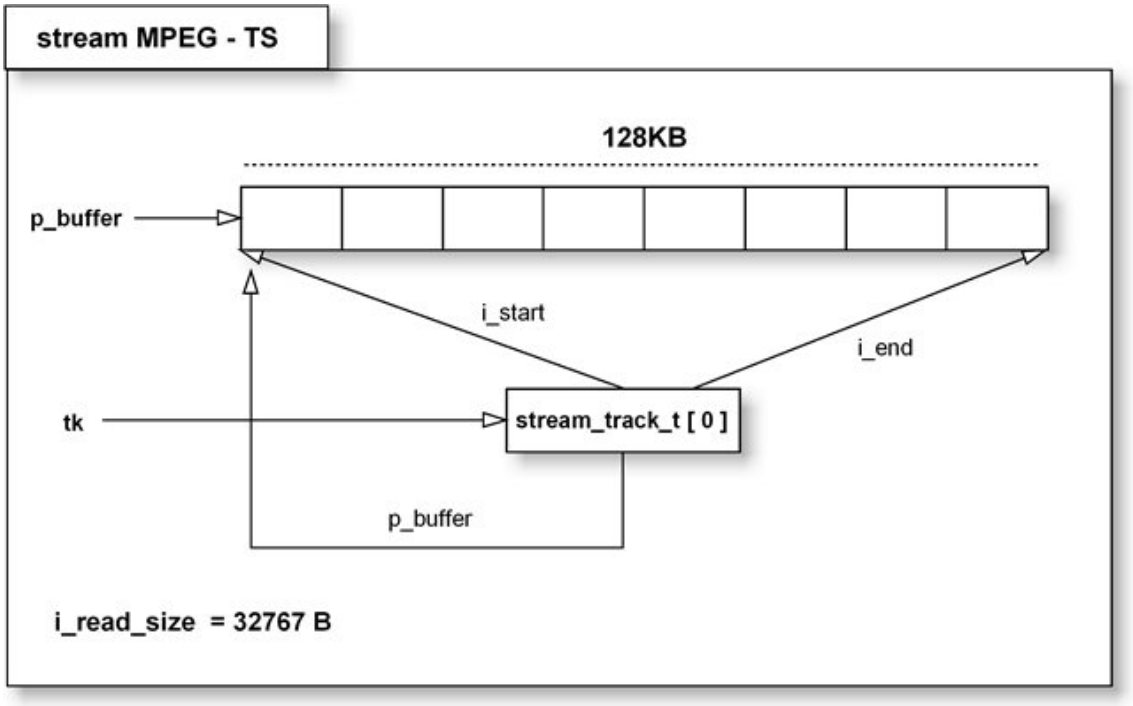


Figura 98: Buffer de stream MPEG – TS

Ahora que se tiene conocimiento acerca de las estructuras usadas para llevar a cabo el manejo de buffer, se pasará a profundizar en la función de lectura de datos del buffer, el stream_Block() (observar la Figura 97).

Como se puede observar en la Figura 99 la función stream_Block() llama a la función stream_Read() con el fin de obtener los paquetes TS, a su vez esta última función llama a AStreamReadStream(), que es quien se encarga de manejar la estructura anteriormente presentada.

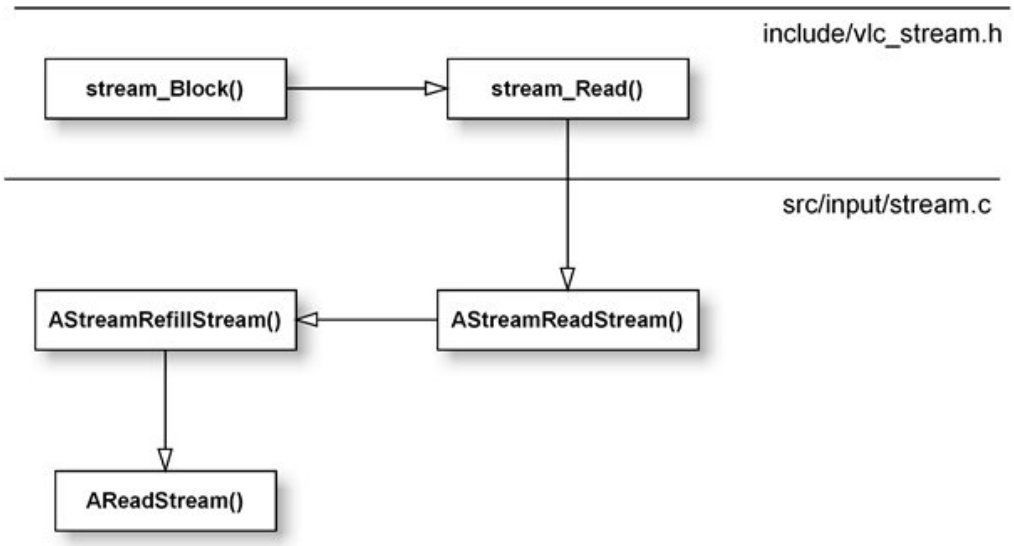


Figura 99: Obtención de paquetes TS

La función AStreamReadStream() recibe por parámetro la cantidad de bytes a leer y esta leerá del buffer dichos bytes, recargando el buffer (a través de AStreamRefillStream()) cuando se consumió todo el anillo (`tk->i_start + p_sys->stream.i_offset >= tk->i_end`) o cuando se esta consumiendo una gran cantidad de bytes de un solo intento (`p_sys->stream.i_used >= p_sys->stream.i_read_size`)

```
if( tk->i_start + p_sys->stream.i_offset >= tk->i_end ||
    p_sys->stream.i_used >= p_sys->stream.i_read_size )
{
    if( AStreamRefillStream( s ) )
    {
        /* EOF */
        if( tk->i_start >= tk->i_end ) break;
    }
}
```

Un detalle importante a tener en cuenta es que el buffer se carga en forma circular, es decir que al llegar al final del mismo se sigue por el comienzo de éste, pero tanto el tk->i_start, como el tk->i_end y como el i_offset son incrementados en forma lineal, porque lo que para manejar estos datos en forma concreta es necesario realizarles el módulo entre STREAM_CACHE_TRACK_SIZE.

```
int i_off = (tk->i_start + p_sys->stream.i_offset) %
                  STREAM_CACHE_TRACK_SIZE;
```

Para finalizar con el manejo del buffer, la función AStreamRefillStream() recarga el buffer llamando a la función AReadStream(), la cual fue comentada en la Figura 90.

1.6.2. Armado de paquetes PES

Para decodificar lo guardado en el buffer se llama a la función GatherPES() (ver Figura 15). Antes de continuar con la implementación de dicha función veremos una breve introducción a otro de los aspectos especificados dentro del estándar MPEG y el cual tiene un papel central dentro de la transmisión de streaming, la misma es la sincronización entre el codificador y el decodificador.

Para esto el estándar MPEG define dentro el Anexo D el llamado Timing Model el cual presentaremos muy brevemente a continuación. Para esta descripción no solo nos basamos en el estándar ISO/IEC 13818-1 [Ref14], sino también en [Ref13] de donde fue tomada la Figura 100.

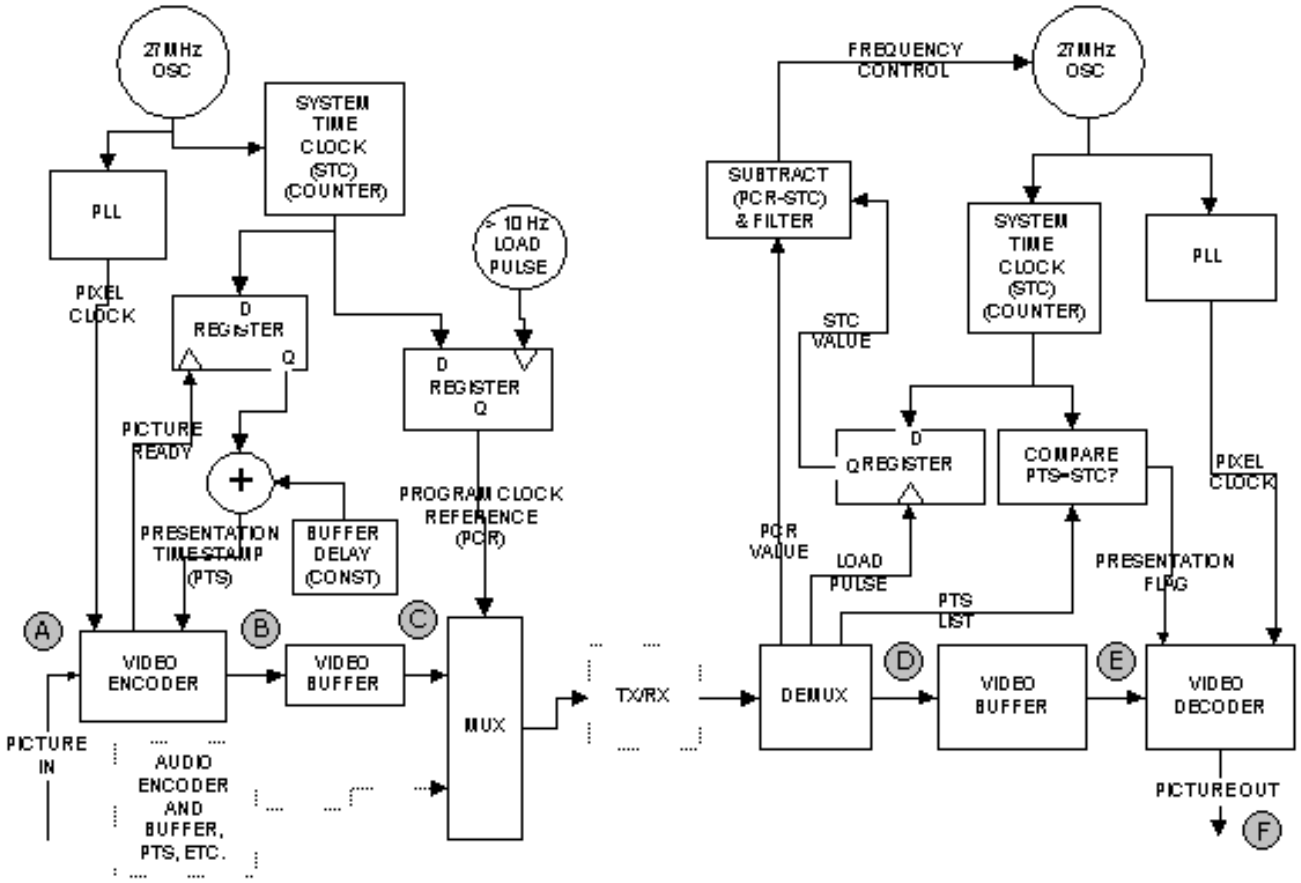


Figura 100: Sincronización de la codificación

El flujo de datos normal dentro de la ejecución de un streaming es presentado en la Figura 100, el cual se corresponde con la entrada de datos al encoder (representada por A), la entrada de los datos codificados a el buffer del encoder (representado por B), de allí los datos codificados son multiplexados (representado por C), de allí enviados mediante un canal de transporte al demultiplexador, quien recibe estos datos, los demultiplexa y los almacena en su propio buffer (representado por D), luego estos son enviados al decoder (representado por E) quien decodifica estos datos y los despliega (representado por F).

El modelo establece que la llegada de datos al encoder (A) se realiza bajo una tasa constante, mientras que los datos son almacenados en el buffer (B) en base a una tasa variable, y a pesar de esto se establece que los datos son enviados al demux mediante un tasa constante (de (C) a (D)), luego se establece que los datos son entregados al decoder bajo una tasa variable (E) y este ultimo presenta los datos decodificados mediante una tasa constante o especificada (F).

A su vez se asume que tanto la transmisión de los datos a través de la red, como el proceso de demux, como el proceso de decoding, no generan ningún atraso a los bloques transmitidos, demultiplexados y decodificados respectivamente, por lo que los únicos atrasos que puede sufrir un streaming son generados por los buffers tanto del emisor como del receptor. Puntualmente se asume que tanto el proceso de encoding como el de decoding son realizados en tiempo 0.

En base a este modelo, el encoder se sincroniza con el decoder mediante el pasaje de timestamps, los cuales son generados de la siguiente manera. El encoder posee un reloj principal (cuya frecuencia es de 27 MHz) junto con un contador el cual se incrementa en dicha frecuencia llamado STC (System Time Clock). Mediante este contador se rigen todos los ESs dentro de un programa del Stream. Observar que en un streaming con n programas pueden existir n STCs.

En base a este STC y en intervalos no mayores a 700 mS, ciertos paquetes PES son marcados con el valor del STC en el instante exacto en el cual estos han sido codificados (punto B en el diagrama), sumándole una constante la cual representa el delay agregado por los buffers del encoder y del decoder. A estos timestamps se le llaman PTS (Presentation Time Stamp) e indican el momento exacto en que dicho PES debe ser retirado del buffer de decoding, decodificado y presentado.

Bajo ciertas condiciones también puede existir otro timestamp llamado DTS (Decoding Time Stamp), el cual es marcado en el momento exactamente antes de que un bloque sea decodificado, teniendo en cuenta que el proceso de encoding es llevado a cabo en tiempo 0, el PTS y el DTS van a ser iguales, salvo en el caso en que se produce reordenamiento de frames y que justamente es este el caso cuando dichos timestamps son usados.

Todo lo visto se basa en el hecho en que dentro del receptor también existe un STC a la misma frecuencia que el del emisor (27 MHz), mediante el cual se interpretan los timestamps anteriormente ingresados (el PTS y el DTS).

El problema que se describe a continuación surge exactamente por esto mismo, es extremadamente difícil que los 2 relojes se ejecuten bajo una frecuencia exacta de 27 MHz, por lo que va a ser necesario sincronizar estos relojes periódicamente.

Con el fin de solucionar este problema aparece el tercer timestamp manejado por el estándar, el llamado PCR (Program Clock Reference). Los cuales son marcados dentro de los paquetes TS, en periodos no mayores a 100 mS insertados dentro del proceso de múltiplexación (punto C de la Figura 101) . Mediante estos se sincroniza el STC del decoder, pausando el mismo por unos momentos en el caso en que el mismo tenga mayor frecuencia que el STS del encoder o seteándolo al nuevo valor del PRC en el caso en que la frecuencia del mismo sea menor que la del encoder.

Terminada con esta introducción profundizaremos en el desarrollo de la función GatherPES(), que como se puede observar en la Figura 97, cuando se lee un paquete cuyo PID se corresponde con un ES válido, esta misma es invocada.

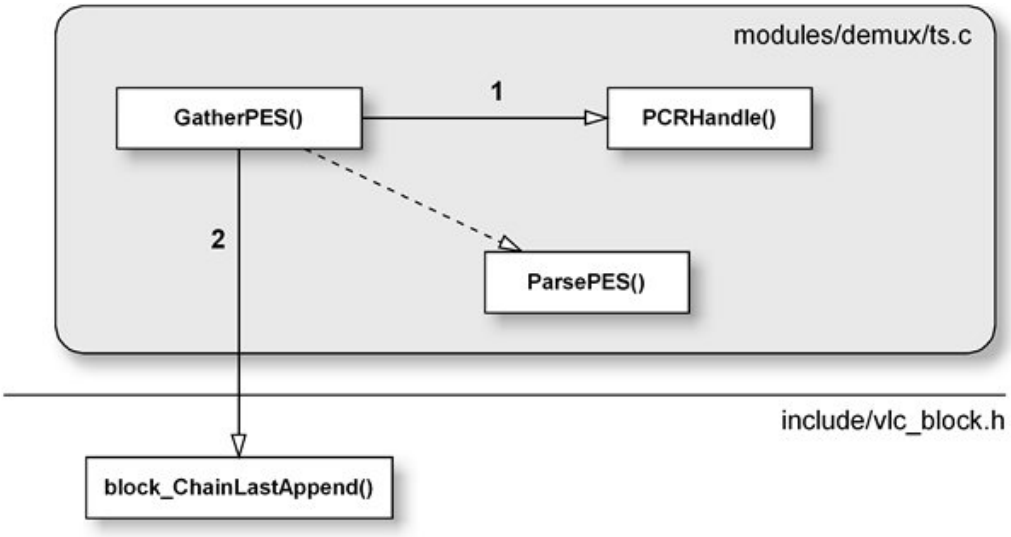


Figura 101: Decodificado del buffer

Dentro de la función GatherPES(), inicialmente se realiza un parseo del paquete leído extrayendo información tal como el Start Indicador, Adaptation Field Control, Continuity Counter, entre otros. Por más información acerca de los campos incluidos en el header de un paquete TS hacer referencia a la sección Detalle del Transport Stream del Marco Teórico.

```
const uint8_t    *p = p_bk->p_buffer;
const vlc_bool_t b_unit_start = p[1]&0x40;
const vlc_bool_t b_adaptation = p[3]&0x20;
const vlc_bool_t b_payload    = p[3]&0x10;
const int        i_cc         = p[3]&0x0f;   /* continuity counter */
```

También dentro de esta función es donde se registran las discontinuidades en la transmisión de paquetes TS.

```
/* Test continuity counter */
/* continuous when (one of this):
    * diff == 1
    * diff == 0 and payload == 0
    * diff == 0 and duplicate packet (playload != 0) <- should we
    *   test the content ?
 */
i_diff = ( i_cc - pid->i_cc )&0x0f;
if( b_payload && i_diff == 1 )
{
    pid->i_cc++;
}
```

```
else
{
    if( pid->i_cc == 0xff )
    {
        pid->i_cc = i_cc;
    }
    else if( i_diff != 0 && !b_discontinuity )
    {
        pid->i_cc = i_cc;
        if( pid->es->p_pes && pid->es->fmt.i_cat != VIDEO_ES )
        {
            /* Small video artifacts are usually better then
             * dropping full frames */
            pid->es->p_pes->i_flags |= BLOCK_FLAG_CORRUPTED;
        }
    }
}
```

Una vez obtenida esta información y como se presenta en la Figura 101, se invocará a la función PCRHandle() quien como veremos a continuación será la responsable de establecer la sincronía entre los relojes del emisor y del receptor.

Luego se almacenará el paquete recibido dentro del buffer propio del ES en cuestión (mediante la función block_ChainLastAppend()) y en el caso en que se haya completado un paquete PES (recordar que un paquete PES está generalmente compuesto por varios paquetes TS), se invocará a la función ParsePES(), la cual también profundizaremos a continuación.

La función PCRHandle() va a analizar el paquete TS recibido, y en el caso en que éste contenga un PCR, éste buscará el programa al cual corresponda a dicho ES y llamará a la función es_out_Control() pasando el valor del PCR recibido, el programa y el parámetro ES_OUT_SET_GROUP_PCR.

```
static void PCRHandle( demux_t *p_demux, ts_pid_t *pid, block_t *p_bk )
{
    demux_sys_t   *p_sys = p_demux->p_sys;

    // Este es el paquete TS recibido
    const uint8_t *p = p_bk->p_buffer;

       if( ( p[3]&0x20 ) && /* adaptation */
        ( p[5]&0x10 ) &&
        ( p[4] >= 7 ) )
    {
        int i;
        mtime_t i_pcr;  /* 33 bits */

        i_pcr = ( (mtime_t)p[6] << 25 ) |
                ( (mtime_t)p[7] << 17 ) |
                ( (mtime_t)p[8] << 9 ) |
                ( (mtime_t)p[9] << 1 ) |
                ( (mtime_t)p[10] >> 7 );

        /* Search program and set the PCR */
        for( i = 0; i < p_sys->i_pmt; i++ )
        {
            int i_prg;
            for( i_prg = 0; i_prg < p_sys->pmt[i]->psi->i_prg; i_prg++ )
            {
                if( pid->i_pid == p_sys->pmt[i]->psi->prg[i_prg]->i_pid_pcr )
```

```
                {
                    es_out_Control( p_demux->out, ES_OUT_SET_GROUP_PCR,
                                    (int)p_sys->pmt[i]->psi->prg[i_prg]->i_number,
                                    (int64_t)(i_pcr * 100 / 9) );
                }
            }
        }
    }
}
```

Como se puede observar en la Figura 102 la función es_out_Control() invoca a es_out_vaControl(), la cual invoca a EsOutControl, quien finalmente invocará a input_ClockSetPCR(), quien será el real responsable de establecer el sincronismo entre el encoder y el decoder.

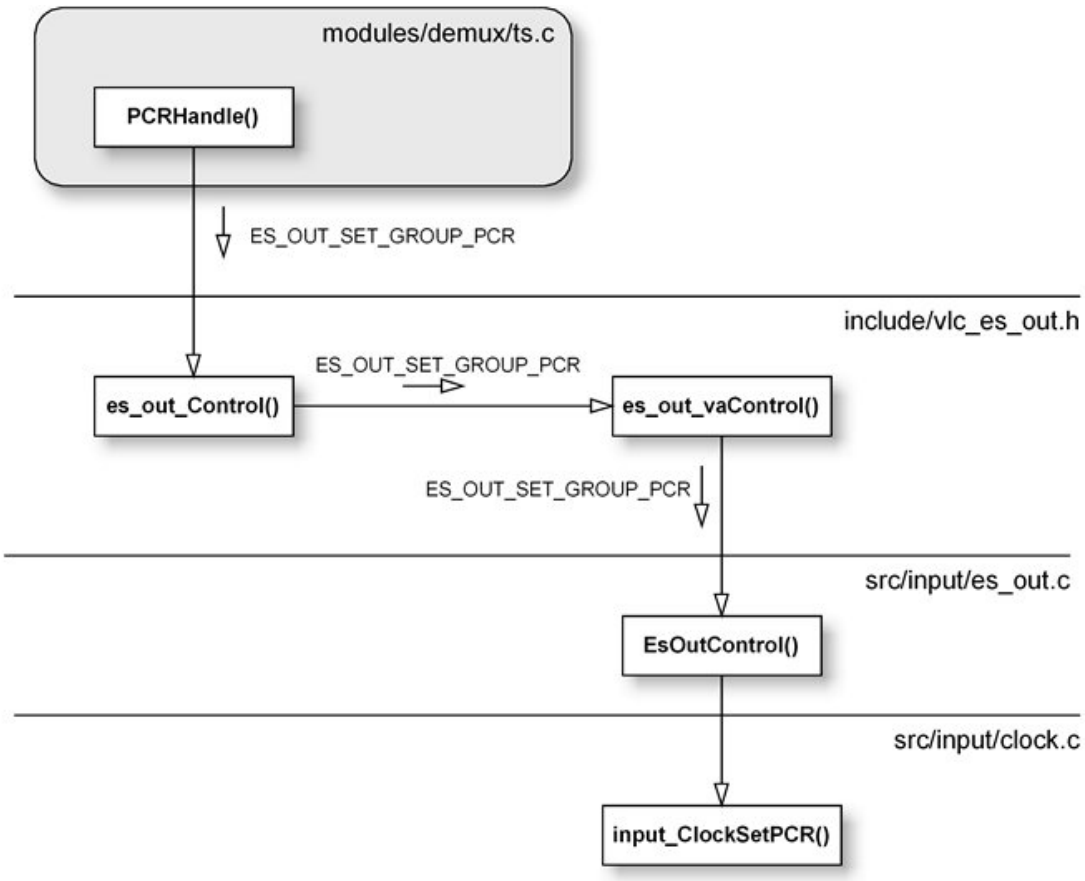


Figura 102: Sincronismo Encoder – Decoder

Lo que se puede ver a continuación es en la forma en que el input_ClockSetPCR(), realiza la sincronización del STC:

```
mtime_t i_wakeup = ClockToSysdate( p_input, cl, i_clock );
while( (i_wakeup - mdate()) / CLOCK_FREQ > 1 )
{
    msleep( CLOCK_FREQ );
    if( p_input->b_die ) i_wakeup = mdate();
}
mwait( i_wakeup );
```

A continuación profundizaremos en la implementación de la función ParsePES(), función la cual es llamada desde GatherPES() (observar la Figura 103) al completar un paquete PES. Esta parsea el paquetes PES recibido, extrayendo el payload del mismo el cual será entregado al decoder.

Mediante la función block_ChainExtract() se extrae los primeros 30 bytes del paquete PES, de donde se analizara el header de dicho paquete calculando cual es el real tamaño del header del PES (observar la Figura 103).

```
// p_pes: cadena de paquetes TS que constituyen el PES
block_t *p_pes = pid->es->p_pes;
i_max = block_ChainExtract( p_pes, header, 30 );
```

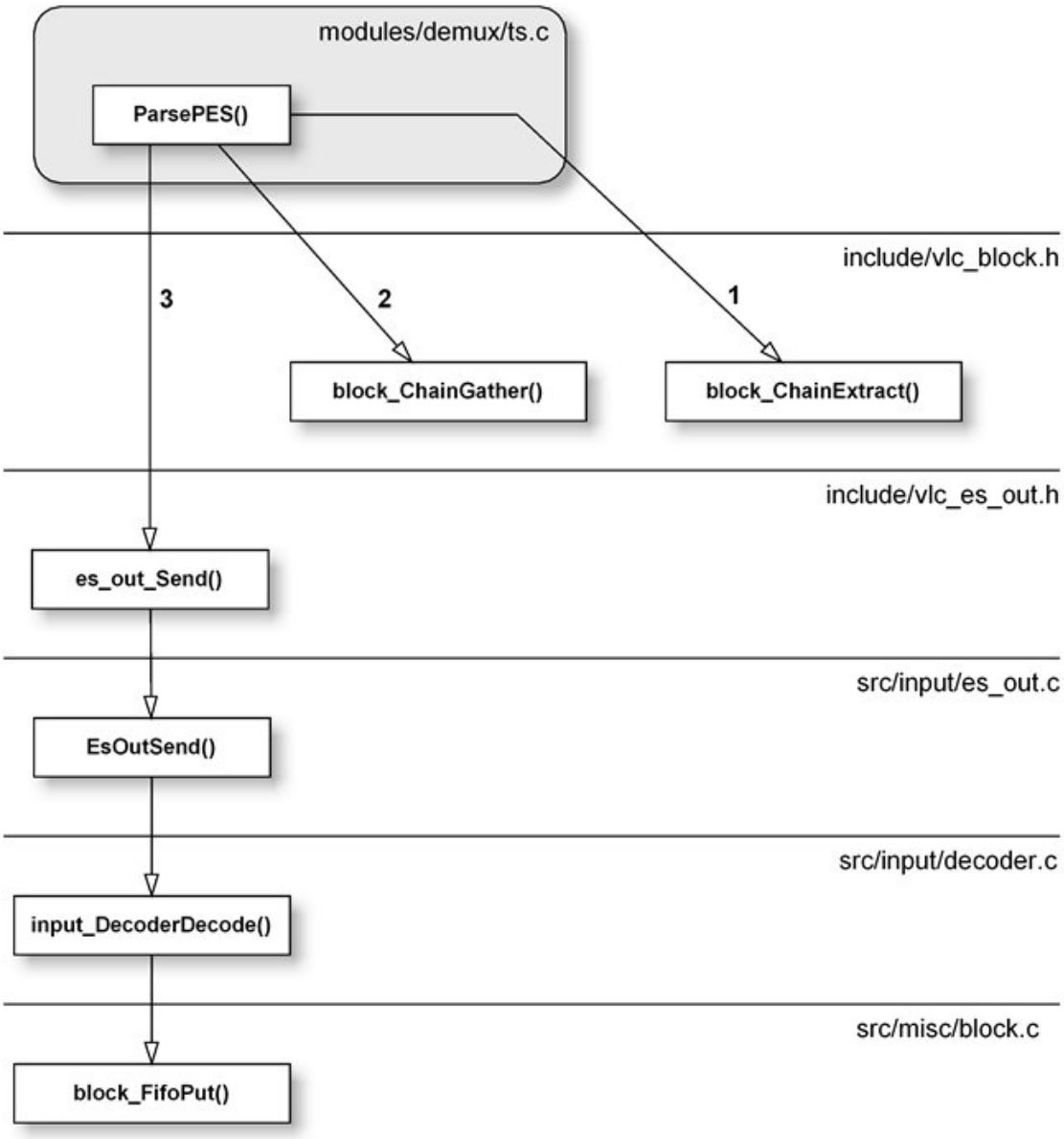


Figura 103: Parseo del PES

Una vez que se conoce el tamaño del header, el mismo es retirado del paquete PES:

```
/* skip header */
while( p_pes && i_skip > 0 )
{
    if( p_pes->i_buffer <= i_skip )
    {
        block_t *p_next = p_pes->p_next;

        i_skip -= p_pes->i_buffer;
        block_Release( p_pes );
        p_pes = p_next;
    }
    else
```

```
    {
        p_pes->i_buffer -= i_skip;
        p_pes->p_buffer += i_skip;
        break;
    }
}
```

Luego se invoca a la función block_ChainGather() la cual dado la cadena de paquetes TS correspondiente con el payload del paquete PES, la comprime en un único bloque, el cual luego es pasado a la función es_out_Send():

```
block_t *p_block;
p_block = block_ChainGather( p_pes );
.. .. ..
es_out_Send( p_demux->out, pid->es->id, p_block );
```

La funcion es_out_Send() llama a EsOutSend(), en donde se setéan los parámetros DTS y PTS del paquete PES y se llama a la función input_DecoderDecode(), quien inserta el paquete PES en una cola de paquetes (mediante la función block_FifoPut()) con el fin de que luego el decoder las consuma:

```
// El decoder se ejecuta en su propio thread como se vio anteriormente
if( p_dec->p_owner->b_own_thread )
{
    .. .. ..
    .. .. ..
    if( p_dec->p_owner->p_fifo->i_size > 50000000 /* 50 MB */ )
    {
        // Si el tamaño del FIFO supera los 50 MB el mismo es vaciado
        block_FifoEmpty( p_dec->p_owner->p_fifo );
    }

    block_FifoPut( p_dec->p_owner->p_fifo, p_block );
}
```

1.6.3. Proceso de Decoding

Como se puede observar en la Figura 94, en la creación del decoder (uno para cada ES) se crea un thread en donde se ejecuta la función DecoderThread(), cada thread a su vez tiene asociado un FIFO (`p_dec->p_owner->p_fifo`) en donde se irán colocando los bloques a decodificar.

A nivel general en proceso de decoding se puede considerar de la siguiente manera (observar la Figura 104):

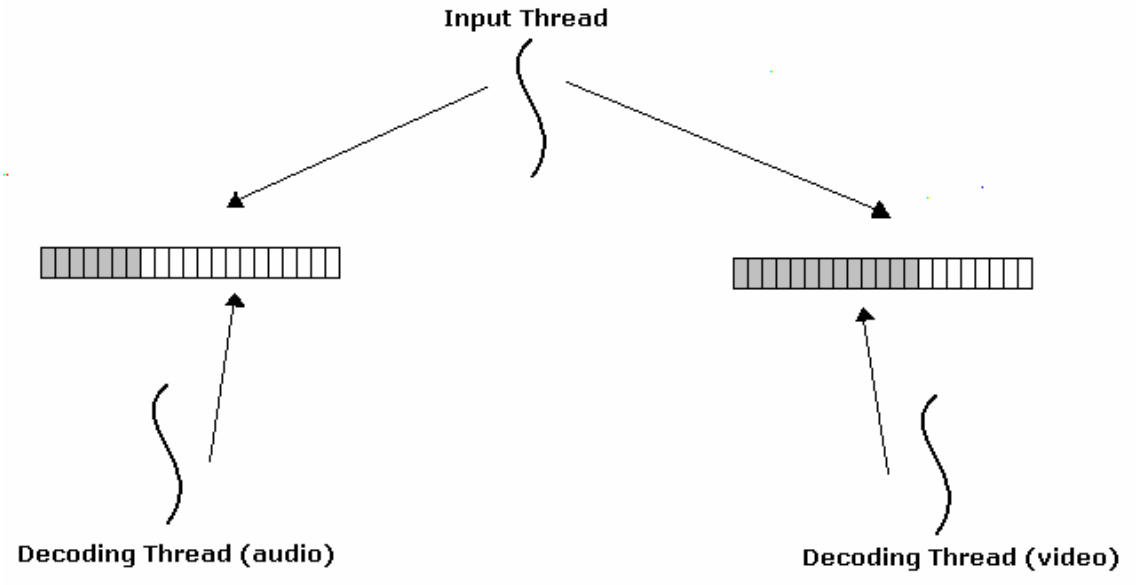


Figura 104: Proceso de Decoding

El input thread se encarga de parsear los paquetes TS, generando así los paquetes de tipo PES, los cuales son colocados en la FIFO del decoder correspondiente y en paralelo se encuentran los threads de los decoders los cuales van consumiendo los paquetes de estos FIFOs y los van decodificando.

Considerando el proceso de decoding a más bajo nivel, se tienen las interacciones presentadas en la Figura 95.

Dentro de la función DecoderThread(), se encuentra la iteración principal de decoding, como se puede observar en el siguiente código:

```
/* The decoder's main loop */
while( !p_dec->b_die && !p_dec->b_error )
{
    if( ( p_block = block_FifoGet( p_dec->p_owner->p_fifo ) ) == NULL )
    {
        p_dec->b_error = 1;
        break;
    }
    if( DecoderDecode( p_dec, p_block ) != VLC_SUCCESS )
    {
        break;
    }
}
```

En donde se extraen bloques de la FIFO propia del decoder (p_dec->p_owner->p_fifo) y se la decodifica mediante la función DecoderDecode().

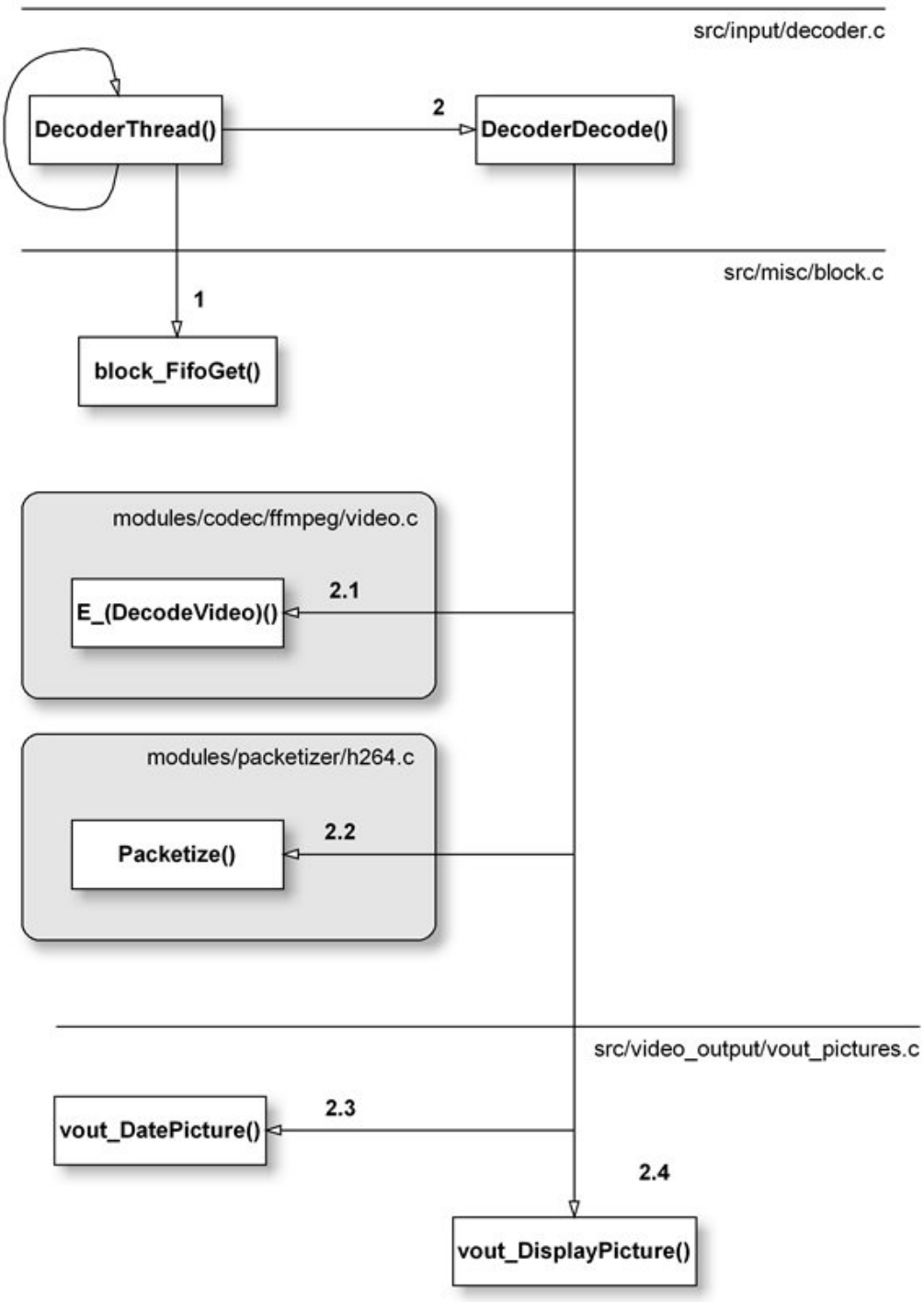


Figura 105: Decodificación de video

Interiorizándonos en la función DecoderDecode() y teniendo en cuenta nuestro ejemplo particular (streaming de tipo HTTP/MPEG-TS/H264) y a su vez recordando lo presentado en la sección 1.5.6 (puntualmente la Figura 95), en donde se mostraba la inicialización tanto un módulo de decoding como un módulo de packetizer (esto ultimo debido a que el módulo de decoding lo requería), es que dentro de la función DecoderDecode() existen llamadas tanto a Packetize() como a E_(DecodeVideo)().

Esto mismo se puede observar en esta porción de código extraída de la función DecoderDecode():

```
if( p_dec->fmt_in.i_cat == VIDEO_ES )
{
  picture_t *p_pic;

  if( p_dec->p_owner->p_packetizer )
  {
      block_t *p_packetized_block;
      decoder_t *p_packetizer = p_dec->p_owner->p_packetizer;

      while( (p_packetized_block =
              p_packetizer->pf_packetize( p_packetizer, &p_block )) )
      {
          ... ... ..
          while( p_packetized_block )
          {
              block_t *p_next = p_packetized_block->p_next;
              p_packetized_block->p_next = NULL;
              p_packetized_block->i_rate = i_rate;

              while( (p_pic = p_dec->pf_decode_video( p_dec,
                                                  &p_packetized_block )) )
              {
                ... ... ...
                vout_DatePicture( p_dec->p_owner->p_vout, p_pic,
                                  p_pic->date );
                vout_DisplayPicture( p_dec->p_owner->p_vout, p_pic );
              }

              ... ... ..
              p_packetized_block = p_next;
          }
      }
  }
}
```

No se profundizará en la implementación de las funciones de packetizer, ni de las de decoding, ni tampoco de las funciones de video output, debido a que éstas no son de interés para el desarrollo de nuestro proyecto.

1.7. Ejemplo de Ejecución (a nivel de red)

A continuación se presentará datos acerca de los paquetes que fluyen desde el servidor de streaming hacia el cliente multimedia (y viceversa) durante la ejecución de un streaming HTTP/MPEG-TS/H264.

Para registrar los datos que se presentaran a continuación, se usaron las siguientes aplicaciones: Ethereal (network protocol analyzer) versión 0.99.0 y TSReader Lite versión 2.7.44.

El ejemplo consiste en la transmisión de la secuencia de prueba "Elephants_Dream-h264.mpg". El servidor de streaming fue localizado en la IP 10.10.1.81 (Ubuntu) mientras que el cliente se localizó en la IP 10.10.1.122 (Windows XP) en donde a su vez se encontraban ejecutando las aplicación anteriormente mencionadas.

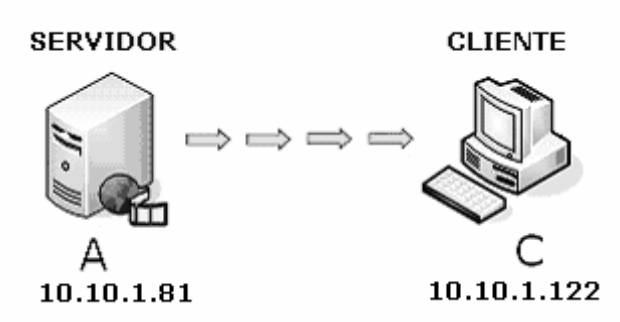


Figura 106: Arquitectura del ejemplo

Tanto en el cliente como en el servidor de streaming se ejecutaron una instancia del VLC, en la instancia localizada en el servidor de streaming se envía la secuencia de prueba hacia la IP 10.10.1.81, al puerto 1234, mientras que en la localizada en el cliente, se leyó dicho streaming.

En la Figura 26 se presenta la secuencia de paquetes establecida al comienzo de la transmisión del streaming, en donde como se puede observar en la misma, primeramente se resuelven la direcciones IP mediante el protocolo ARP, luego se establece la conexión TPC, para finalmente comenzar con la transmisión de la secuencia de paquetes correspondientes con la información propia del streaming.

Source	Destination	Protocol	Info
Asiarock_95:4a:29	Broadcast	ARP	Who has 10.10.1.81? Tell 10.10.1.122
Micro-St_b9:c9:ab	Asiarock_95:4a:29	ARP	10.10.1.81 is at 00:0c:76:b9:c9:ab
10.10.1.122	10.10.1.81	TCP	3229 > 1234 [SYN] Seq=0 Len=0 MSS=1460
10.10.1.81	10.10.1.122	TCP	1234 > 3229 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0
10.10.1.122	10.10.1.81	TCP	3229 > 1234 [ACK] Seq=1 Ack=1 Win=64240 Len=0
10.10.1.122	10.10.1.81	TCP	[TCP segment of a reassembled PDU]
10.10.1.81	10.10.1.122	TCP	1234 > 3229 [ACK] Seq=1 Ack=40 Win=5840 Len=0
10.10.1.122	10.10.1.81	TCP	[TCP segment of a reassembled PDU]
10.10.1.81	10.10.1.122	TCP	1234 > 3229 [ACK] Seq=1 Ack=160 Win=5840 Len=0
10.10.1.122	10.10.1.81	HTTP	GET / HTTP/1.1
10.10.1.81	10.10.1.122	TCP	1234 > 3229 [ACK] Seq=1 Ack=181 Win=5840 Len=0
10.10.1.81	10.10.1.122	HTTP	HTTP/1.0 200 OK
10.10.1.81	10.10.1.122	HTTP	Continuation or non-HTTP traffic

Figura 107: Secuencia de paquetes inciales de un Streaming

El primer paquete HTTP enviado desde el cliente hacia el servidor de streaming (mediante el cual se establece la transmisión del streaming) es el siguiente:

```
GET / HTTP/1.1
Host: 10.10.1.81:1234
User-Agent: VLC media player - version 0.8.5 Janus - (c) 1996-2006 the VideoLAN team
Range: bytes=0-
Icy-MetaData: 1
Connection: Close
```

La respuesta por parte del servidor de streaming:

```
HTTP/1.0 200 OK
Content-type: application/octet-stream
Cache-Control: no-cache
```

Luego de esta respuesta, el servidor de streaming ya comienza con la transmisión de los paquetes con información propia del streaming (el envió de los paquetes PES).

En la Tabla 13 se presenta un resumen de los datos registrados. El total de paquetes (dentro de estos paquetes se encuentran tanto paquetes TPC como paquetes ARP) transmitidos entre el cliente y el servidor es de 10326, refinando esta numero, la cantidad de paquetes enviados desde el cliente hacia el servidor es de 3588 paquetes mientras que la cantidad enviada desde el servidor hacia el cliente es de 6738 paquetes.

IP Cliente	10.10.1.122
Puerto Cliente	3229
IP Servidor	10.10.1.81
Puerto Servidor	1234
Total de Paquetes	10326
Totak de Bytes	10174508
Paquetes Cliente->Servidor	3588
Bytes Cliente->Servidor	193940
Paquetes Cliente<-Servidor	6738
Bytes Cliente<-Servidor	9980568

Tabla 13: Ejemplo Ejecución – Nivel de Red – Datos registrados

El tamaño del payload de los paquetes HTTP enviados por parte del servidor hacia el cliente y mediante los cuales se transmiten el streaming, es de 1460 bytes por lo que aproximadamente dentro de un paquete HTTP se pueden llegar a encontrar 7 paquetes TS (dado que cada paquete TS ocupa 188 bytes).

Tomando las estadísticas brindadas por el VLC (en la sección: Ver → Streaming and Media Info → Estadísticas) en la instancia localizada en el servidor de streaming, la misma indica que se transmitieron 50580 paquetes TS. Lo cual es consistente con los datos recogidos mediante el sniffer (si realizamos el siguiente cálculo: (cantidad paquetes TS) / (cantidad de paquetes HTTP enviados por el servidor), esto es 50580/6738, lo cual resulta en aproximadamente 7 paquetes TS por paquetes HTTP).

Finalmente y entrando en el detalle del streaming transmitido (datos recogidos mediante la aplicación TSReader) tenemos que (y como se puede observar en la Figura 108) el streaming esta constituido por un único programa (Programa 1), para el cual, la PMT se transmitirá en los paquetes con PID 66 (0x0042). Dicho programa esta compuesto por 2 ES, el de audio (Stream Type: 0x03 MPEG-1 Audio) el cual será transmitido con PID 68 (0x0044) y el de video (Stream Type: 0x1b H.264 Video) el cual será transmitido con PID 69 (0x0045). Finalmente los PCR para este programa se encontrarán en los paquetes con PID 69 (0x0045) es decir irán transmitidos periódicamente en los paquetes de video .

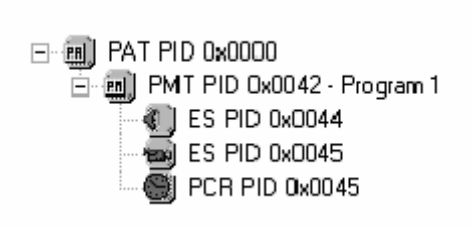


Figura 108: Estructura PAT y PMT del ejemplo

A su vez se tiene que el 80.11 % de los paquetes TS transmitidos corresponden con el PID 69 (ES de video), el 18.81 % corresponden con el PID 68 (ES de audio), el 0.54 con el PID 66 (información de PMT) y el 0.54 restante con el PID 0 (información de la PAT).

ANEXO II

2. Como agregar módulos al VLC

2.1. Introducción

Evaluamos importante encapsular nuestro trabajo en un conjunto de módulos. De esta forma podemos ejecutar el VLC con las funcionalidades habituales, o pasarle parámetros para que se habiliten las funcionalidades desarrolladas en este proyecto. En este anexo presentamos como crear un modulo en VLC para la versión 0.8.5.

2.2. Pasos para crear un módulo

1. Crear un directorio bajo modules, el cual va a contener los archivos de el módulo a crear. En nuestro caso dicho directorio lo llamaremos qos.

2. Una vez creado dicho directorio se debe crear un archivo de nombre Modules.am en donde se deben incluir todos los módulos que se depositarán dentro de este directorio (notar que pueden existir más de un módulo dentro de un directorio, estos directorios por lo general representan la categoría del módulo a agregar, por ejemplo dentro del directorio access, se encuentran los diferentes módulos de acceso, como puede ser el http.c, el udp.c, etc.). Para cada módulo a agregar, se debe indicar todos los archivos que lo componen a este. En nuestro caso nos será necesario 3 módulos.

La estructura del archivo Modules.am debe ser la siguiente:

```
SOURCES_module_1 = \
    module1_file1.x \
    module1_file2.x \
        ......
    module1_filem.x
...

SOURCES_module_n = \
    modulen_file1.x \
    modulen_file2.x \
        ......
    modulen_filem.x
```

Un detalle importante a considerar es que el nombre del módulo no debe incluir el prefijo “SOURCES_”.

En nuestro caso particular este es el contenido de dicho archivo:

```
SOURCES_qos = \
        qos_common.h \
        qos.c \
        lstStream.h \
        lstStream.c \
        genericVars.h \
        genericVars.c \
        frameVars.h \
        frameVars.c \
        vlcAuditMib_enums.h \
        audit_mapping.h \
        audit_mapping.c \
        logFormat.h \
        logFormat.c
SOURCES_snmp = \
        qos_common.h \
        vlcAuditMib.h \
        vlcAuditMib.c \
        vlcAuditMib_columns.h \
        vlcAuditMib_enums.h \
        snmp.c
SOURCES_loggerAudit = \
        loggerAudit.c \
        qos_common.h \
        loggerAudit_common.h \
        rotatelogs.h \
        rotatelogs.c
```

3. Uno de los archivos del módulo debe cumplir con ciertos requisitos, para ser el punto de entrada del módulo, es decir quien sea invocado en el caso en que el módulo sea cargado.

Básicamente se debe incluir el macro de definición de módulos en donde se debe:

- Definir su capacidad. En nuestro caso es la capacidad "interface".
- Definir la categoría y sub-categoria. En nuestro caso CAT_ADVANCED y SUBCAT_ADVANCED_MISC
- Brindar una descripción y un nombre corto.
- Definir las funciones “callbacks”.
- Implementar las funciones de “callbacks”

Las declaraciones incluidas en el archivo punto de entrada de nuestro módulo fueron las siguientes:

```
static int  Open    ( vlc_object_t * );
static void Close   ( vlc_object_t * );
static void Run     ( intf_thread_t * );
```

```
vlc_module_begin();
    set_shortname( N_( "LoggerAt" ) );
    set_description( _("Logger Audit") );

    set_category( CAT_ADVANCED );
    set_subcategory( SUBCAT_ADVANCED_MISC );

    set_capability( "interface", 0 );
    set_callbacks( Open, Close );
vlc_module_end();
```

4. Una vez definido lo establecido en el paso 3, es necesario indicarle al vlc que hemos agregado nuevos módulos y que debe compilar dichos módulos junto con el makefile del vlc.

Para eso es necesario modificar el archivo configure.ac (el cual se encuentra en la raíz del directorio del vlc) de la siguiente manera:

a) Primero debemos declarar el nuevo directorio creado con el fin de que el VLC reconozca la existencia de dicho directorio.

Para eso debemos buscar la definición del AC_CONFIG_FILES, en donde se presenta una lista de todos los Makefiles que se encuentran en el VLC. Debemos agregar una línea con la ubicación del Makefile (el cual se explicará mas adelante como crear dicho archivo), dicho Makefile debe compilar todos los módulos incluidos dentro del nuevo directorio.

En nuestro caso añadiremos la siguiente línea:

```
modules/qos/Makefile
```

El resultado es el siguiente:

```
AC_CONFIG_FILES([
  modules/access/Makefile
  modules/access/dshow/Makefile
  ...
  modules/visualization/galaktos/Makefile
  modules/qos/Makefile])
```

b) Hasta aqui solo hemos declarado la existencia del nuevo directorio, pero con el fin que al ejecutar el makefile del VLC se ejecute también el makefile de nuestro módulo, hace falta aun un paso más.

Se debe incluir la siguiente línea en el configure.ac:

```
VLC_ADD_PLUGINS([modulo_n])
```

Para incluir la línea en el lugar correcto debemos buscar el lugar en donde se encuentran los módulos que se compilan por defecto. Para esto debemos buscar el comienzo de los "default modules" e incluir debajo de estos nuestro módulo.

De esta forma, la sección del configure.ac donde nos estamos concentrando quedaría de la siguiente manera:

```
dnl
dnl  default modules
dnl
VLC_ADD_PLUGINS([dummy logger memcpy])
VLC_ADD_PLUGINS([mpgv mpga m4v m4a h264 ps pva avi asf mp4 rawdv nsv
real aiff mjpeg demuxdump flac])
...
VLC_ADD_PLUGINS([packetizer_mpeg4video packetizer_mpeg4audio])
VLC_ADD_PLUGINS([qos])
VLC_ADD_PLUGINS([snmp])
VLC_ADD_PLUGINS([loggerAudit])
```

5. Es importante aclarar que los Makefiles son archivos generados automáticamente. En este tipo de proyectos, los Makefile tienen un grado de complejidad tal, que sería prácticamente imposible mantener un Makefile manualmente.

Para esto es que se encuentra el archivo configure, el cual es un archivo cuyo contenido es escrito en un lenguaje de scripting, ubicado en la raíz del directorio del VLC, el cual se encarga de generar el Makefile. Al configure se le puede pasar parámetros tales como el directorio de donde se encuentra ubicada cierta librería, o la habilitación o des-habilitación de cierta librería del VLC.

A su vez, el configure es un archivo generado por el bootstrap. Dicho programa (el bootstrap) leerá el configure.ac para generar el configure. Por lo tanto, cada vez que se cambie algo en el configure.ac, se deberá ejecutar el bootstrap, con el fin de generar nuevamente el configure.

La secuencia de ejecución de comandos es la siguiente:

```
> ./bootstrap

> ./configure configure-options

> make

> make install
```

En nuestro caso particular, la ejecución del configure, fue realizada con los siguientes parámetros:

```
> ./configure --enable-x11 --enable-xvideo --disable-gtk --enable-sdl
--enable-ffmpeg          --with-ffmpeg-mp3lame          --with-ffmpeg-
tree=/opt/ffmpeg/ffmpeg-20051126  --enable-mad  --enable-libdvbpsi  --
enable-a52  --enable-dts  --enable-libmpeg2  --enable-dvdnav  --enable-
faad  --enable-vorbis  --enable-ogg  --enable-theora  --enable-faac  --
enable-mkv --enable-freetype --enable-fribidi --enable-speex --enable-
flac      --enable-caca      --enable-wxwidgets      --with-wx-config-
path=/opt/wxwidgets/wxWidgets-2.6.3/ --disable-skins2 --disable-kde --
disable-qt  --enable-release  --enable-esd  --enable-alsa  --enable-
livedotcom                          --with-livedotcom-tree=/opt/live
```

6. Para la correcta ejecución del vlc, es necesario setear ciertas variables de entorno, para que puedan encontrar tanto las librerías, como los .h.

Se debe setear las siguientes variables:

```
> CPPFLAGS="-I/usr/local/include":$CPPFLAGS
> LDFLAGS="-L/usr/local/lib":$LDFLAGS
> PKG_CONFIG_PATH="/usr/local/pkgconfig":$PKG_CONFIG_PATH
> LD_LIBRARY_PATH="/usr/local/lib":$LD_LIBRARY_PATH
> export CPPFLAGS LDFLAGS PKG_CONFIG_PATH LD_LIBRARY_PATH
```

Luego, para ejecutar el vlc, se debe ejecutar el siguiente comando:

```
> ./vlc
```

Para ejecutar el vlc, con nuestro módulo cargado es necesario pasarle un parámetro al vlc:

```
> ./vlc --extraint modulo_n
```

En nuestro caso particular:

```
> ./vlc --extraint loggerAudit
```

ANEXO III

3. Módulos de Stream Out

3.1. Introducción

En este apéndice veremos una breve introducción referente a los módulos de stream out provistos por el VLC, junto con la manera en que estos módulos se integran en el proceso de ejecución presentado en este documento. Para esto nos centraremos en el módulo de “Display”.

3.2. Módulos de Stream Out

Los módulos de stream out son módulos encargados de realizar acciones sobre el flujo de streaming. Estos son representados por una cadena de texto, con el siguiente formato:

```
:sout=#module_name{param1=value1, param2=value2, ..., paramN=valueN}
```

En este caso será ejecutado el módulo “module_name” pasándole los parámetros “param1”, “param2”, ..., “paramN” con sus valores asociados.
Un ejemplo de esto seria el siguiente comando:

```
:sout=#duplicate{dst=display}
```

A su vez estos módulos se pueden componer entre si aumentando asi la potencialidad brindada por el VLC:

```
:sout=#module_name{param1=value1, param2=module_name2{param1=value1}}
```

Por ejemplo:

```
:sout=#duplicate{dst=display,dst=std{access=http,mux=ts,dst=127.0.0.1:1234}}
```

Algunos de estos módulos pueden ser seteados mediante la interfaz gráfica, desde las opciones de volcado de salida.

Entre estos se encuentran el módulo de “Duplicate” (duplicate) encargado de duplicar la salida del stream por diferentes medios, el módulo de “Display” (display) encargado de realizar el volcado del stream en pantalla, el módulo de “Transcode” (transcode) encargado de realizar el transcoding de un formato a otro, el módulo “Standard” (std) encargado de volcar un streaming a la red o a un archivo, etc.

3.3. Integración con el proceso de streaming

A continuación se presentará como es que se integran estos módulos dentro del proceso de streaming. Para esto será necesario observar la Figura 109 en donde se presenta las diferentes interacciones dentro de la función Init() del src/input/input.c, en donde se inicializa el proceso de streaming.

Lo presentado en dicha figura corresponde con la ejecución de un streaming sin haber seteado ningún módulo de stream out, como veremos a continuación cuando algunos de estos módulos es seteado el proceso cambia con respecto a lo presentado en el documento.

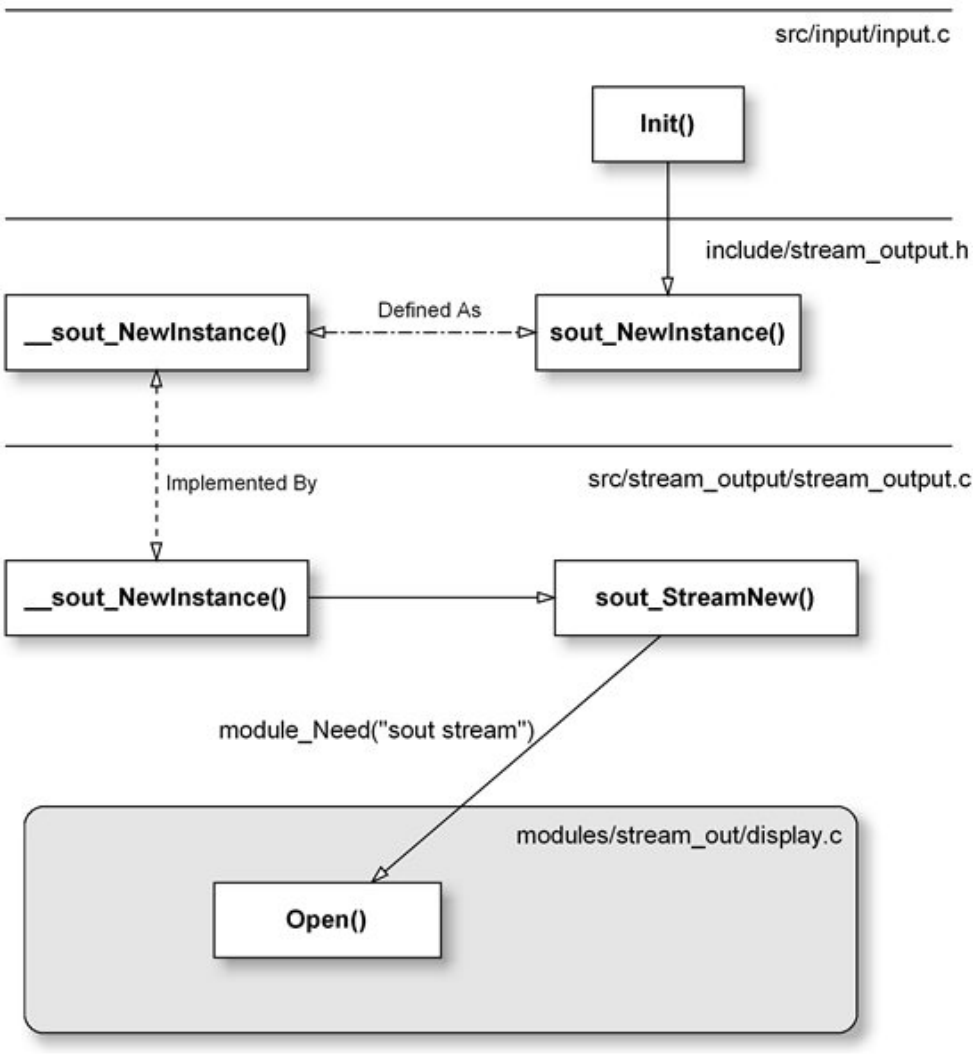


Figura 109: Modulo Stream Out en el proceso de Streaming

Dentro de la función Init() antes de la llamada a input_EsOutNew() (presentada en la Figura 109), se encuentra el siguiente código:

```
/* handle sout */
```

```
// Aquí si extrae el comando referente al stream output:
psz = var_GetString( p_input, "sout" );
if( *psz && strncasecmp( p_input->input.p_item->psz_uri, "vlc:", 4 ) )
{
    p_input->p_sout = sout_NewInstance( p_input, psz );
    if( p_input->p_sout == NULL )
    {
        msg_Err( p_input, "cannot start stream output instance, " \
                          "aborting" );
        free( psz );
        return VLC_EGENERIC;
    }
}
free( psz );
```

Como se puede observar en dicho código se extrae la cadena la cual representa los módulos de stream output y si esta no es vacía se llama a sout_NewInstance() pasándole dicha cadena (observar la Figura 108).

La función sout_NewInstance() será la encargada de inicializar los diferentes módulos de stream out, en nuestro caso, dado que la cadena pasada fue ":sout=#display" únicamente se iniciara el módulo de Display.

Luego de la inicialización de los módulos de stream out, el proceso de inicialización del streaming continua igual a lo presentado en el documento hasta el momento de la creación de los módulos de decoding (ver Sección 1.5.6). En particular el cambio ocurre dentro de la llamada a la función input_DecoderNew() (observar Figura 94).

Dentro de esta función existe el siguiente código:

```
/* If we are in sout mode, search for packetizer module */
if( p_input->p_sout && !b_force_decoder )
{
    /* Create the decoder configuration structure */
    p_dec = CreateDecoder( p_input, fmt, VLC_OBJECT_PACKETIZER );
    if( p_dec == NULL )
    {
        msg_Err( p_input, "could not create packetizer" );
        return NULL;
    }
}
else
{
    /* Create the decoder configuration structure */
    p_dec = CreateDecoder( p_input, fmt, VLC_OBJECT_DECODER );
    if( p_dec == NULL )
    {
        msg_Err( p_input, "could not create decoder" );
        return NULL;
```

```
    }
}
```

En la ejecución de streaming sin módulos de stream output seteados el p_input->p_sout es NULL por lo que la ejecución entra en el “else” de la condición, creado un decoder de tipo VLC_OBJECT_DECODER. En el caso en que haya módulos de stream output se cumplirá la condición del “if”, por lo que se creará un decoder de tipo VLC_OBJECT_PACKETIZER.

Por lo que en este caso no vamos a tener un decoder el cual decodifique los paquetes PES sino que vamos a tener un módulo de packetizer.

El proceso de ejecución del streaming va a continuar incambiado hasta la etapa de decodificación (ver Sección 1.6.3) en donde dentro de la función DecoderDecode() en lugar de ejecutar el código presentado en dicha sección (correspondiente con la decodificación de los bloques) se ejecutará el siguiente código:

```
if( p_dec->i_object_type == VLC_OBJECT_PACKETIZER )
{
    block_t *p_sout_block;

    while( ( p_sout_block =
                 p_dec->pf_packetize( p_dec, p_block ? &p_block : 0 ) ) )
    {
        if( !p_dec->p_owner->p_sout_input )
        {
            ... ... ...
            p_dec->p_owner->p_sout_input =
                sout_InputNew( p_dec->p_owner->p_sout,
                               &p_dec->p_owner->sout );
            ... ... ...
        }

        while( p_sout_block )
        {
            block_t *p_next = p_sout_block->p_next;
            ... ... ...
            sout_InputSendBuffer( p_dec->p_owner->p_sout_input,
                                  p_sout_block );

            p_sout_block = p_next;
        }
    }
}
```

La primer vez que se ingrese al “while” se ingresará al segundo “if”, llamándose así a la función sout_InputNew() llevándose a cabo el proceso presentado en la Figura 110.

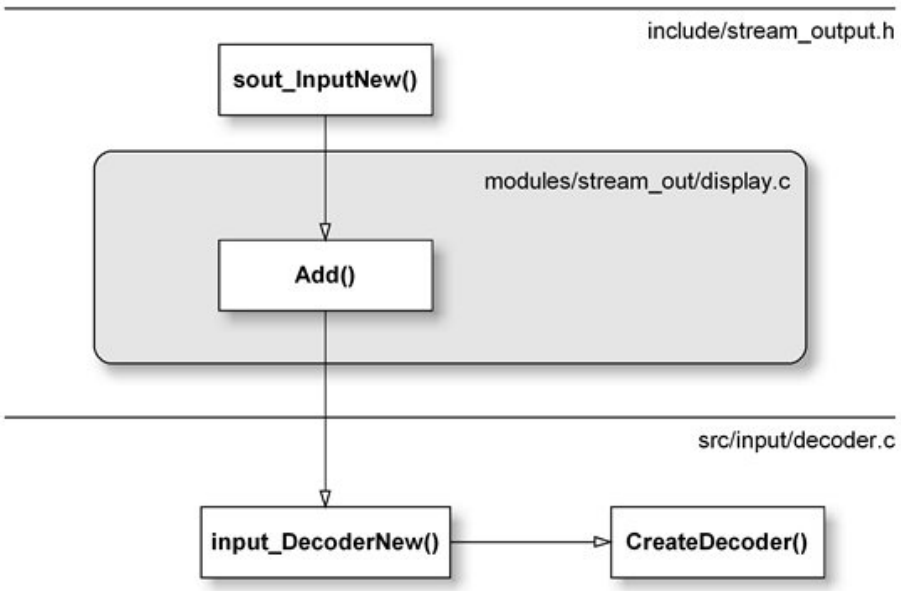


Figura 110: Creación del Decoder por Modulo de Stream Output

La llamada a la función input_DecoderNew() presentada en la Figura 110, es equivalente a la presentada en la Figura 94, por lo que el proceso de inicialización del decoder será el mismo que el presentado en la Sección 1.5.6.

Luego de este proceso de inicialización de los decoders, se encuentra una llamada a la función sout_InputSendBuffer(), la cual como se puede observar en la Figura 111, se encarga de agregar los bloques paquetizados, en la FIFO del decoder correspondiente.

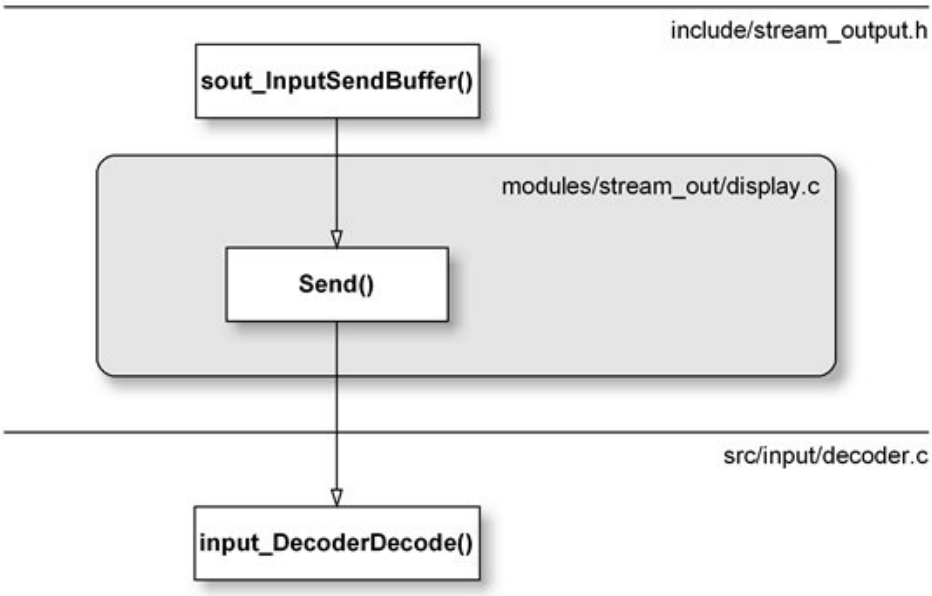


Figura 111: Envio de paquetes desde Modulo de Stream Output

Esta llamada a la función input_DecoderDecode(), a su vez es equivalente a la presentada en la Figura 103, de donde se desencadenará todo el proceso de decoding.

En conclusión se podría decir que los módulos de stream out son posibles funcionalidades las cuales son incorporados entre el proceso de lectura y parseo de datos (independientemente de la fuente de estos) y el proceso de decodificación de los mismos.

Esto último se puede ver reflejado, al comparar la Figura 112, la cual presenta una visión a muy alto nivel del proceso de decoding de un stream sujeto a ejecución del módulo de Display, con la Figura 104 en la cual se presenta la ejecución normal de dicho proceso.

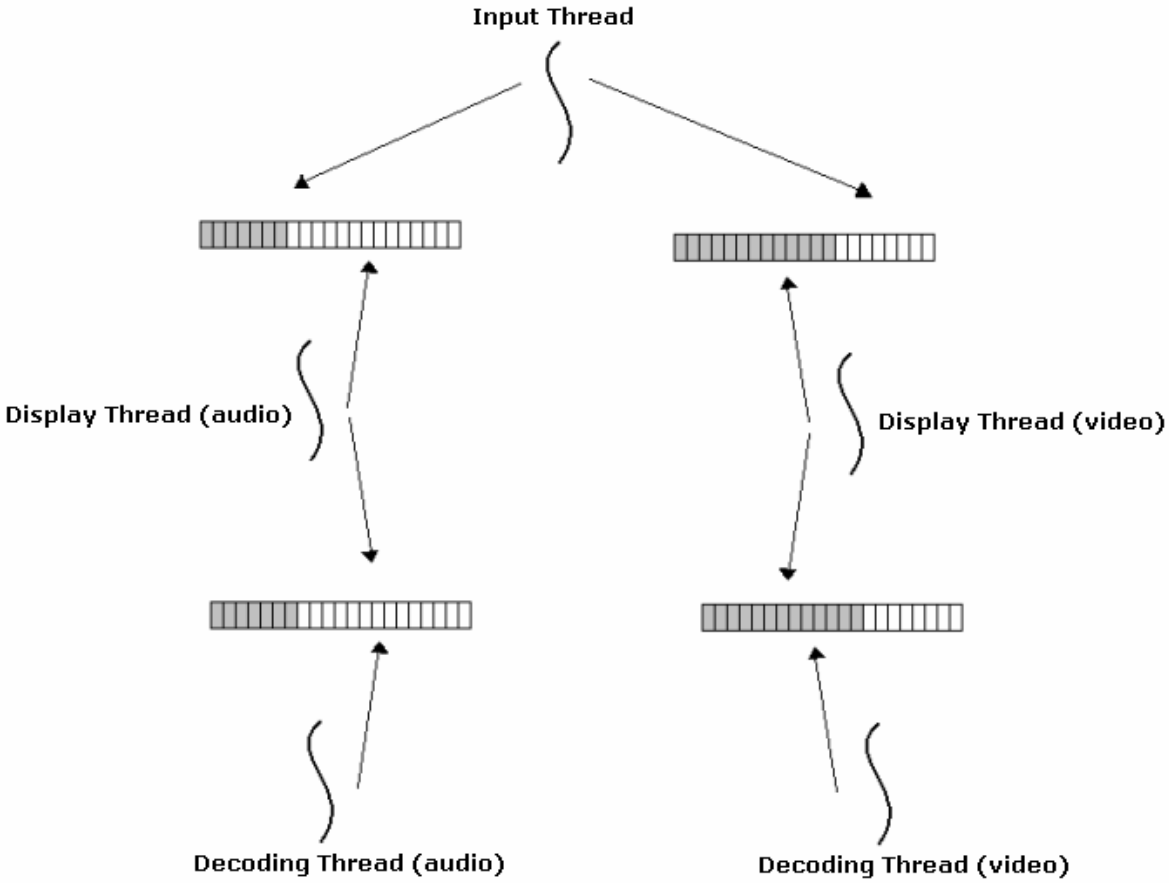


Figura 112: Ejecución de Decoding con Modulo de Stream Out (Display)

4. MIBS

4.1. Introducción

Este anexo contiene una descripción de como definir un MIB, como implementarla, como compilarla y como crear un subagente el cual responda a pedidos realizados sobre la misma.

4.2. Instalación

Se debe instalar la librería NET SNMP (snmp en el repositorio), el agente NET SNMP (snmpd en el repositorio) y el libnet-snmp-perl para tener acceso al comando mib2c.

4.3. Configuración del demonio

Una vez instalado se debe ejecutar el comando snmpconf, este mostrará un wizard mediante el cual vamos a configurar el demonio snmpd. Dos puntos importantes a tener en cuenta dentro de este wizard es que debemos definir el nombre de usuario y pass (si vamos a usar v3) o nombre de la comunidad (si usamos v2 o v1) y se debe habilitar el master agentx.

4.4. Definicion de una MIB

Para definir una MIB (lo cual consiste en un archivo de texto plano) existe una cierta sintaxis llamada SMI (Structure of Management Information) [Ref15], la cual se define en los RFC 1155 (SMIv1), 1212 (Consice MIBS Definitions), 2578 (SMIv2), 2579 (Textual Conventions).

Una vez definida la MIB, llamémosle VLC-TEST-MIB, la misma es recomendable validarla con algún validador de MIBS por ejemplo [Ref16].

Luego debemos hacer que la librería SNMP reconozca esta MIB, para eso debemos copiar la definición de la MIB a /usr/share/snmp/mibs (de donde la librería NET SNMP lee las MIBS) y debemos incluirla en la variable de entorno MIBS, por ejemplo:

```
cp VLC-TEST-MIB.txt /usr/share/snmp/mibs
export MIBS=+VLC-TEST-MIB
```

Para chequear que todo haya salido bien se puede ejecutar el siguiente comando:

```
-- % snmptranslate -M+. -mVLC-TEST-MIB -Tp -IR vlcTestMIB
--
-- +--vlcTestMIB(7)
--    |
--    +-- -RW- Integer32 vlcTestIntValue(1)
--    +-- -RW- IpAddr    vlcTestIPValue(2)
```

Donde vlcTestMIB es el nombre que le pusimos a nuestra MIB (notar que puede ser diferente del nombre del archivo donde se encuentra definida (VLC-TEST-MIB))

En el Apéndice A se presenta un ejemplo de una definición de una MIB.

4.5. Implementacion de la MIB

Una vez que tenemos la definición la MIB es necesario realizar una implementación de misma para que se la pueda agregar a un agente (o subagente) y se puedan realizar solicitudes sobre las diferentes propiedades de la misma.

Para esto se debe ejecutar el comando mib2c (en el ejemplo mib2c vlcTestMIB) lo cual desplegara un wizard el cual dependiendo de la MIB a implementar sugerirá diferentes opciones. La salida de este comando es un conjunto de archivos C, (.c y .h), los cuales son un template de la MIB, teniendo comentado donde debemos insertar el código particular de muestra implementación (por lo general que debemos implementar son los métodos de Get, Set y de Traps).

En el Apéndice B se presenta un ejemplo de template generado con esta herramienta.

4.6. Creacion de un subagente

Un subagente es un proceso creado por nosotros, el cual se adjunta al agente maestro snmp (el snmpd) quienes se comunican entre si mediante el protocolo AgentX. El NET SNMP, provee primitivas para crear un subagente.

En el Apéndice C se presenta un ejemplo de la implementación de un subagente, por mas detalles consultar [Ref17].

4.7. Ejecucion del Subagente

Por ultimo para verificar que todo se haya implementado e integrado de forma correcta, se puede ejecutar el subagente:

```
./example-demon &
```

y realizar alguna solicitud sobre la MIB definida:

```
snmpget -v 2c -c ddv_com localhost VLC-TEST-MIB::vlcTestIPValue.0
```

4.8. VLC-AUDIT-MIB

La siguiente es la mib definida para la transmisión de los datos recogidos por nuestro sistema.

```
--
-- VLC-AUDIT-MIB.my
--

     VLC-AUDIT-MIB DEFINITIONS ::= BEGIN

         IMPORTS
           netSnmpExamplesFROM NET-SNMP-EXAMPLES-MIB
           OBJECT-GROUP, MODULE-COMPLIANCE, NOTIFICATION-GROUP
                FROM SNMPv2-CONF
           IpAddress, Integer32, Unsigned32, Counter32, OBJECT-TYPE, Counter64,
           MODULE-IDENTITY, NOTIFICATION-TYPE
                FROM SNMPv2-SMI
           TruthValue, DateAndTime, DisplayString, RowStatus, TEXTUAL-
CONVENTION
                FROM SNMPv2-TC;


         vlcAuditMib MODULE-IDENTITY
           LAST-UPDATED "200608191400Z"
           ORGANIZATION
                "FING - UDELAR"
           CONTACT-INFO
                "Proyecto AUDIT
                Facultad De Ingenieria
                Montevideo, Uruguay
                Julio Herrera y Reissig 565
                Codigo Postal 11.300
                Tel: +05982 711 06 98
                Fax: +05982 711 54 46"
           DESCRIPTION
                "Esta MIB brinda soporte a las diferentes mediciones
establecidas
                 en el proyecto AUDIT"
           ::= { netSnmpExamples 7 }

--
-- Textual conventions
--

         StreamNumber ::= TEXTUAL-CONVENTION
           DISPLAY-HINT
                "d"
           STATUS current
           DESCRIPTION
                "StreamNumber es usado para identificar un stream
                 dentro de la tabla de Streams"
           SYNTAX INTEGER (1..10)

         StreamingProtocol ::= TEXTUAL-CONVENTION
           STATUS current
           DESCRIPTION
                "Enumerado con los diferentes protocolos de Streaming"
```

```
  SYNTAX INTEGER
       {
       http(1),
       udp(2),
       rtp(3),
       mmsh(4),
       unknown(9999)
       }

StreamingMuxer ::= TEXTUAL-CONVENTION
  STATUS current
  DESCRIPTION
       "Enumerado con los diferentes Muxers"
  SYNTAX INTEGER
       {
       mpegts(1),
       mpegps(2),
       mpeg1(3),
       ogg(4),
       asf(5),
       mp4(6),
       mov(7),
       wav(8),
       raw(9),
       unknown(9999)
       }

StreamingVideoCodec ::= TEXTUAL-CONVENTION
  STATUS current
  DESCRIPTION
       "Enumerado con los diferentes codecs de
        video"
  SYNTAX INTEGER
       {
       mp1v(1),
       mp2v(2),
       mp4v(3),
       div1(4),
       div2(5),
       div3(6),
       h263(7),
       h264(8),
       wmv1(9),
       wmv2(10),
       mjpg(11),
       theo(12),
       mpgv(13),
       unknown(9999)
       }

StreamingAudioCodec ::= TEXTUAL-CONVENTION
  STATUS current
  DESCRIPTION
       "Enumerado con los diferentes codecs de
        audio"
  SYNTAX INTEGER
       {
       mpga(1),
       mp2a(2),
       mp3(3),
       mp4a(4),
       a52(5),
       vorb(6),
       flac(7),
       spx(8),
       s16l(9),
```

```
                fl32(10),
                unknown(9999)
                }

          StreamingSubtitleCodec ::= TEXTUAL-CONVENTION
            STATUS current
            DESCRIPTION
                "Enumerado con los diferentes codecs de
                 subtitulos"
            SYNTAX INTEGER
                {
                dvbs(1),
                unknown(9999)
                }

          ActiveTime ::= TEXTUAL-CONVENTION
            DISPLAY-HINT
                "d"
            STATUS current
            DESCRIPTION
                "Este valor es ingrementado en uno cada un segundo"
            SYNTAX Unsigned32

-- Mediciones de calidad percibida por el usuario final en los streaming
-- Proyecto AUDIT 2006
--
-- Variables Generales a todos los streams
--

          vlcStreamsTable OBJECT-TYPE
            SYNTAX SEQUENCE OF VLCStreamsEntry
            MAX-ACCESS not-accessible
            STATUS current
            DESCRIPTION
                "The tsTestsSummaryTable provides access to the state of all of
                the Transport Stream tests enumerated in
IndexTransportStreamTest.
                The status relates to the whole Transport Stream.  In the case
                of tests which have a status per PID, the tsTestsSummaryTable
                gives the 'worst' status across all the PIDs and the status for
                each PID is available in tsTestsPIDTable."
            REFERENCE
                "TR 101 290 section 5.2"
            ::= { vlcAuditMib 1 }

          vlcStreamsEntry OBJECT-TYPE
            SYNTAX VLCStreamsEntry
            MAX-ACCESS not-accessible
            STATUS current
            DESCRIPTION
                "Row specification"
            INDEX { vlcStreamsStreamNumber }
            ::= { vlcStreamsTable 1 }

          VLCStreamsEntry ::=
            SEQUENCE {
                vlcStreamsStreamNumber
                  StreamNumber,
                vlcStreamsSessionNumber
                  Unsigned32,
                vlcStreamsURL
                  DisplayString,
                vlcStreamsProtocol
                  StreamingProtocol,
                vlcStreamsIP
                  IpAddress,
```

```
        vlcStreamsPort
          Unsigned32,
        vlcStreamsMuxer
          StreamingMuxer,
        vlcStreamsVideoCodec
          StreamingVideoCodec,
        vlcStreamsAudioCodec
          StreamingAudioCodec,
        vlcStreamsSubtitleCodec
          StreamingSubtitleCodec,
        vlcStreamsStartTime
           DisplayString,
        vlcStreamsExecutionTime
          ActiveTime
    }

vlcStreamsStreamNumber OBJECT-TYPE
  SYNTAX StreamNumber
  MAX-ACCESS read-only
  STATUS current
  DESCRIPTION
       "Identificador de Stream"
  ::= { vlcStreamsEntry 1 }

vlcStreamsSessionNumber OBJECT-TYPE
  SYNTAX Unsigned32
  MAX-ACCESS read-only
  STATUS current
  DESCRIPTION
       "Identificador de Sesion del Stream, si el mismo
        es 0, esto indica que este stream no se encuentra
        en ejecucion"
  ::= { vlcStreamsEntry 2 }

vlcStreamsURL OBJECT-TYPE
  SYNTAX DisplayString
  MAX-ACCESS read-only
  STATUS current
  DESCRIPTION
       "URL del stream el cual se esta recibiendo"
  ::= { vlcStreamsEntry 3 }

vlcStreamsProtocol OBJECT-TYPE
  SYNTAX StreamingProtocol
  MAX-ACCESS read-only
  STATUS current
  DESCRIPTION
       "Protocolo mediante el cual el stream se esta
        recibiendo"
  ::= { vlcStreamsEntry 4 }

vlcStreamsIP OBJECT-TYPE
  SYNTAX IpAddress
  MAX-ACCESS read-only
  STATUS current
  DESCRIPTION
       "IP del stream el cual se esta recibiendo"
  ::= { vlcStreamsEntry 5 }

vlcStreamsPort OBJECT-TYPE
  SYNTAX Unsigned32
  MAX-ACCESS read-only
  STATUS current
  DESCRIPTION
       "Puerto mediante el cual el stream se esta
        transmitiendo desde el server"
```

```
          ::= { vlcStreamsEntry 6 }

        vlcStreamsMuxer OBJECT-TYPE
          SYNTAX StreamingMuxer
          MAX-ACCESS read-only
          STATUS current
          DESCRIPTION
               "Muxer del streaming"
          ::= { vlcStreamsEntry 7 }

        vlcStreamsVideoCodec OBJECT-TYPE
          SYNTAX StreamingVideoCodec
          MAX-ACCESS read-only
          STATUS current
          DESCRIPTION
               "Codec de video del streaming"
          ::= { vlcStreamsEntry 8 }

        vlcStreamsAudioCodec OBJECT-TYPE
          SYNTAX StreamingAudioCodec
          MAX-ACCESS read-only
          STATUS current
          DESCRIPTION
               "Codec de audio del streaming"
          ::= { vlcStreamsEntry 9 }

        vlcStreamsSubtitleCodec OBJECT-TYPE
          SYNTAX StreamingSubtitleCodec
          MAX-ACCESS read-only
          STATUS current
          DESCRIPTION
               "Codec de subtitulos del streaming,
                vacio si no aplica"
          ::= { vlcStreamsEntry 10 }

        vlcStreamsStartTime OBJECT-TYPE
          SYNTAX DisplayString
          MAX-ACCESS read-only
          STATUS current
          DESCRIPTION
               "The Timestamp shows the Real time at which the stream was
          detected.  The Timestamp format is YYYY/MM/DD/HH/MM/SS."
          ::= { vlcStreamsEntry 11 }

        vlcStreamsExecutionTime OBJECT-TYPE
          SYNTAX ActiveTime
          MAX-ACCESS read-only
          STATUS current
          DESCRIPTION
               "Tiempo de ejecucion del Stream"
          ::= { vlcStreamsEntry 12 }



--
-- Variables Dinamicas
--

     vlcDynamicTable OBJECT-TYPE
      SYNTAX      SEQUENCE OF VLCDynamicEntry
       MAX-ACCESS  not-accessible
       STATUS      current
       DESCRIPTION
           "Tabla con las variables dinamicas de VLC"
       ::= { vlcAuditMib 2 }
```

```
vlcDynamicEntry OBJECT-TYPE
   SYNTAX VLCDynamicEntry
   MAX-ACCESS not-accessible
   STATUS     current
   DESCRIPTION
        "IP Stream Dynamic Table Entry."
   INDEX  { vlcDynamicStreamNumber }
   ::= { vlcDynamicTable 1 }

VLCDynamicEntry ::= SEQUENCE {
   vlcDynamicStreamNumber
        StreamNumber,
    vlcDynamicSessionNumber
        Unsigned32,
   vlcDynamicIFramesExpected
       Unsigned32,
   vlcDynamicIFramesArrived
       Unsigned32,
   vlcDynamicPFramesExpected
       Unsigned32,
   vlcDynamicPFramesArrived
       Unsigned32,
   vlcDynamicBFramesExpected
       Unsigned32,
   vlcDynamicBFramesArrived
       Unsigned32,
   vlcDynamicIDLastPeriod
       Unsigned32,
   vlcDynamicLastPeriod
       Counter64,
   vlcDynamicIFramesSizePeriod
       Unsigned32,
   vlcDynamicIFramesErrorPeriod
       Unsigned32,
   vlcDynamicPFramesSizePeriod
       Unsigned32,
   vlcDynamicPFramesErrorPeriod
       Unsigned32,
   vlcDynamicBFramesSizePeriod
       Unsigned32,
   vlcDynamicBFramesErrorPeriod
       Unsigned32,
   vlcDynamicBitratePeriod
       Unsigned32
       }

vlcDynamicStreamNumber OBJECT-TYPE
      SYNTAX StreamNumber
      MAX-ACCESS read-only
      STATUS current
      DESCRIPTION
           "Identificador de Stream"
  ::= { vlcDynamicEntry 1 }

vlcDynamicSessionNumber OBJECT-TYPE
      SYNTAX Unsigned32
      MAX-ACCESS read-only
      STATUS current
      DESCRIPTION
           "Identificador de Sesion del Stream, si el mismo
            es 0, esto indica que este stream no se encuentra
            en ejecucion"
    ::= { vlcDynamicEntry 2 }

vlcDynamicIFramesExpected OBJECT-TYPE
   SYNTAX   Unsigned32
```

```
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
          "Cantidad total de I Frames enviados por el Streaming Server"
   ::= { vlcDynamicEntry 3 }

vlcDynamicIFramesArrived OBJECT-TYPE
   SYNTAX     Unsigned32
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
          "Cantidad total de I Frames recividos en el cliente"
   ::= { vlcDynamicEntry 4 }

vlcDynamicPFramesExpected OBJECT-TYPE
   SYNTAX   Unsigned32
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
          "Cantidad total de P Frames enviados por el Streaming Server"
   ::= { vlcDynamicEntry 5 }

vlcDynamicPFramesArrived OBJECT-TYPE
   SYNTAX     Unsigned32
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
          "Cantidad total de P Frames recividos en el cliente"
   ::= { vlcDynamicEntry 6 }

vlcDynamicBFramesExpected OBJECT-TYPE
   SYNTAX   Unsigned32
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
          "Cantidad total de B Frames enviados por el Streaming Server"
   ::= { vlcDynamicEntry 7 }

vlcDynamicBFramesArrived OBJECT-TYPE
   SYNTAX     Unsigned32
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
          "Cantidad total de B Frames recividos en el cliente"
   ::= { vlcDynamicEntry 8 }

vlcDynamicIDLastPeriod OBJECT-TYPE
   SYNTAX  Unsigned32
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
          "Id del ultimo periodo de medicion"
   ::= { vlcDynamicEntry 9 }

vlcDynamicLastPeriod OBJECT-TYPE
   SYNTAX Counter64
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
          "Duracion del ultimo periodo de medicion"
   ::= { vlcDynamicEntry 10 }

 vlcDynamicIFramesSizePeriod OBJECT-TYPE
   SYNTAX Unsigned32
   MAX-ACCESS read-only
   STATUS     current
```

```
      DESCRIPTION
             "Tamanio promedio de I Frames en el ultimo periodo de medicion"
      ::= { vlcDynamicEntry 11 }

    vlcDynamicIFramesErrorPeriod OBJECT-TYPE
      SYNTAX Unsigned32
      MAX-ACCESS read-only
      STATUS     current
      DESCRIPTION
             "I Frames perdidos en el ultimo periodo de medicion"
      ::= { vlcDynamicEntry 12 }

    vlcDynamicPFramesSizePeriod OBJECT-TYPE
      SYNTAX Unsigned32
      MAX-ACCESS read-only
      STATUS     current
      DESCRIPTION
            "Tamanio promedio de P Frames en el ultimo periodo de medicion"
      ::= { vlcDynamicEntry 13 }

    vlcDynamicPFramesErrorPeriod OBJECT-TYPE
      SYNTAX Unsigned32
      MAX-ACCESS read-only
      STATUS     current
      DESCRIPTION
             "P Frames perdidos en el ultimo periodo de medicion"
      ::= { vlcDynamicEntry 14 }

    vlcDynamicBFramesSizePeriod OBJECT-TYPE
      SYNTAX Unsigned32
      MAX-ACCESS read-only
      STATUS     current
      DESCRIPTION
             "Tamanio promedio de B Frames en el ultimo periodo de medicion"
      ::= { vlcDynamicEntry 15 }

    vlcDynamicBFramesErrorPeriod OBJECT-TYPE
      SYNTAX Unsigned32
      MAX-ACCESS read-only
      STATUS     current
      DESCRIPTION
             "B Frames perdidos en el ultimo periodo de medicion"
      ::= { vlcDynamicEntry 16 }

    vlcDynamicBitratePeriod OBJECT-TYPE
      SYNTAX Unsigned32
      MAX-ACCESS read-only
      STATUS     current
      DESCRIPTION
             "Bitrate en el ultimo periodo de medicion"
      ::= { vlcDynamicEntry 17 }

--
-- Variables Semi Estaticas (Promedios reportados por los agentes)
--

    vlcAverageTable OBJECT-TYPE
     SYNTAX      SEQUENCE OF VLCAverageEntry
      MAX-ACCESS  not-accessible
      STATUS      current
      DESCRIPTION
          "Tabla con las variables dinamicas de VLC"
      ::= { vlcAuditMib 3 }

   vlcAverageEntry OBJECT-TYPE
      SYNTAX VLCAverageEntry
```

```
   MAX-ACCESS not-accessible
   STATUS     current
   DESCRIPTION
        "IP Stream Average Table Entry."
   INDEX  { vlcAverageStreamNumber }
   ::= { vlcAverageTable 1 }

VLCAverageEntry ::= SEQUENCE {
   vlcAverageStreamNumber
       StreamNumber,
    vlcAverageSessionNumber
        Unsigned32,
   vlcAverageIFramesSize
       Unsigned32,
   vlcAveragePFramesSize
       Unsigned32,
   vlcAverageBFramesSize
       Unsigned32,
   vlcAverageBitrate
       Unsigned32
       }

vlcAverageStreamNumber OBJECT-TYPE
      SYNTAX StreamNumber
      MAX-ACCESS read-only
      STATUS current
      DESCRIPTION
           "Identificador de Stream"
  ::= { vlcAverageEntry 1 }

vlcAverageSessionNumber OBJECT-TYPE
      SYNTAX Unsigned32
      MAX-ACCESS read-only
      STATUS current
      DESCRIPTION
           "Identificador de Sesion del Stream, si el mismo
            es 0, esto indica que este stream no se encuentra
            en ejecucion"
    ::= { vlcAverageEntry 2 }

vlcAverageIFramesSize OBJECT-TYPE
   SYNTAX   Unsigned32
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
          "Tamanio promedio de I Frames"
   ::= { vlcAverageEntry 3 }

vlcAveragePFramesSize OBJECT-TYPE
   SYNTAX     Unsigned32
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
          "Tamanio promedio de P Frames"
   ::= { vlcAverageEntry 4 }

vlcAverageBFramesSize OBJECT-TYPE
   SYNTAX   Unsigned32
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
          "Tamanio promedio de B Frames"
   ::= { vlcAverageEntry 5 }

vlcAverageBitrate OBJECT-TYPE
   SYNTAX     Unsigned32
```

```
        MAX-ACCESS read-only
        STATUS     current
        DESCRIPTION
               "Bitrate Promedio"
        ::= { vlcAverageEntry 6 }


      END

--
-- VLC-AUDIT-MIB.my
--
```

4.9. Apéndice A - Ejemplo de Definición de una MIB

```
VLC-TEST-MIB DEFINITIONS ::= BEGIN

-- A Comment!

-- IMPORTS: Include definitions from other mibs here, which is always
-- the first item in a MIB file.
IMPORTS
      OBJECT-TYPE, Integer32,
      IpAddress, MODULE-IDENTITY,
      enterprises              FROM SNMPv2-SMI;

--
-- A brief description and update information about this mib.
--
vlcTestMIB MODULE-IDENTITY
    LAST-UPDATED "200604250000Z"            -- 25 April 2006, midnight
    ORGANIZATION "fing"
    CONTACT-INFO "email:    devera@adinet.com.uy"
    DESCRIPTION  "A simple mib for demonstration purposes."
    ::= { enterprises ut(785) 7 }


-- +--vlcTestMIB(7)
--    |
--    +-- -RW- Integer32 vlcTestIntValue(1)
--    +-- -RW- IpAddr    vlcTestIPValue(2)

vlcTestIntValue OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
      "Just Testing an Integer Value"
    DEFVAL { 1 }
    ::= { vlcTestMIB 1 }

vlcTestIPValue OBJECT-TYPE
    SYNTAX      IpAddress
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
      "Just Testing an IP value"
    DEFVAL { "127.0.0.1" }
    ::= { vlcTestMIB 2 }

END
```

4.10. Apendice B - Template generado con mibc2

Archivo: vlcTestMIB.h (en este archivo no hay que modificar nada)

```
/*
 * Note: this file originally auto-generated by mib2c using
 *        : mib2c.scalar.conf,v 1.7 2003/04/08 14:57:04 dts12 Exp $
 */
#ifndef VLCTESTMIB_H
#define VLCTESTMIB_H

/* function declarations */
void init_vlcTestMIB(void);
Netsnmp_Node_Handler handle_vlcTestIPValue;
Netsnmp_Node_Handler handle_vlcTestIntValue;

#endif /* VLCTESTMIB_H */
```

Archivo: vlcTestMIB.c

```
/*
 * Note: this file originally auto-generated by mib2c using
 *        : mib2c.scalar.conf,v 1.7 2003/04/08 14:57:04 dts12 Exp $
 */

#include <net-snmp/net-snmp-config.h>
#include <net-snmp/net-snmp-includes.h>
#include <net-snmp/agent/net-snmp-agent-includes.h>
#include "vlcTestMIB.h"

/** Initializes the vlcTestMIB module */
void
init_vlcTestMIB(void)
{
    static oid vlcTestIPValue_oid[] = { 1,3,6,1,4,1,785,7,2 };
    static oid vlcTestIntValue_oid[] = { 1,3,6,1,4,1,785,7,1 };

  DEBUGMSGTL(("vlcTestMIB", "Initializing\n"));

    netsnmp_register_scalar(
        netsnmp_create_handler_registration("vlcTestIPValue",
handle_vlcTestIPValue, vlcTestIPValue_oid,
OID_LENGTH(vlcTestIPValue_oid), HANDLER_CAN_RONLY ));

    netsnmp_register_scalar(
        netsnmp_create_handler_registration("vlcTestIntValue",
handle_vlcTestIntValue, vlcTestIntValue_oid,
OID_LENGTH(vlcTestIntValue_oid), HANDLER_CAN_RONLY ));
}


int handle_vlcTestIPValue(netsnmp_mib_handler *handler,
                          netsnmp_handler_registration *reginfo,
                          netsnmp_agent_request_info   *reqinfo,
                          netsnmp_request_info         *requests)
```

```
{
    /* We are never called for a GETNEXT if it's registered as a
       "instance", as it's "magically" handled for us.  */

 /* a instance handler also only hands us one request at a time, so
 we don't need to loop over a list of requests; we'll only get one. */

    switch(reqinfo->mode) {

      case MODE_GET:
    // -- Aca es donde va nuestra implementacion concreta --
            snmp_set_var_typed_value(requests->requestvb,
ASN_IPADDRESS, (u_char *) /* XXX: a pointer to the scalar's data */,
/* XXX: the length of the data in bytes */);

      break;


      default:
        /* we should never get here, so this is a really bad error */
        return SNMP_ERR_GENERR;
    }

    return SNMP_ERR_NOERROR;
}


int handle_vlcTestIntValue(netsnmp_mib_handler *handler,
                           netsnmp_handler_registration *reginfo,
                           netsnmp_agent_request_info   *reqinfo,
                           netsnmp_request_info         *requests)
{
    /* We are never called for a GETNEXT if it's registered as a
       "instance", as it's "magically" handled for us.  */

 /* a instance handler also only hands us one request at a time, so
 we don't need to loop over a list of requests; we'll only get one. */

    switch(reqinfo->mode) {

        case MODE_GET:
          // -- Aca es donde va nuestra implementacion concreta --
            snmp_set_var_typed_value(requests->requestvb, ASN_INTEGER,
               (u_char *) /* XXX: a pointer to the scalar's data */,
                /* XXX: the length of the data in bytes */);
            break;


        default:
         /* we should never get here, so this is a really bad error */
         return SNMP_ERR_GENERR;
    }

    return SNMP_ERR_NOERROR;
}
```

4.11. Apéndice C - Implementación de un subagente

```
#include <net-snmp/net-snmp-config.h>
#include <net-snmp/net-snmp-includes.h>
#include <net-snmp/agent/net-snmp-agent-includes.h>
#include <signal.h>

#include <vlcTestMIB.h>

static int keep_running;

RETSIGTYPE
stop_server(int a) {
    keep_running = 0;
}

int
main (int argc, char **argv) {
  int agentx_subagent=1; /* change this if you want to be a SNMP
master agent */
  int background = 0; /* change this if you want to run in the
background */
  int syslog = 0; /* change this if you want to use syslog */

  /* print log errors to syslog or stderr */
  if (syslog)
    snmp_enable_calllog();
  else
    snmp_enable_stderrlog();

  /* we're an agentx subagent? */
  if (agentx_subagent) {
    /* make us a agentx client. */
    netsnmp_ds_set_boolean(NETSNMP_DS_APPLICATION_ID,
NETSNMP_DS_AGENT_ROLE, 1);
  }

  /* run in background, if requested */
  if (background && netsnmp_daemonize(1, !syslog))
      exit(1);

  /* initialize tcpip, if necessary */
  SOCK_STARTUP;

  /* initialize the agent library */
  init_agent("example-demon");

  /* initialize mib code here */

  /* mib code: init_nstAgentSubagentObject from
nstAgentSubagentObject.C */
  init_vlcTestMIB();

  /* initialize vacm/usm access control  */
  if (!agentx_subagent) {
      init_vacm_vars();
```

```
      init_usmUser();
  }

  /* example-demon will be used to read example-demon.conf files. */
  init_snmp("example-demon");

  /* If we're going to be a snmp master agent, initial the ports */
  if (!agentx_subagent)
    init_master_agent();  /* open the port to listen on (defaults to
udp:161) */

  /* In case we recevie a request to stop (kill -TERM or kill -INT) */
  keep_running = 1;
  signal(SIGTERM, stop_server);
  signal(SIGINT, stop_server);

  snmp_log(LOG_INFO,"example-demon is up and running.\n");

  /* your main loop here... */
  while(keep_running) {
    /* if you use select(), see snmp_select_info() in snmp_api(3) */
    /*     --- OR ---  */
    agent_check_and_process(1); /* 0 == don't block */
  }

  /* at shutdown time */
  snmp_shutdown("example-demon");
  SOCK_CLEANUP;

  return 0;
}
```

ANEXO V

5. Arquitectura del Ambiente de Desarrollo

5.1. Introducción

Dentro de este documento se presentará la arquitectura sobre la cual se producirá el desarrollo del proyecto así como las diferentes pruebas a realizar. Se presentará la topología de la red, el software de base de cada equipo y las diferentes configuraciones realizadas sobre estos.

El objetivo de este documento es presentar una guía práctica de cómo montar el ambiente de desarrollo, con el fin de aportar a la reproducción y repetición de las pruebas a realizar.

5.2. Topología de la red

La red esta compuesta por 3 equipos: el equipo A, el cual como se puede observar en la siguiente figura, tiene una única interfaz de red, configurada con la IP de 10.10.1.80, luego se encuentra el equipo B, el cual contiene 2 interfaces de red, con IPs 10.10.1.1 y 10.10.2.1 respectivamente, y por ultimo se encuentra el equipo C con una única interfaz de red configurada bajo la IP de 10.10.2.80.

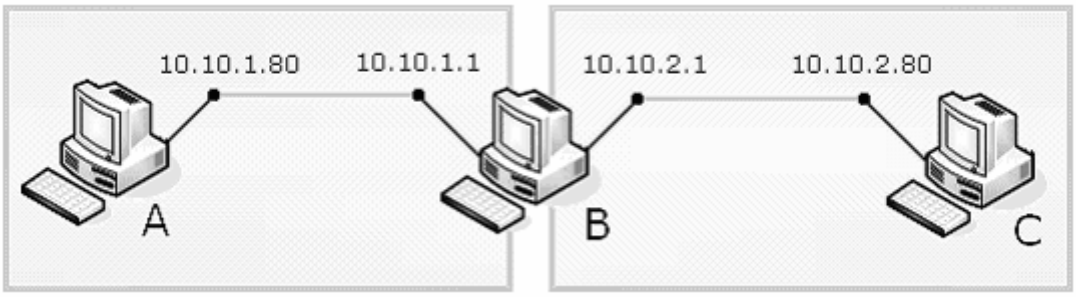


Figura 113: Topología de la Red

El equipo A se encuentra conectado con la interfaz 10.10.1.1 del equipo B, formando un LAN de tipo C (10.10.1.X), mientras que el equipo C esta conectado con el equipo B mediante la interfaz 10.10.2.1 formando por su parte otra LAN de tipo C (10.10.2.X).

La topología de la red fue diseñada de esta manera con el fin de poder emular un router con el equipo B, de esta manera la única función provista por el equipo B será la de rutear los paquetes de una LAN a la otra, y es en este equipo donde se configurará la perdida de paquetes, la cual se detallará mas adelante.

De ahora en más (y como se puede observar en la Figura 114), al equipo A lo llamaremos “Servidor” (o Server) que será quien oficie de servidor de streaming, al equipo B la llamaremos “Router” y al equipo C lo llamaremos “Cliente”.

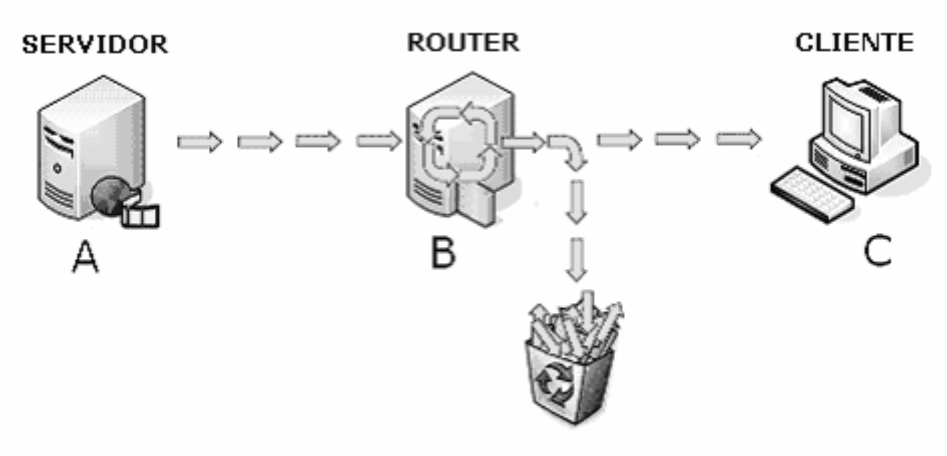


Figura 114: Conceptualización de la arquitectura

5.3. Software de base

Tanto en el Servidor como en el Cliente, se instaló el sistema operativo Ubuntu versión 5.04, en el router, se instaló el Debian GNU/Linux 3.1 r2 "Sarge".

En el Router y para simular las diferentes distorsiones sufridas en una red de amplio alcance, se instaló el modulo provisto por el kernel de linux, llamado NetEm. Finalmente, para controlar dicha aplicación, se instaló la librería iproute2, la cual también fue instalada tanto en el Cliente como en el Servidor, con el fin de facilitar los comandos sobre el ruteo de paquetes.

En el Cliente y el Servidor se instaló la librería iproute2 versión 20041019-1 mediante el comando:

```
aptitude install iproute
```

En el Router se instaló la versión 20041019-3 mediante el comando:

```
apt-get install iproute
```

5.4. Montaje de la red

Primero se deben configurar las IP de cada una de las interfaces de red de cada uno de los equipos. Esto se puede realizar mediante el comando:

```
ifconfig eth0 "IP_EN_LA_RED" netmask "MASCARA_RED" up
```

en donde "IP_EN_LA_RED" es la IP la cual se le quiere asignar a la interfaz eth0 (en el caso del Servidor, esta sería 10.10.0.80) y "MASCARA_RED" es la máscara de red a usar en la LAN, en nuestro caso es siempre es 255.255.255.0

5.4.1. Configuraciones en el Servidor y el Cliente

Una vez que tenemos las IPs de cada interfaz de red bien configuradas debemos setear el default gateway (tanto del Servidor como del Cliente) para que estos apunten al router. Para esto se debe ejecutar los siguientes comandos:

En el caso del Servidor:

```
up route add -net 10.10.2.0 netmask 255.255.255.0 gateway 10.10.1.1
```

En el caso del Cliente:

```
up route add -net 10.10.1.0 netmask 255.255.255.0 gateway 10.10.2.1
```

5.4.2. Configuraciones en el Router

En el caso del Router es necesario crear tablas de routeo para que los paquetes provenientes del Servidor con destino al Cliente, es decir que para los paquetes con origen 10.10.1.80 y destino 10.10.2.80, los redirija correctamente, y viceversa, que los paquetes provenientes del Cliente con destino al Servidor también sean correctamente ruteados.

Para esto se debe ejecutar los siguientes comandos:

- **Crear tablas de ruteo**

Aquí se crea la tabla de ruteo audit_table_server, en donde se encontraran las reglas a ser ejecutadas sobre los paquetes provenientes del Servidor:

```
echo 200 audit_table_server >> /etc/iproute2/rt_tables
```

De esta misma manera se crea la tabla para los paquetes provenientes del Cliente:

```
echo 201 audit_table_client >> /etc/iproute2/rt_tables
```

Se debe asegurar antes de crear estas tablas que los IDs asociados a las mismas (200 y 201), no estén definidos dentro del archivo /etc/iproute2/rt_tables, si estas ya se encuentran definidos, simplemente se debe elegir otros.

- **Asociar paquetes a dichas tablas de ruteo**

Una vez que las tablas se encuentran creadas se les debe asociar paquetes a las mismas (solo estos paquetes se ejecutarán las reglas de ruteo que contengan cada una de las tablas), para esto se debe ejecutar los siguientes comandos:

```
up ip rule add from 10.10.1.80 table audit_table_server
```

Como se comento anteriormente, los paquetes provenientes del Servidor (cuya IP de origen es la 10.10.1.80), serán ruteados bajo la tabla audit_table_server.
Lo mismo ocurre con los paquetes de origen el Cliente, son ruteados con la tabla audit_table_client:

```
up ip rule add from 10.10.1.80 table audit_table_client
```

- **Agregar reglas a las tablas de ruteo**

Finalmente debemos agregar las reglan a cada una de estas tablas, es decir, que los paquetes ruteados con la tabla audit_table_server, sean enviados por la interfaz de red eth1 (cuya IP es la 10.10.2.1):

```
up ip route add default via 10.10.2.1 dev eth1 table \
audit_table_server
```

Y lo mismo para la tabla audit_table_client:

```
up ip route add default via 10.10.1.1 dev eth0 table \
audit_table_client
```

Estas configuraciones se deben ejecutar cada vez que las interfaces de red sean iniciadas, para esto, se puede ejecutar estos comandos en forma manual o agregar estas líneas en el archivo: /etc/network/interfaces.

Un comando que puede llegar a ser muy útil en casos en donde las tablas de ruteo sean modificadas es el siguiente:

```
up ip route flush cache
```

El cual limpia el cache acerca de las tablas de ruteo.

- **Habilitar el Forwarding**

Por último y para que todas estas configuraciones sean llevadas a la práctica (es decir que el sistema operativo haga uso de las misma) es necesario habilitar la capacidad de forwarding (reenvío) de paquetes TCP/IP en el Router.

Para realizar esto, tendremos que asignarle el valor 1 a la variable de sistema net.ipv4.ip_forward, añadiendo (o modificando, si ya existe) una línea como que se presenta a continuación en el fichero /etc/sysctl.conf:

```
net.ipv4.ip_forward = 1
```

En caso de querer profundizar en el uso de las diferentes funcionalidades provistas por la librería iproute2, referirse a [Ref20], [Ref11] y [Ref14].

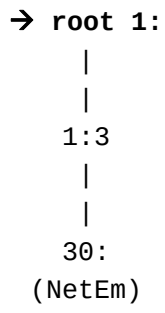
5.5. Generación de distorsiones en la red

Como se vio anteriormente unos de los fines de esta arquitectura de red es la de poder simular pérdidas de paquetes, delays en la red y otros tipos de errores de común ocurrencia en redes como Internet.

Los comandos utilizados para setear la simulación de dichos errores son los siguientes:

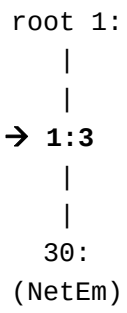
```
tc qdisc add dev eth0 root handle 1: prio
```

Mediante este comando creamos el nodo de inicio de nuestro filtrado, el cual se identifica 1: (o 1:0), esto implica que todo los paquetes que pasen por la interfaz eth0 pasarán también por ese nodo.



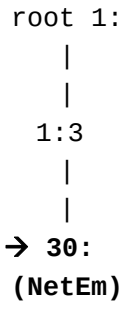
A continuación se agrega un filtrado, para que ciertos paquetes sean manejados con un cierto handler:

```
tc filter add dev eth0 protocol ip parent 1:0 prio 3 u32 match ip \
sport 1234 0xffff flowid 10:3
```



El criterio de filtrado establecido en nuestro caso, fue el de filtrar todos los paquetes cuyo puerto de origen sea el 1234, el cual será utilizado por el Servidor para realizar el Streaming

Una vez establecido el filtro a utilizar, vamos a asignarle un handler a eso filtro para que se encargue del manejo de los paquetes filtrados, en nuestro caso, el mismo será NetEm. A continuación se presentan algunos ejemplos de diferentes maneras de configurar el NetEm:




```
tc qdisc add dev eth0 parent 1:3 handle 30: netem loss 1% \
distribution normal
```

En este caso el 1 % de los paquetes recibidos será descartados, bajo una distribución normal.

```
tc qdisc add dev eth0 parent 1:3 handle 30: netem corrupt 1%
```

Aquí el 1% de los paquetes se les inyectará una corrupción.

```
tc qdisc add dev eth0 parent 1:3 handle 30: netem gap 5 delay 10ms
```

Finalmente en este caso cada 5 paquetes recibidos, se atrasará 10 milisegundos.

Por más información acerca de la configuración del NetEm referirse a [Ref23], [Ref26]. En el caso de querer profundizar en el armado de los diferentes filtros referirse a [Ref21], [Ref22] y [Ref23].

ANEXO VI

6. Manual de usuario del vlc-audit

6.1. Introducción

En este anexo explicamos como usar las funcionalidades del vlc-audit.

El vlc puede ser ejecutado con 3 interfaces distintas:

1. LoggerAudit – Se logguea a archivo los diferentes eventos captados como: comienzo de un stream, llegada de un frame, etc.
2. SNMP – Se exporta datos mediante SNMP
3. QoS – Se realiza el loggueo a archivo y la exportación mediante SNMP

El vlc tiene que ser ejecutado con los parámetros adecuados (en el servidor y el cliente) para que el sistema pueda funcionar correctamente.

6.2. Servidor

El servidor de streaming debe ejecutarse únicamente con la interfaz LoggerAudit. Esto es debido a que solo los clientes exportan datos vía SNMP y tanto la interfaz SNMP como QoS, instancian el módulo de SNMP.

Cuando se stremea un video, se debe especificar:

- Protocolo
- Método de encapsulamiento(muxer)
- Host
- Puerto
- Archivo

Esta información puede ser ingresada:

6.2.1. Como parámetro al ejecutar el vlc

El comando tiene la siguiente sintaxis:

```
> vlc --extraint loggerAudit --sout
"#duplicate{dst=std{access=<protocol>,mux=<muxer>,dst=<host>:<port>}}"  <file>
```

6.2.2. Mediante la interfaz gráfica

Primero, tenemos que ejecutar el vlc con el siguiente comando:

> vlc –-extraint loggerAudit

Luego en el menú, se debe seleccionar la opción *Archivo -> Abrir Archivo*
A continuación se abrirá la ventana que podemos ver en la figura siguiente.

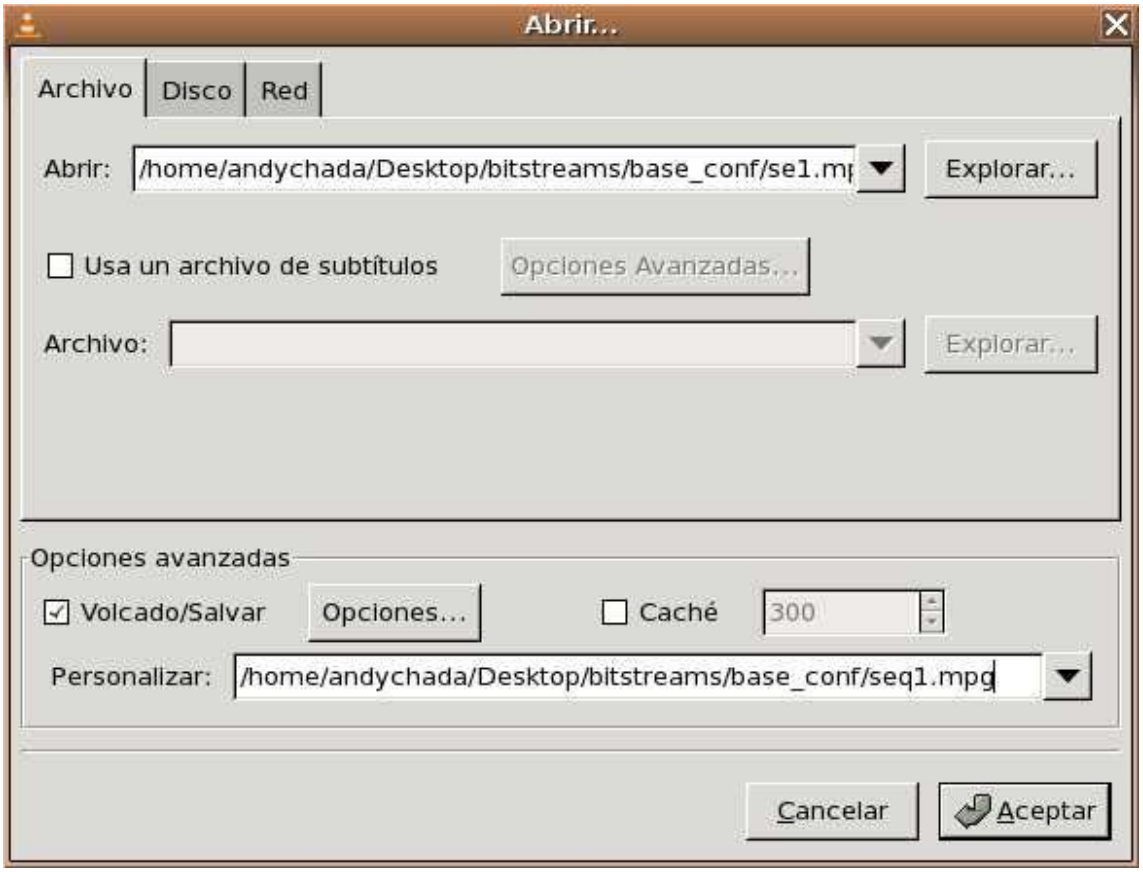


Figura 115: Manual de Usuario VLC - Audit - Selección de archivo

Haciendo clic en el botón *Explorar* se elige el video. Luego se hace clic sobre el check box Volcado de Salida y se presiona el botón Opciones.
A continuación se abrirá la ventana que podemos ver en la figura siguiente.

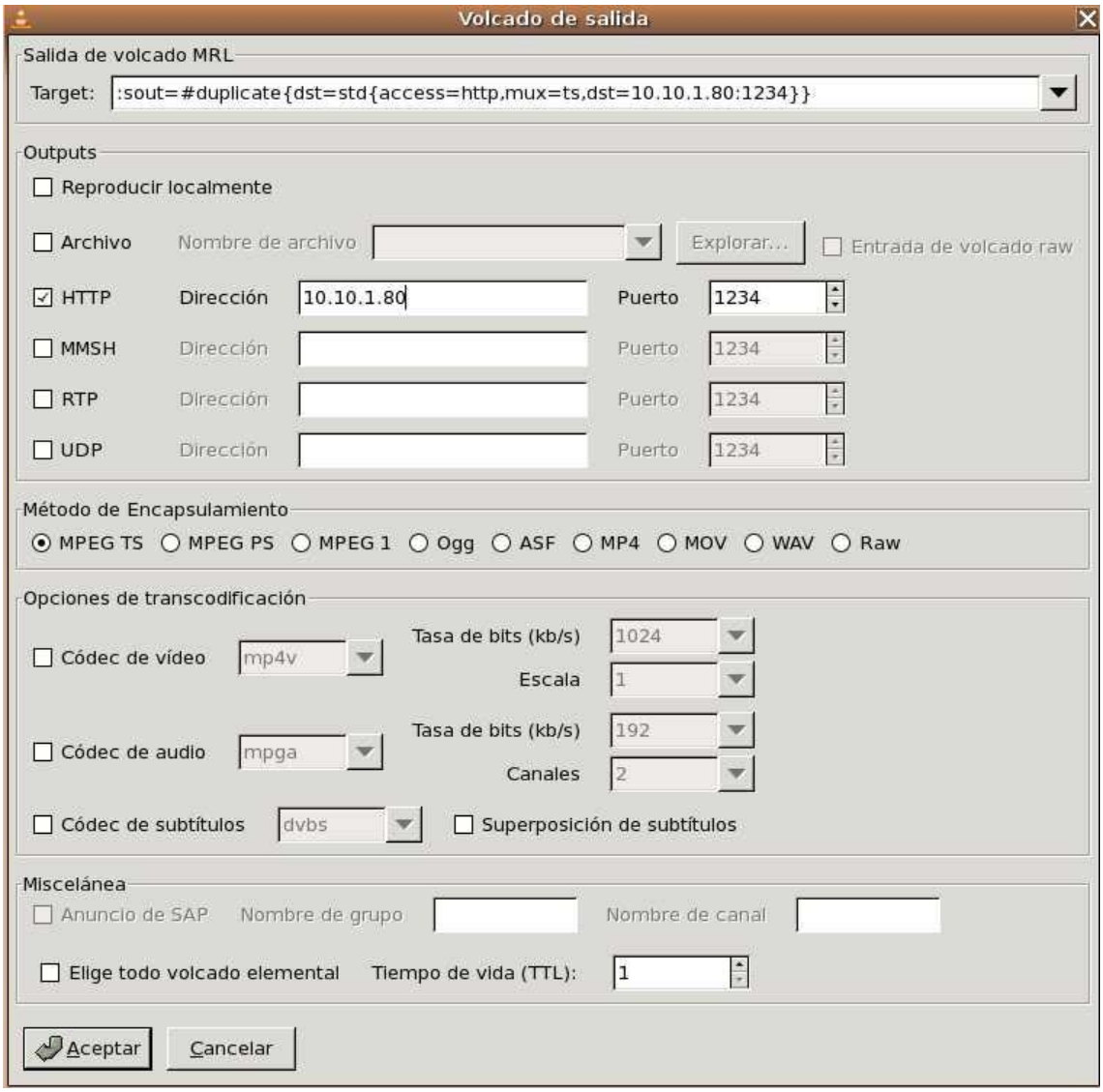


Figura 116: Manual de Usuario VLC - Audit - Opciones

En esta ventana podremos ingresar el host, puerto, muxer y protocolo.

6.3. Cliente

A diferencia del servidor, el cliente puede ser ejecutado con cualquiera de las 3 interfaces.

Cuando se reproduce un video se debe:

- Especificar la URL
- Usar el módulo de stream out *qos-duplicate*

Al igual que en el servidor, la información puede ser ingresada:

6.3.1. Como parámetro al ejecutar el vlc

El comando tiene la siguiente sintaxis:

> vlc –-extraint <interfaz> --sout "#qos-duplicate{dst=display}" <url>

6.3.2. Mediante la interfaz gráfica

Primero, tenemos que ejecutar el vlc con el siguiente comando:

> vlc –-extraint <interfaz>

Luego en el menú, se debe seleccionar la opción *Archivo -> Abrir volcado de Red* y se abrirá la ventana que podemos ver en la a continuación.

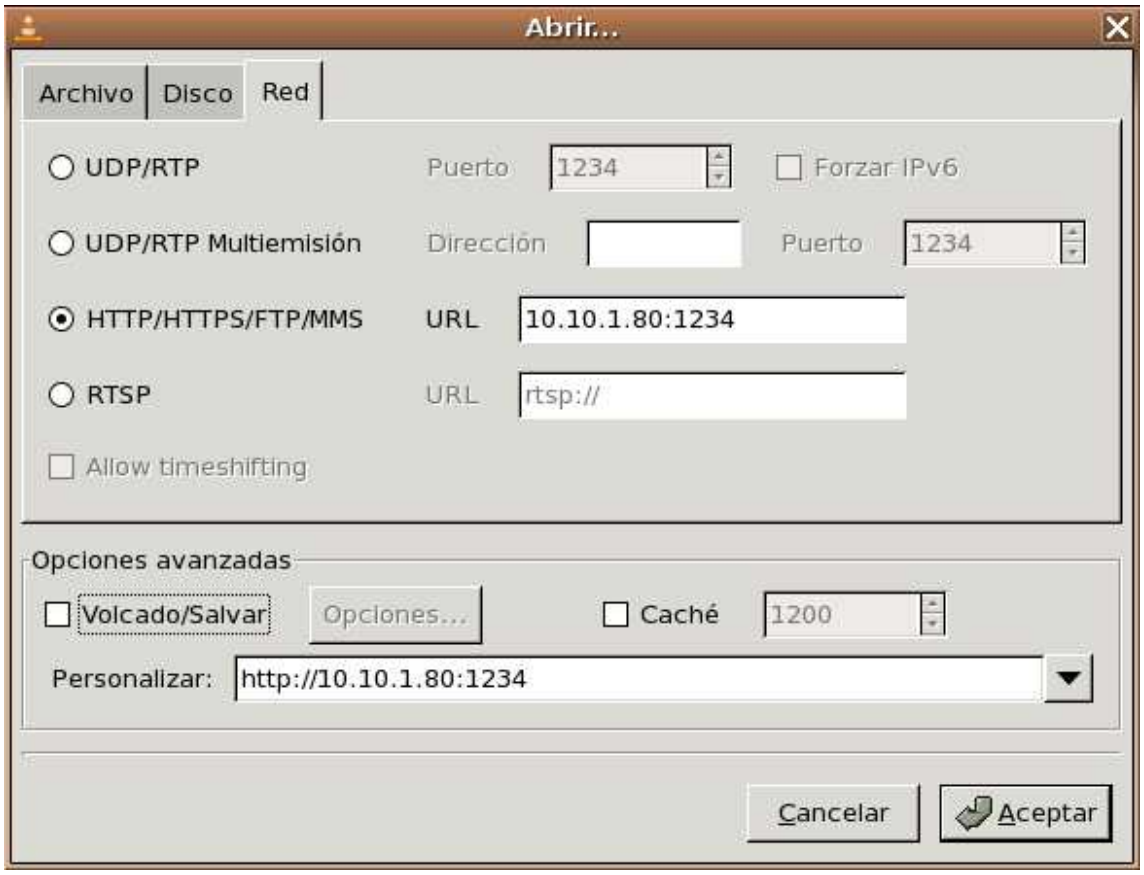


Figura 117: Manual de Usuario VLC - Audit - Volcado de Red

Tenemos que ingresar la URL. Luego se tiene que hacer clic en el check box *Volcado de salida* y hacer clic en *Opciones*.

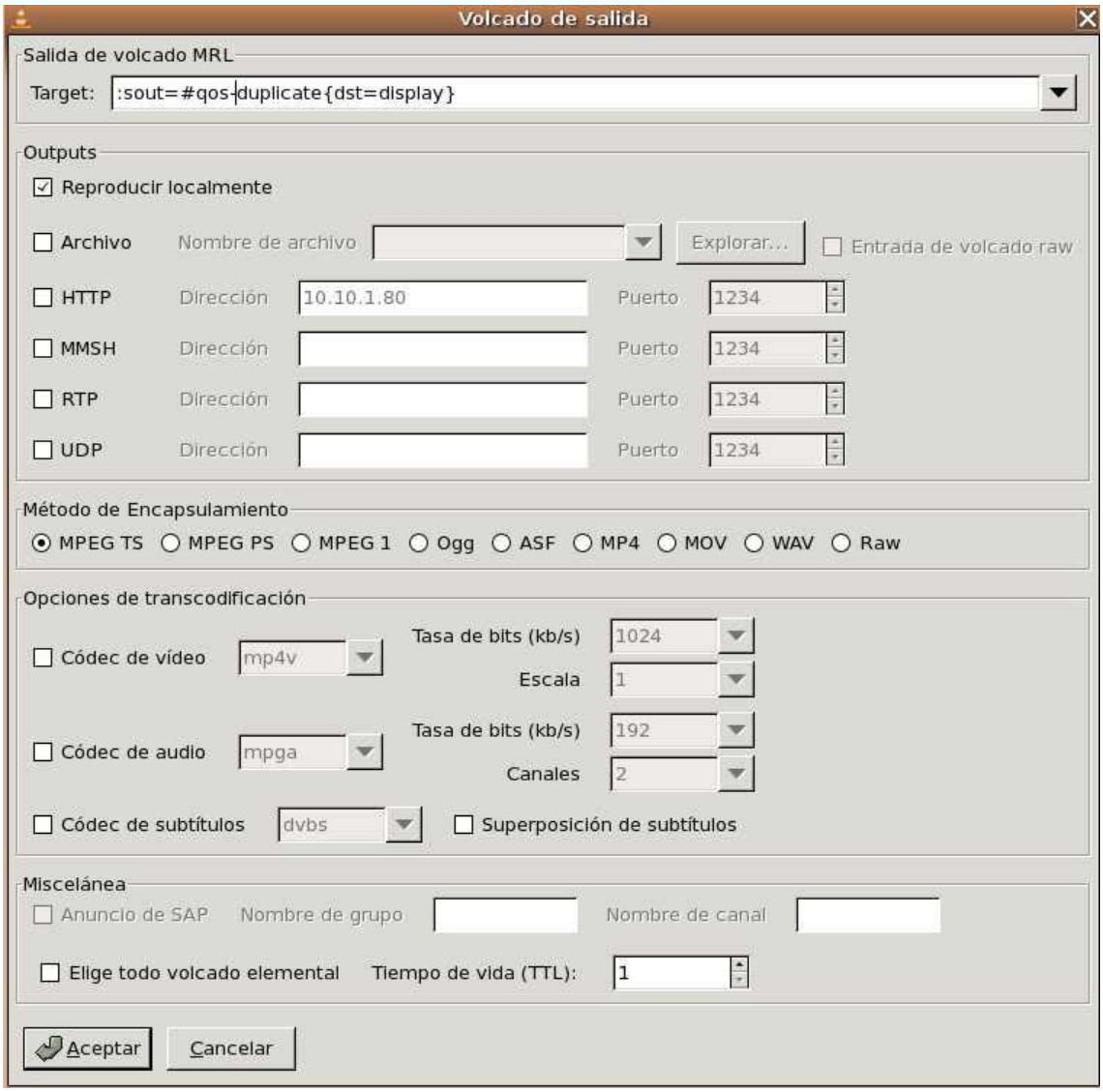


Figura 118: Manual de Usuario VLC - Audit - Reproducción Módulo de stream out QoS

Se elige reproducir localmente y el módulo de stream out qos-duplicate.

6.4. Interfaz telnet

En las partes anteriores, explicamos como ejecutar el servidor y el cliente. De la forma antes descrita, desde el servidor solo se puede strimear un único video, y desde el cliente se puede ver también un único video.

Hay casos en los que se necesita stremear o ver más de un video.

Si queremos reproducir varios videos, podríamos ejecutar varias instancias del vlc, pero si queremos usar la misma instancia, ejecutamos el vlc con interfaz telnet.

La sintaxis del comando es la siguiente:

```
> vlc –I telnet ––telnet-port <port> ––extraint <interfaz>
```

De esta forma queda un puerto abierto, para que nos podamos comunicar vía telnet. Luego ejecutamos el comando telnet, ingresando el puerto y el host.

```
> telnet <host> <port>
```

Luego se debe ingresar la contraseña que por defecto es admin.

A partir de ahí, tenemos un prompt para ejecutar los comandos. Por comodidad, guardamos secuencias de comandos en un archivo de configuración.

Para poder ejecutar un archivo de configuración, se debe ejecutar:

```
> load NombreArchivo.conf
```

En dicho archivo debemos incluir los comandos que serán ejecutados secuencialmente.

En el archivo, vamos a poder incluir comandos para stremear un video y para reproducir un video.

- Para stremear un video, debemos poner en el archivo de configuración algo como lo siguiente:

```
new channel1 broadcast enabled
setup channel1 input <filename>
setup channel1 output
#duplicate{dst=std{access=http,mux=<muxer>,dst=<host>:<port>}}
control channel1 play
```

- En el caso de reproducción de un video

```
new channel2 broadcast enabled
setup channel2 input  <url>
setup channel2 output #qos-duplicate{dst=display}
control channel2 play
```

Notar que en un archivo de configuración, podemos stremear y reproducir videos al mismo tiempo, y en el número deseado.

6.5. Ejemplo

Para ejemplificar lo que explicamos anteriormente, veamos un ejemplo.

Tenemos el servidor de streaming. El mismo va a stremear 2 videos (video1.mp4 y video2.mp4). En el servidor vamos a ejecutar el vlc con la interfaz telnet.

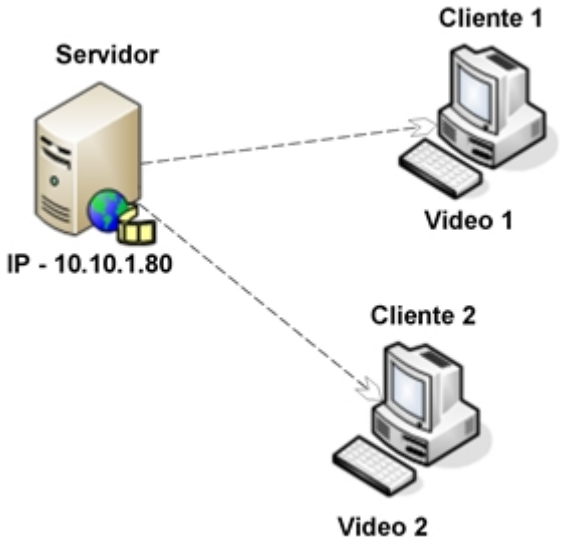


Figura 119: Manual de Usuario VLC - Audit - Ejemplo

Luego el Cliente 1 va a reproducir video1.mp4 y el Cliente 2 que va a reproducir video2.mp4.

Primero tenemos que ejecutar el vlc en el servidor con la interfaz telnet.

```
> vlc –I telnet ––telnet-port 4212 ––extraint loggerAudit
```

Luego creamos un archive de configuración Server.conf con los siguientes comandos:

```
new channel1 broadcast enabled
setup channel1 input /home/audit/video1.mp4
setup channel1 output
#duplicate{dst=std{access=http,mux=ts,dst=10.10.1.80:1234}}
control channel1 play

new channel2broadcast enabled
setup channel2 input /home/audit/video2.mp4
setup channel2 output
#duplicate{dst=std{access=http,mux=ts,dst=10.10.1.80:1235}}
control channel2 play
```

Luego ejecutamos el telnet:

> telnet 10.10.1.80 4212

Finalmente cargamos el archivo de configuración el el telnet:

```
>load Server.conf
```

Luego, en el cliente 1, en la que deseamos exportar datos a SNMP y loguear a archivo, ejecutamos:

```
> vlc –-extraint qos --sout "#qos-duplicate{dst=display}" http://
10.10.1.80:1234
```

Finalmente, en el cliente 2, en la que deseamos solo exportar datos a SNMP, ejecutamos:

```
> vlc –-extraint snmp --sout "#qos-duplicate{dst=display}" http://
10.10.1.80:1235
```

ANEXO VII

7. AUDIT Project Stats Viewer

7.1. Introducción

El AUDIT Project Stats Viewer es una aplicación desarrollada con el fin de recabar y presentar la información reportada por los clientes VLC-AUDIT. En este anexo se presentan las diferentes funcionalidades brindadas por esta aplicación y una guía para poder operarla.

7.2. Tecnologías aplicadas

Esta aplicación fue desarrollada en PHP, como motor de base de datos se usó MySQL, como servidor Web Apache, se usó la librería Net-SNMP (migrada a PHP) para las consultas mediante SNMP y la librería RRDTools (migrada a PHP) para generar los gráficos.

7.3. Arquitectura de la aplicación

Para comenzar veremos un breve reseña de la arquitectura de la aplicación con el fin de comprender de mejor manera su funcionamiento.

Esta aplicación se puede separar inicialmente en 2 grandes módulos diferentes, uno referido a la obtención de los datos y otro referido a la presentación de los mismos.

7.3.1. Obtención de los datos

Como se puede observar en la Figura 120, para cada cliente VLC-AUDIT configurado en el sistema, existe un proceso (P1, P2, … Pn) el cual realiza polling mediante SNMP recabando así la información reportada por cada uno de los clientes multimedia.

A su vez para cada cliente, existe una RRD (***R**ound **R**obin **D**atabase*) en la cual cada uno de los procesos recolectores de información (P1, P2, … Pn) ingresaran la información específica de cada cliente.

También la aplicación trabaja con una base de datos relacional, en donde se ingresa la información general de cada cliente/streaming, como por ejemplo IP, Puerto, Protocolo de streaming, Codecs, etc. Por lo que cada proceso (P1, P2, … Pn) también ingresará esta información en esta base de datos.

Finalmente también existe un proceso global (PG) el cual se encarga de procesar la información ingresada en la base de datos relacional (en particular, de procesar la cantidad de clientes online y de streams activos por cliente) ingresando esta información procesada en su propio RRD (RRDG).

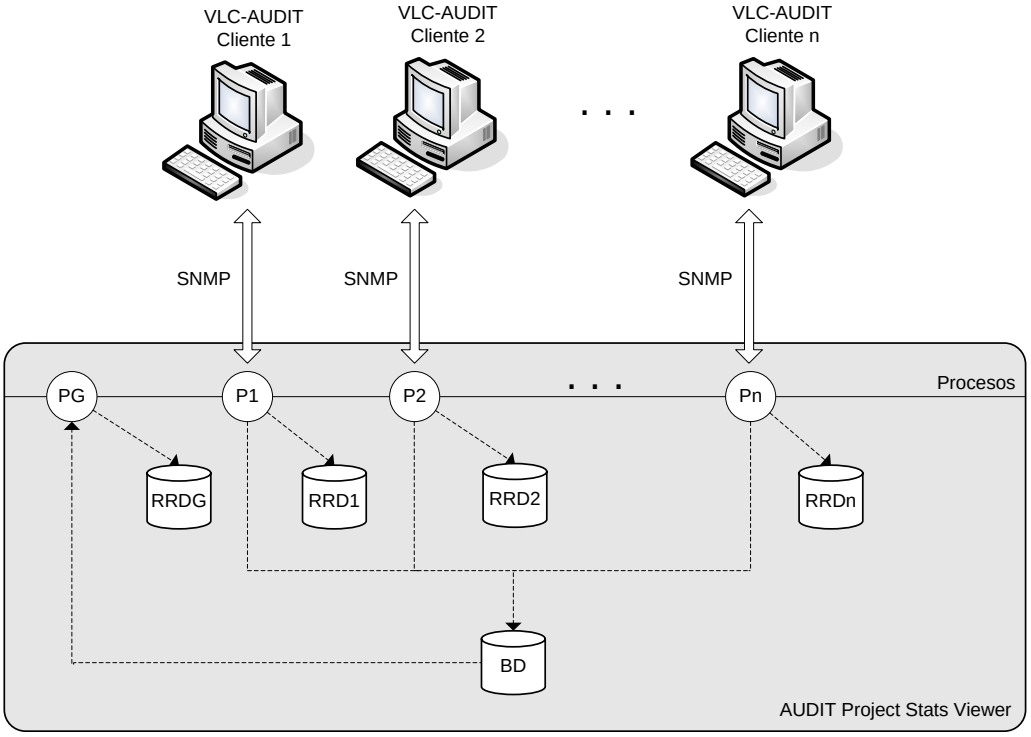


Figura 120: AUDIT Project Stats Viewer – Obtención de datos

7.3.2. Presentación de los datos

La presentación de los datos es generada a demanda (es decir cada vez que un browser realiza un request se genera una nueva grafica) y estas graficas son generadas en base a la información almacenada en los RRDs (observar en la Figura 121).

Existen varios tipos de gráficos y estos se pueden categorizar en 2 grandes grupos, los gráficos a nivel de Streams (en donde se presenta la información específica de un Stream) y los gráficos en los que presentan información consolidada de varios Streams.

Los gráficos a nivel de Streams son generados en base al RRD del cliente en cuestión, mientras que para la generación de gráficos referentes a varios streamings se usan el conjunto de RRDs de los clientes involucrados.

Finalmente existe un par de gráficos (stream activos y clientes activos) los cuales son generados en base al RRD global.

A continuación profundizaremos en cada uno de estos gráficos, describiendo que es lo que estos representan.

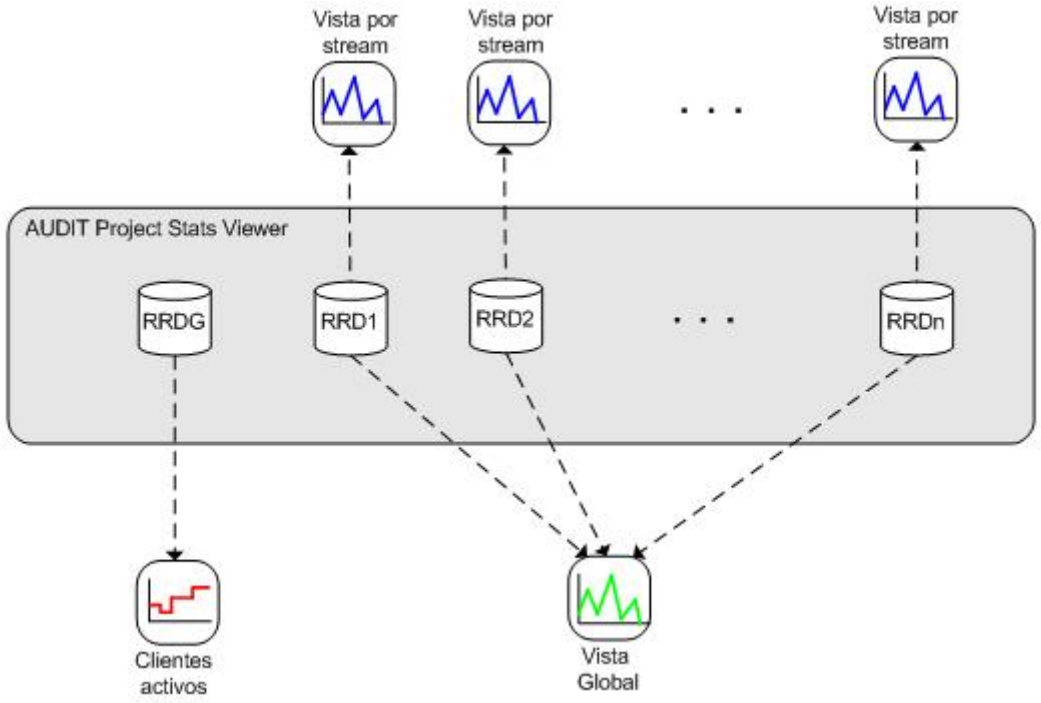


Figura 121: AUDIT Project Stats Viewer – Presentación de datos

7.4. Tipos de gráficos

Como se comentó anteriormente los diferentes tipos de gráficos serán separados en 2 grupos, los gráficos a nivel de stream y los gráficos múltiple streams.

7.4.1. Gráficos a Nivel de Stream

7.4.1.1. Perdidas de Frames

Este gráfico representa la pérdidas de Frames en el tiempo, existe un gráfico para cada tipo de Frame (I, P, B) y uno global (el cual toma en cuenta perdidas de cualquier tipo de Frame.

En la Figura 122 se presentan un ejemplo de pérdidas referidas a Frames I, en la Figura 123 de Frames P, en la Figura 124 de Frame B y en la Figura 125 pérdidas globales.

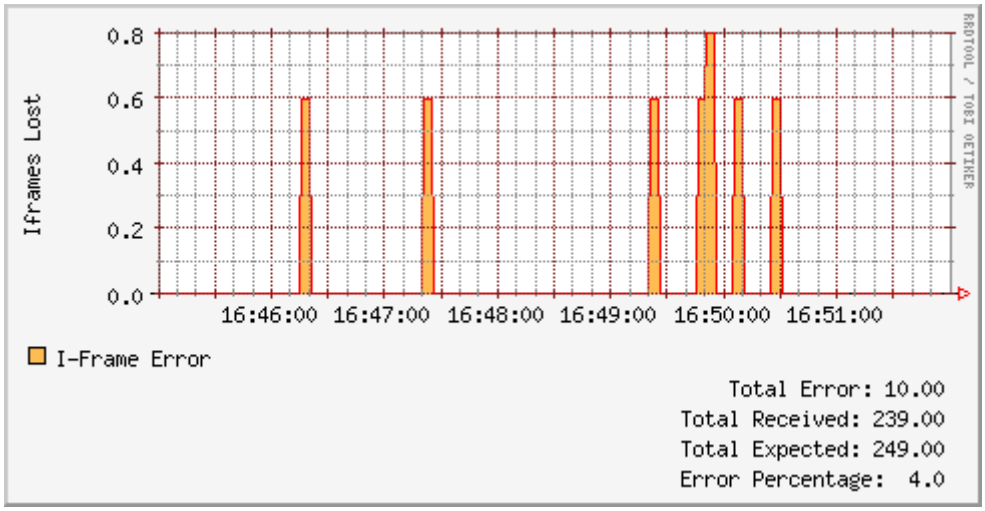


Figura 122: Perdidas de Frames I

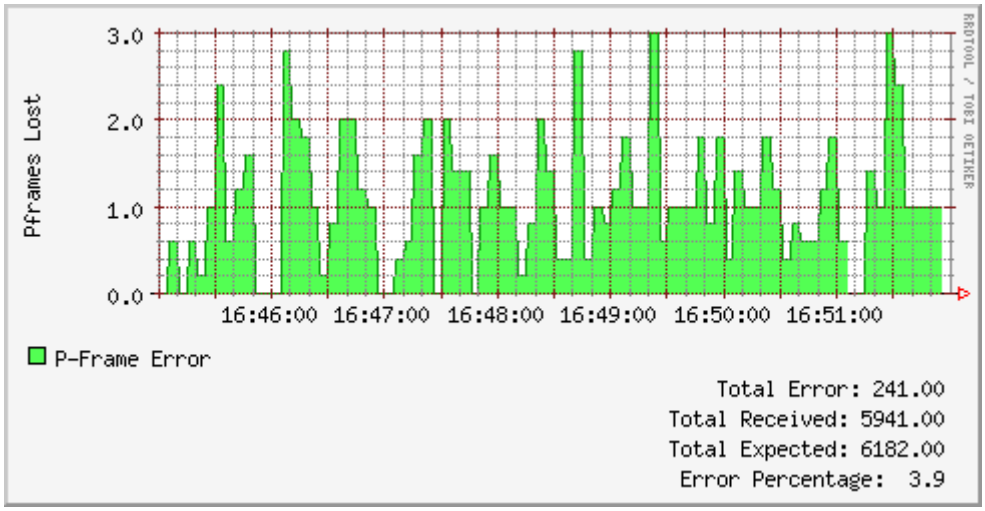


Figura 123: Perdidas de Frames P

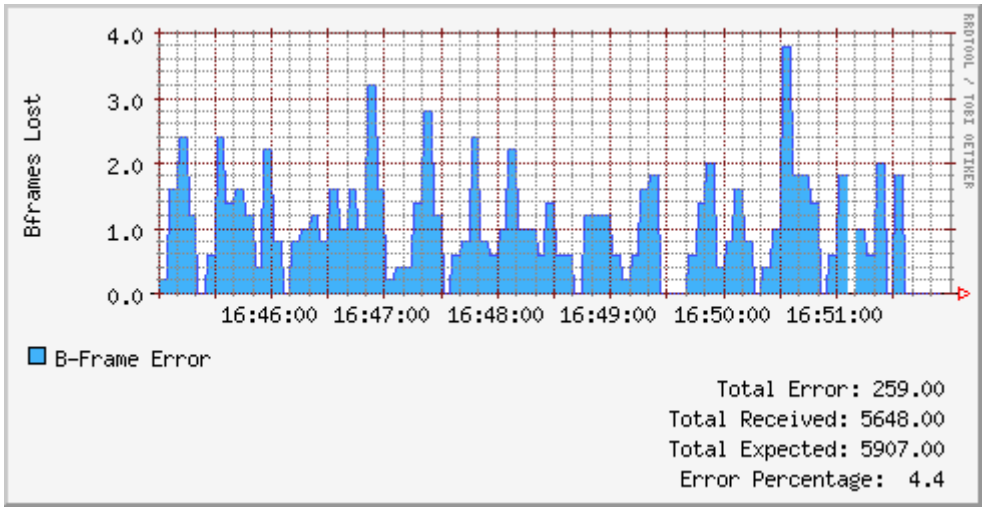


Figura 124: Perdidas de Frames B

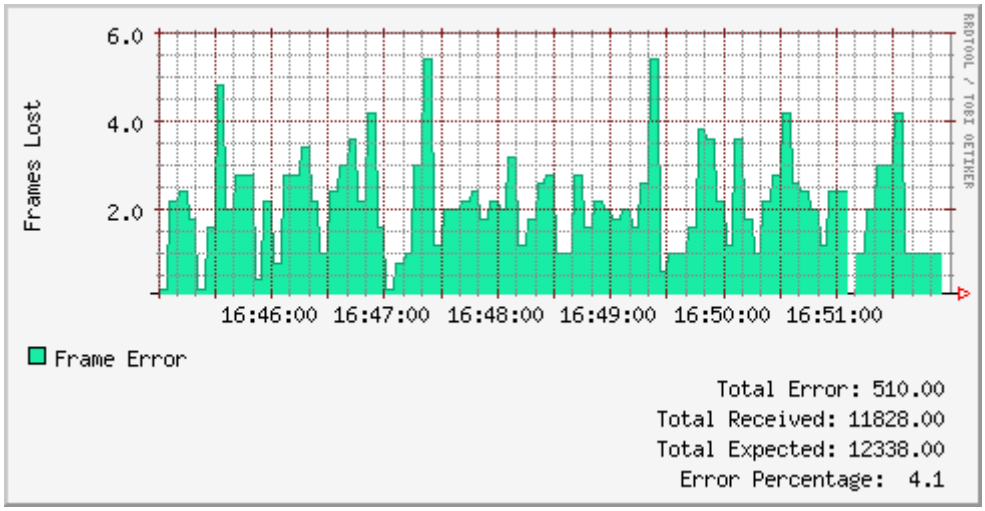


Figura 125: Perdidas de Frames Global

7.4.1.2. Recibidos VS Enviados

En este gráfico se presentan la cantidad de Frames enviados por el servidor y la cantidad de Frames recibidas por el cliente multimedia (de esta se puede deducir las perdidas). Están al igual que en el caso anterior están separadas por tipo de Frame, brindando también una grafica global.

En la Figura 126 se presenta un ejemplo de este gráfico referido a Frames I, en la Figura 127 referente a Frames P, en la Figura 128 referente a Frames B y en la Figura 129 a nivel global.

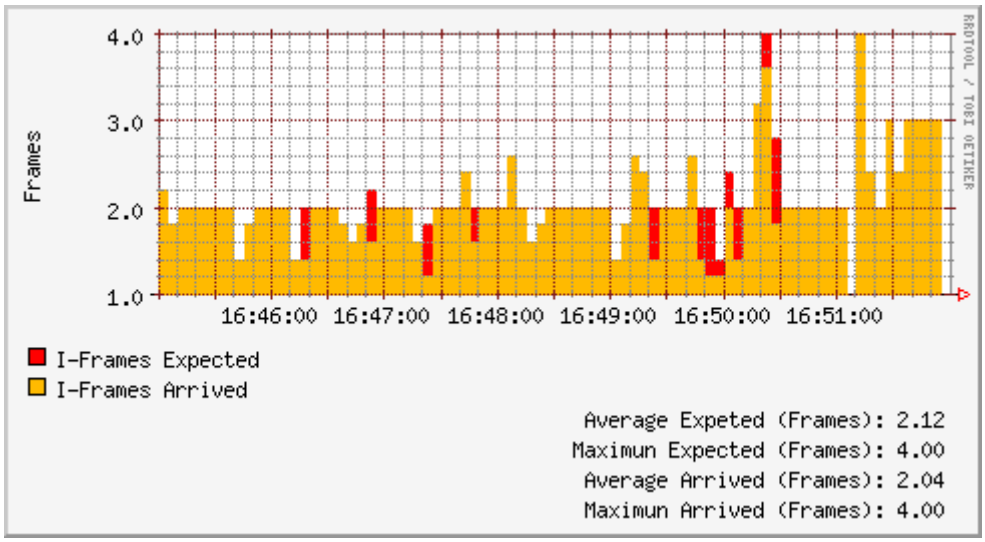


Figura 126: Recibido VS Enviado (Frames I)

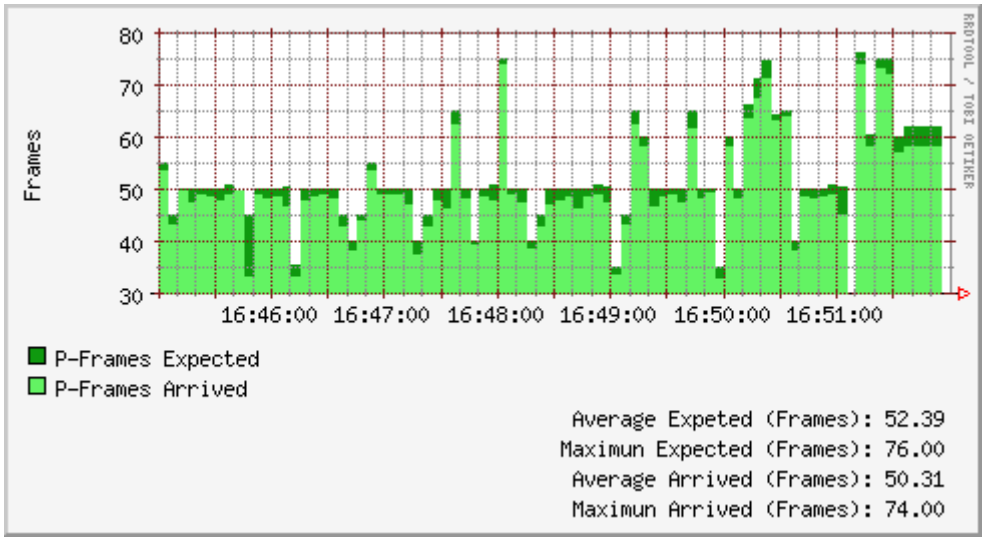


Figura 127: Recibido VS Enviado (Frames P)

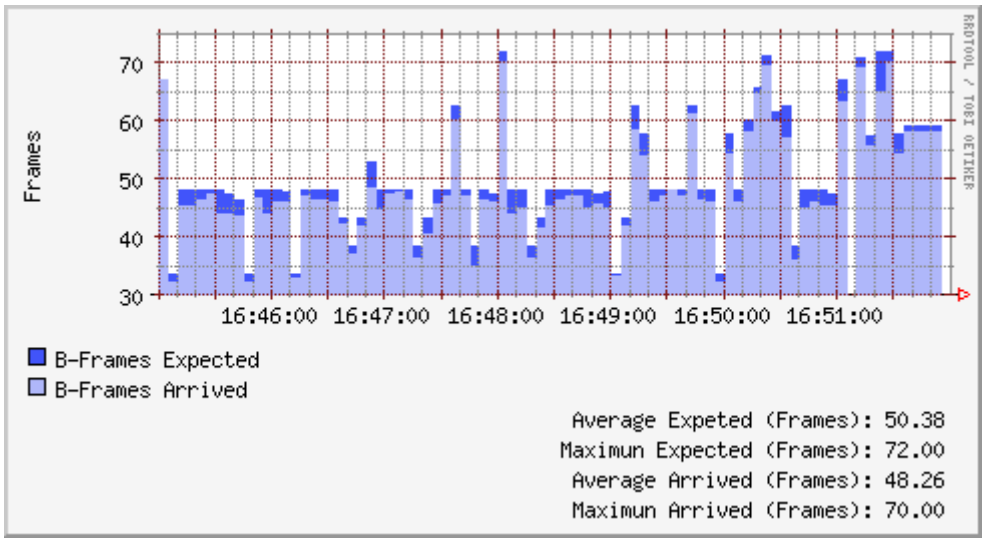


Figura 128: Recibido VS Enviado (Frames B)

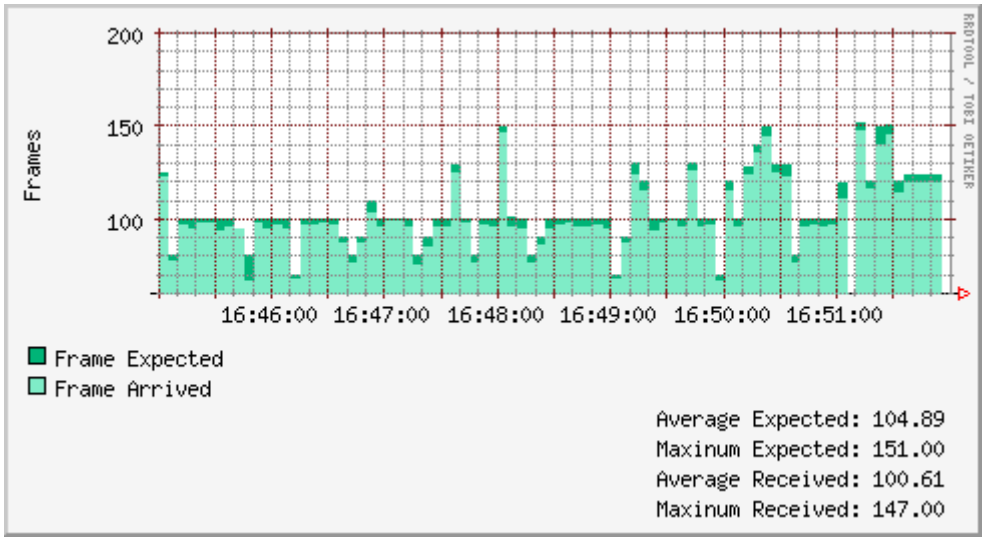


Figura 129: Recibido VS Enviado (Global)

7.4.1.3. Tamaño de Frames

En este gráfico se presenta el tamaño de los diferentes tipos de Frames en el tiempo. A su vez se presenta el tamaño promedio para cada uno de estos tipos.

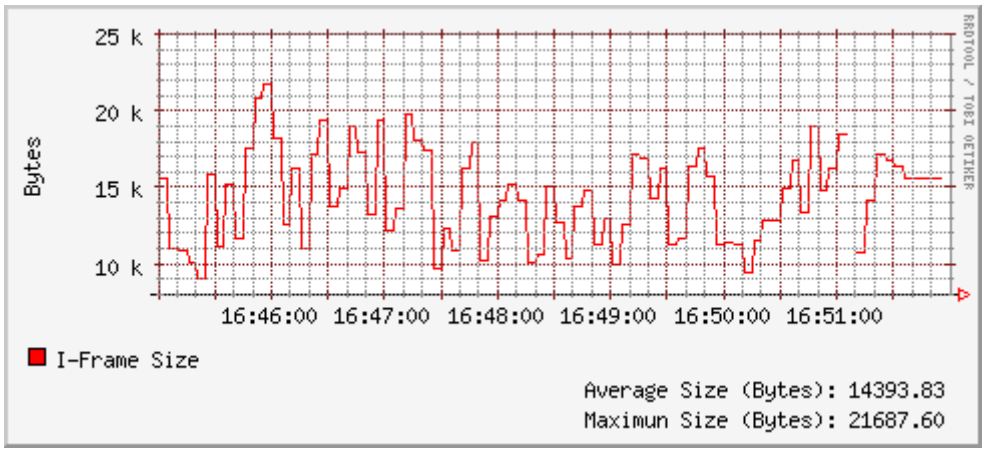


Figura 130: Tamaño de Frames I

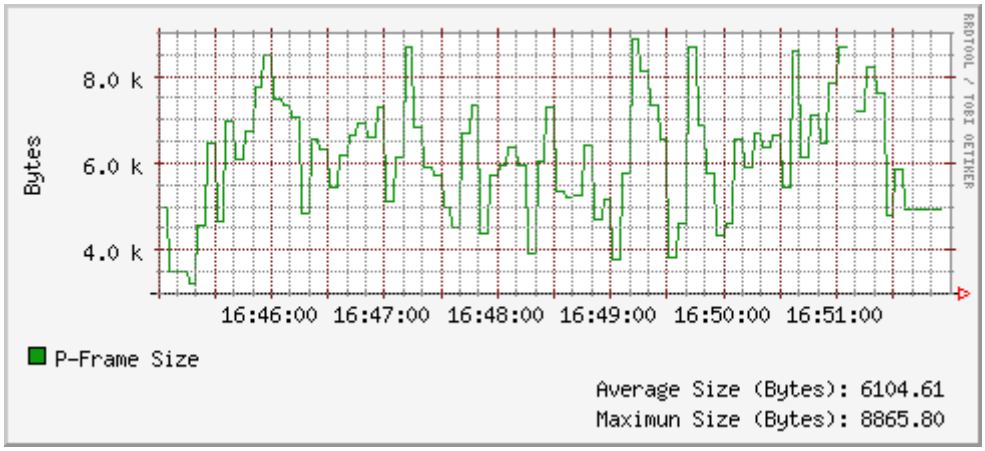


Figura 131: Tamaño de Frames P

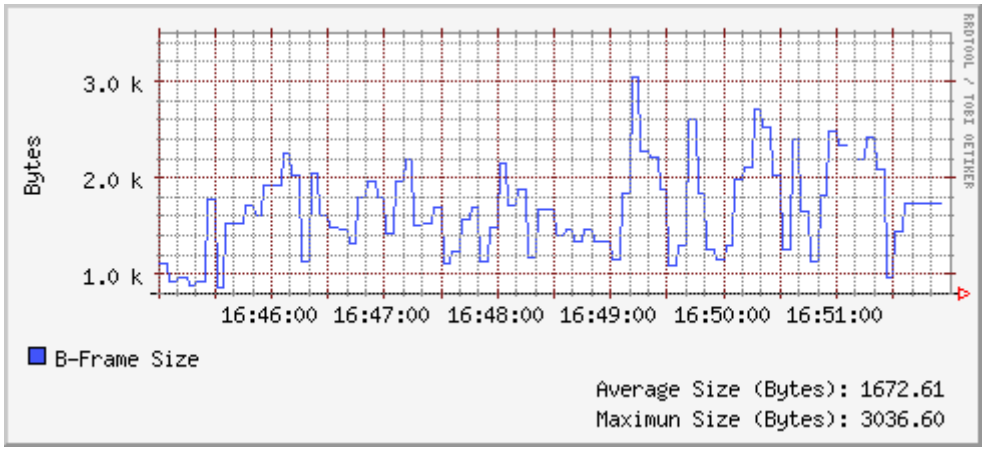


Figura 132: Tamaño de Frames B

En la Figura 130 se presenta un ejemplo de este gráfico para los Frames I, en la Figura 131 se presenta el mismo para los frames P y en la Figura 132 para los Frames B. En este caso no existe un gráfico global dado que el mismo no tiene relevancia.

7.4.1.4. Bitrate Total

En este gráfico se presenta el bitrate total de video (recordar que el mismo es calculado en base al payload de cada uno de los Frames de video).

En la Figura 133 se presenta un ejemplo de este gráfico.

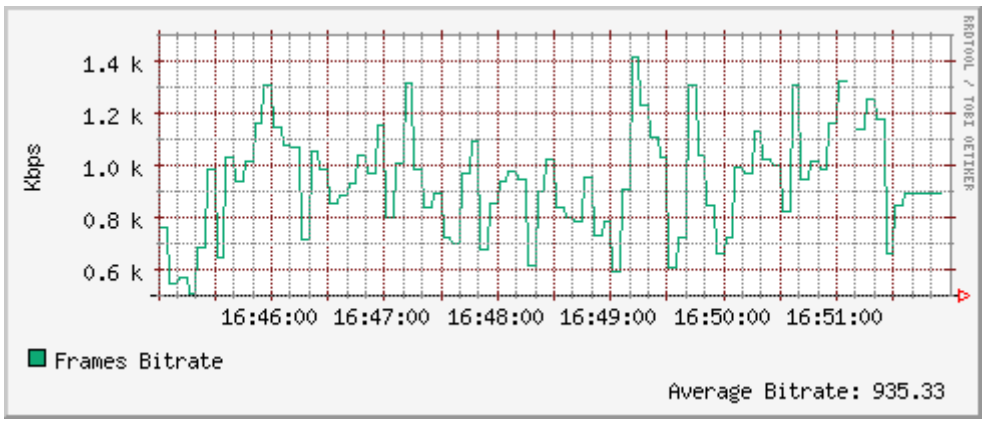


Figura 133: Bitrate Total de video

7.4.2. Gráficos múltiple streams

En este grupo de gráficas se encuentran gráficas a nivel de un cliente (en donde se grafican el consolidado de todos los streams de ese cliente) y gráficas a nivel de cliente en donde se grafican datos consolidados en base a todos los streams de todos los clientes.

7.4.2.1. Cliente activos

Este gráfico representa la cantidad de clientes activos (es decir recibiendo un streaming de algún servidor) en el tiempo.

En la Figura 134 se presenta un ejemplo de esta grafica.

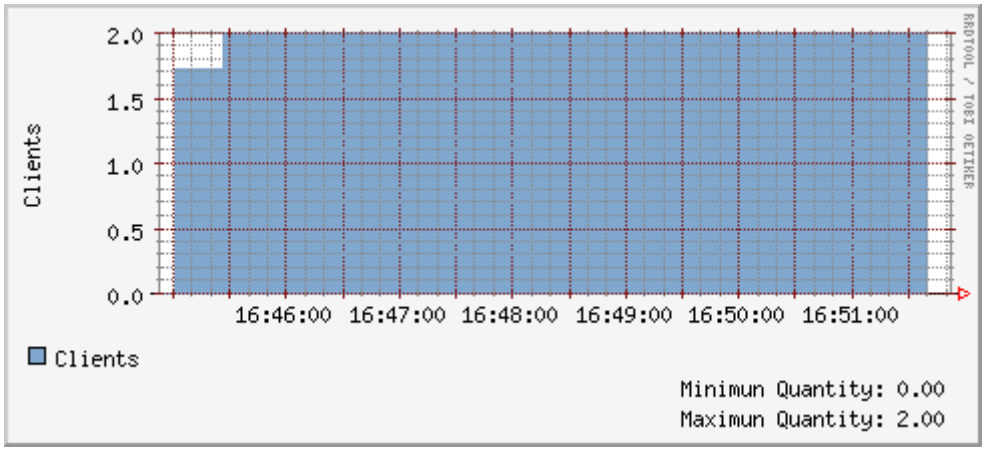


Figura 134: Clientes activos

7.4.2.2. Streams activos

En este gráfico se presenta la cantidad de streams activos en el tiempo. En la Figura 135 se presenta un ejemplo de este gráfico.

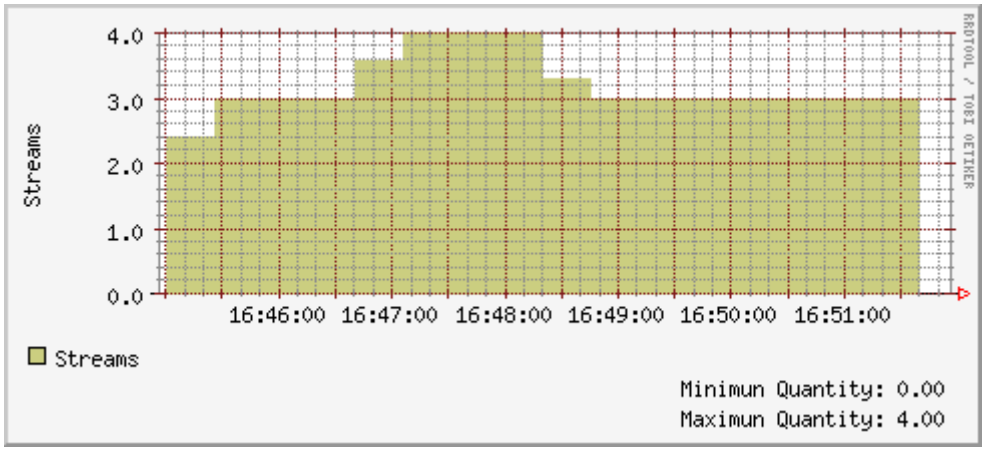


Figura 135: Streams activos

7.4.2.3. Pérdidas de Frames

Este gráfico representa las perdidas de Frames en el tiempo, existe un gráfico para cada tipo de Frame (I, P, B).

En la Figura 136 se presenta un ejemplo de este tipo de gráfico para los Frames I, en la Figura 137 para los Frames P y finalmente en la Figura 138 para los Frames B.

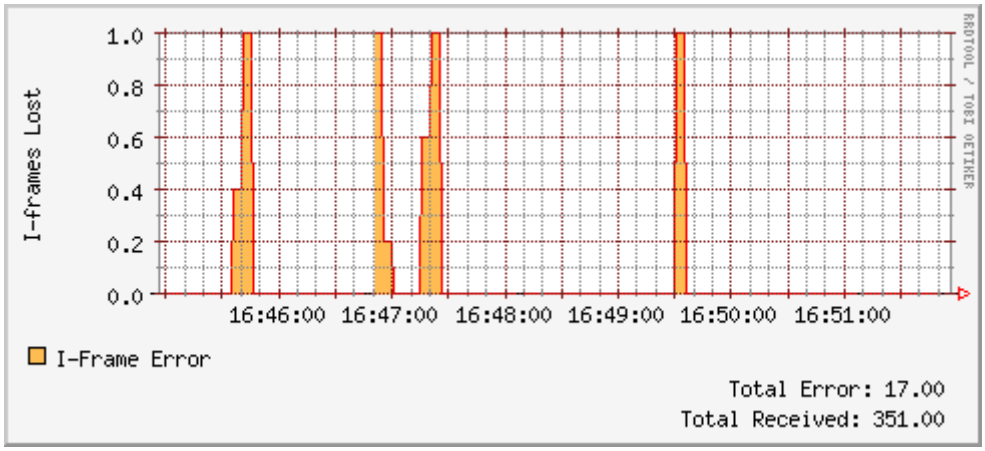


Figura 136: Perdidas de Frames I

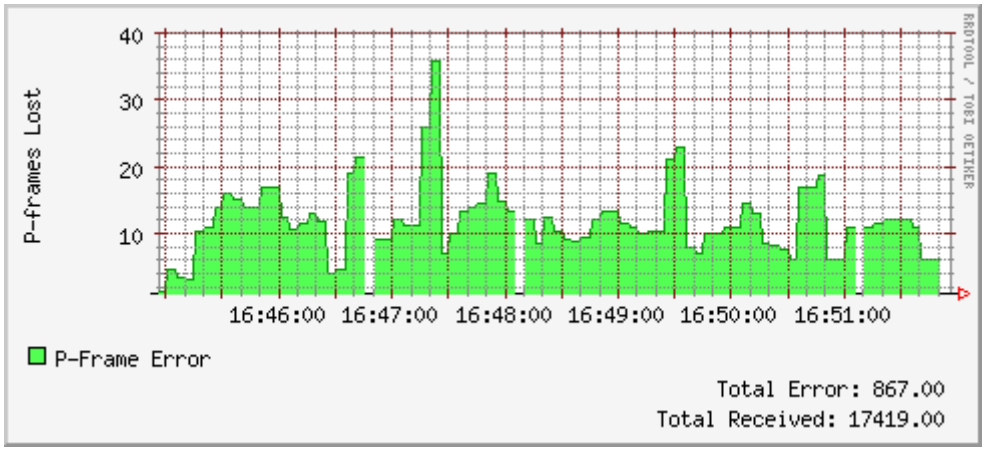


Figura 137: Perdidas de Frames P

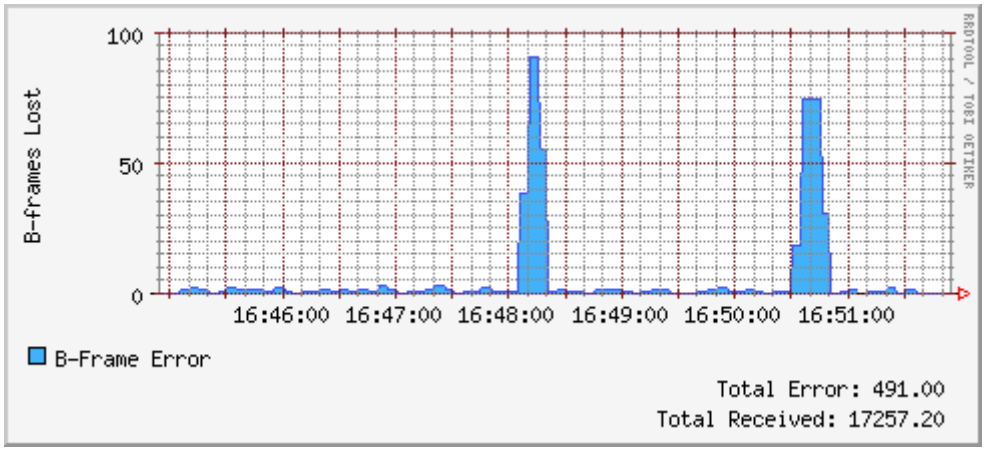


Figura 138: Perdidas de Frames B

7.4.2.4. Bitrate Total

En este gráfico se presenta el bitrate total de video. En la Figura 139 se presenta un ejemplo de este gráfico.

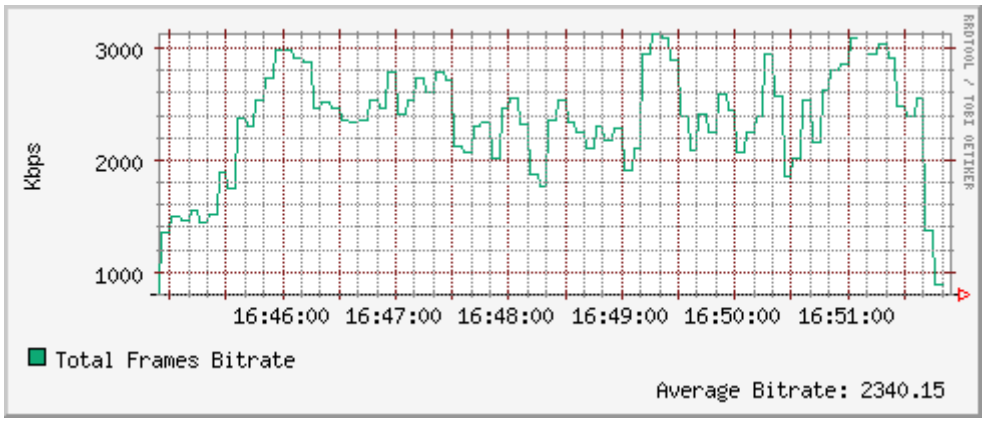


Figura 139: Bitrate total

A continuación veremos una guia explicativa de cómo operar esta aplicación conjuntamente con una explicación en detalle de la interfaz grafica de esta.

7.5. Interfaz de usuario

La aplicación esta compuesta por 3 vistas diferentes, una el index, otra la vista por Streams y finalmente la vista multiple streams. A continuación veremos cada una de están en detalle.

7.5.1. Index Page

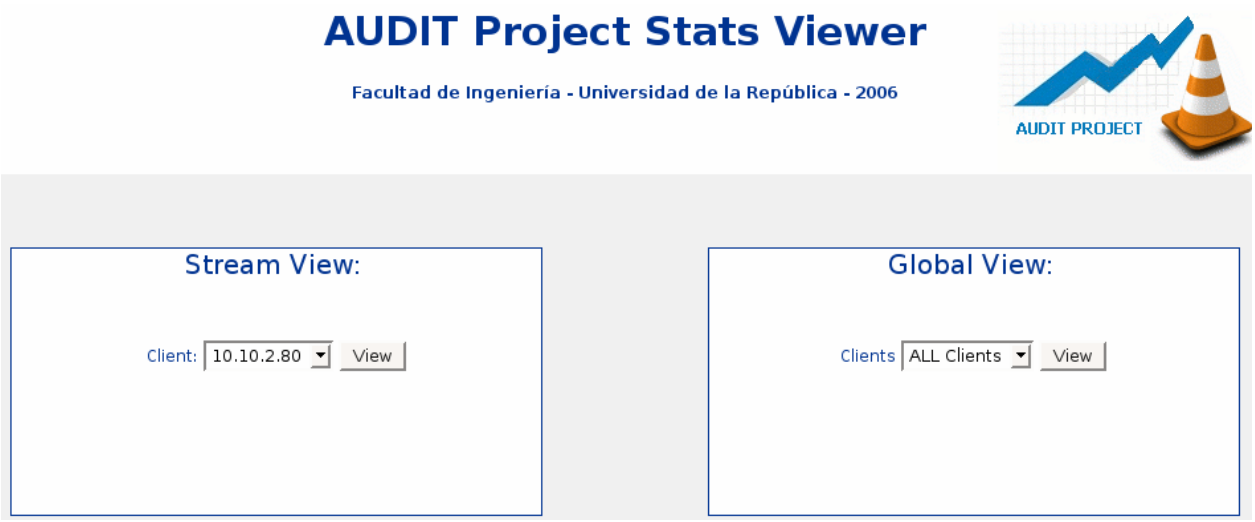


Figura 140: Index Page

Como se puede observar en la Figura 140, en el Index tenemos la posibilidad de ir a algunas de las otras vistas eligiendo el cliente del select que corresponda. En el caso de la vista por stream al elegir el cliente se ira a la vista por stream, del Stream 1 del cliente seleccionado.

7.5.2. Stream View Page

En esta vista vemos los gráficos pertenecientes al grupo “Gráficos a nivel de Streams” junto con los datos genéricos de stream en cuestion (ver Figura 141) los cuales por defecto aparecen no visibles, para hacer visible se debe cliquear sobre el link Stream Info en el margen superior izquierdo del cuerpo de la pagina.

A su vez en esta página tenemos la posibilidad de ir a la vista del resto de los stream del cliente seleccionado (el cual es especificado en el margen superior izquierdo de la cabecera) haciendo clic sobre los botones ubicados en el centro de la cabecera (Stream 2, Stream 3).

Finalmente tenemos la opción de retornar al Index Page haciendo clic sobre el link en el margen superior derecho del cuerpo de la página.

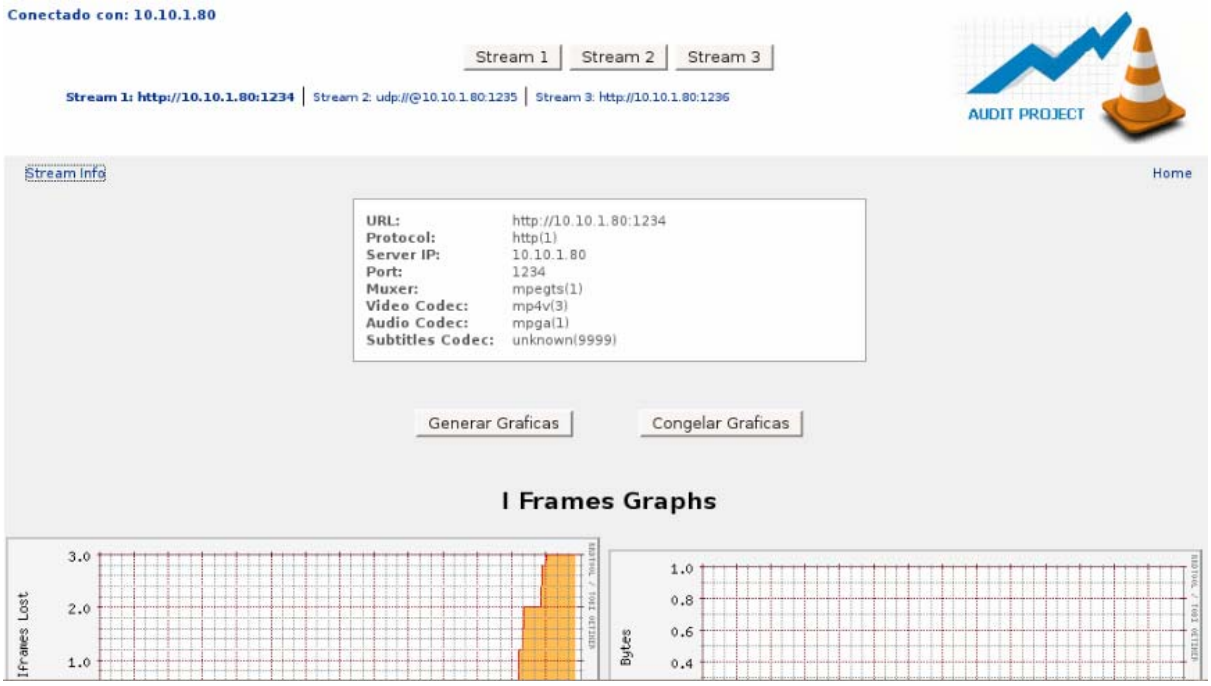


Figura 141: Stream View Page

7.5.3. Global View Page

En esta vista vemos los gráficos pertenecientes al grupo “Gráficos multiple Streams”. Al igual que en la vista anterior podemos cambiar el cliente seleccionado haciendo clic sobre los botones ubicados en el centro de la cabecera (All Clients, Client 1, Client 2) (ver Figura 142).

También en esta vista tenemos un link para retornar al Index Page (en el margen superior derecho del cuerpo de la página).

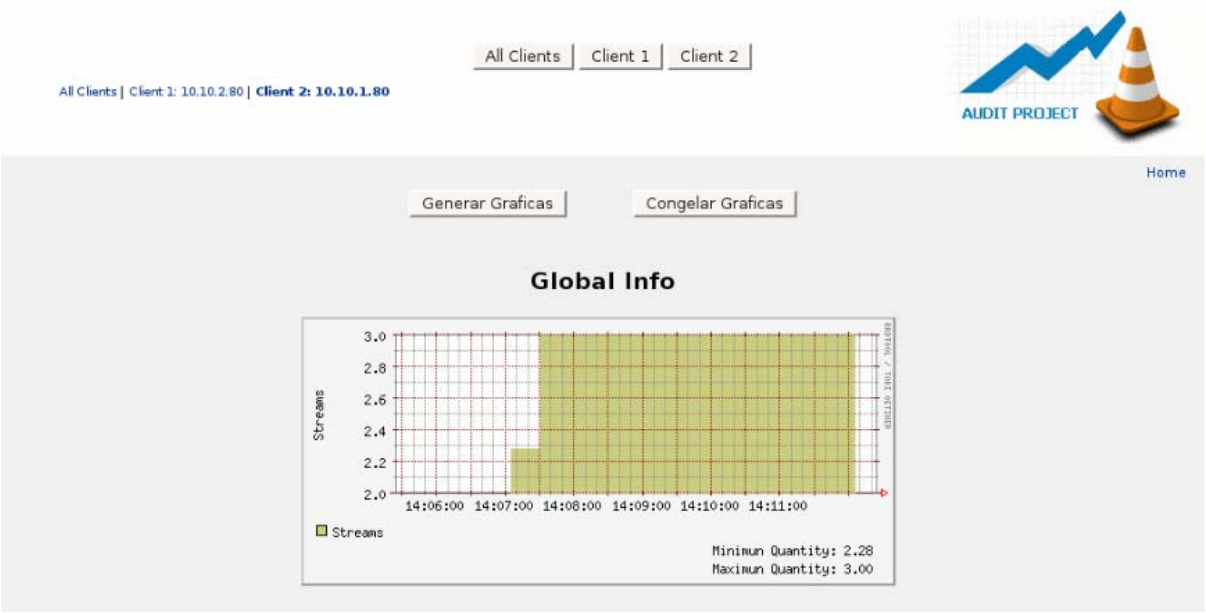


Figura 142: Global View Page

7.6. Ejecución de la aplicación

7.6.1. Configuración de los clientes

Para cada uno de los clientes multimedia a monitorear se debe agregar la configuración del mismo en /inc/clients_config.php, de la siguiente manera:

```
$_clients_info[IDC][IP]        : IP del cliente IDC
$_clients_info[IDC][COMMUNITY] : Comunidad del cliente IDC
```

Por ejemplo:

```
$_clients_info[1][IP]        = '10.10.1.80';
$_clients_info[1][COMMUNITY] = 'vlc_community';
```

Se debe tener en cuenta que el número de cliente identificara al mismo dentro de la aplicación.

7.6.2. Inicialización de los proceso recolectores de datos

Como vimos en la Seccion 7.3.1, es necesario inicializar cada uno de los procesos recolectores de información, por lo que para cada cliente configurado en la aplicación se debe ejecutar el siguiente comando ubicado en el directorio scripts/:

```
php data_collector.php IDC NUM_STREAMS
```

Donde IDC es el identificador del cliente y NUM_STREAMS el número máximo de streamings que este cliente puede recibir.

Por ejemplo:

```
ddv@ddv-ubuntu:$ php data_collector.php 1 5
ddv@ddv-ubuntu:$ php data_collector.php 2 5
```

Una vez inicializados estos procesos se debe inicializar el proceso global mediante el siguiente comando:

```
php global_data_collector.php
```

7.6.3. Visualizacion de los resultados

Finamente para visualizar los resultados obtenidos por los procesos anteriormente inicializados, se debe abrir un browser e ingresar a la url correspondiente con esta aplicación:

Por ejemplo:

http://localhost/server_stats

En donde se accederá al Index Page de la aplicación.

ANEXO VIII

8. Manual de Instalación VLC-AUDIT

En este anexo se presenta una guía de cómo instalar el VLC-AUDIT, el mismo se divide en 2 diferentes secciones, el punto 8.1 en donde se presentan las instalaciones de base para la ejecución de nuestra aplicación y el punto 8.2 donde se explica concretamente como instalar el VLC-AUDIT.

8.1. Instalaciones de base

8.1.1. VLC 0.8.5

Se debe compilar el VLC versión 0.8.5 lo cual requerirá de la instalación de numerosos librerías. No se va a profundizar en este punto, en [Ref36] se presenta una clara guía de cómo realizar la compilación del VLC.

8.1.2. Net-SNMP

Se debe instalar la librería Net-SNMP (ver [Ref37]), tampoco en este punto se profundizará en el como instalar la librería solo se brindara una guía de como configurarla para lograr la integración con el VLC-AUDIT.

Una vez instalada la librería se tendrá a disposición el comando snmpconf mediante el cual se configurara el demonio que atenderá los pedidos mediante SNMP (llamado snmpd).

A continuación se presenta una ejecución de este comando mostrando las opciones a seleccionar:

```
root@ddv-ubuntu:/home/ddv/workspace/VLC-MIB # snmpconf

The following installed configuration files were found:

   1:  ./snmpd.conf
   2:  /etc/snmp/snmpd.conf
   3:  /etc/snmp/snmptrapd.conf

Would you like me to read them in?  Their content will be merged with the
output files created by this session.

Valid answer examples: "all", "none","3","1,2,5"

Read in which (default = all): 1

I can create the following types of configuration files for you.
```

```
Select the file type you wish to create:
(you can create more than one as you run this program)

   1:  snmpd.conf
   2:  snmp.conf
   3:  snmptrapd.conf

Other options: quit

Select File: 1

The configuration information which can be put into snmpd.conf is divided
into sections.  Select a configuration section for snmpd.conf
that you wish to create:

   1:  Trap Destinations
   2:  System Information Setup
   3:  Access Control Setup
   4:  Extending the Agent
   5:  Monitor Various Aspects of the Running Host
   6:  Agent Operating Mode

Other options: finished

Select section: 3

Section: Access Control Setup
Description:
  This section defines who is allowed to talk to your running
  snmp agent.

Select from:

   1:  a SNMPv3 read-write user
   2:  a SNMPv3 read-only user
   3:  a SNMPv1/SNMPv2c read-only access community name
   4:  a SNMPv1/SNMPv2c read-write access community name

Other options: finished, list

Select section: 4

Configuring: rwcommunity
Description:
  a SNMPv1/SNMPv2c read-write access community name
    arguments:  community [default|hostname|network/bits] [oid]

Enter the community name to add read-write access for: vlc_community
The hostname or network address to accept this community name from [RETURN for
all]:
The OID that this community should be restricted to [RETURN for no-
restriction]:

Finished Output: rwcommunity  vlc_community

Section: Access Control Setup
Description:
  This section defines who is allowed to talk to your running
  snmp agent.

Select from:

   1:  a SNMPv3 read-write user
   2:  a SNMPv3 read-only user
   3:  a SNMPv1/SNMPv2c read-only access community name
   4:  a SNMPv1/SNMPv2c read-write access community name
```

```
Other options: finished, list

Select section: finished

The configuration information which can be put into snmpd.conf is divided
into sections.  Select a configuration section for snmpd.conf
that you wish to create:

   1:  Trap Destinations
   2:  System Information Setup
   3:  Access Control Setup
   4:  Extending the Agent
   5:  Monitor Various Aspects of the Running Host
   6:  Agent Operating Mode

Other options: finished

Select section: 6

Section: Agent Operating Mode
Description:
  This section defines how the agent will operate when it
  is running.

Select from:

   1:  Should the agent operate as a master agent or not.
   2:  The system user that the agent runs as.
   3:  The system group that the agent runs as.
   4:  The IP address and port number that the agent will listen on.

Other options: finished, list

Select section: 1

Configuring: master
Description:
  Should the agent operate as a master agent or not.
    Currently, the only supported master agent type for this token
    is "agentx".

    arguments: (on|yes|agentx|all|off|no)

Should the agent run as a AgentX master agent?: agentx

Finished Output: master  agentx

Section: Agent Operating Mode
Description:
  This section defines how the agent will operate when it
  is running.

Select from:

   1:  Should the agent operate as a master agent or not.
   2:  The system user that the agent runs as.
   3:  The system group that the agent runs as.
   4:  The IP address and port number that the agent will listen on.

Other options: finished, list

Select section: finished

The configuration information which can be put into snmpd.conf is divided
into sections.  Select a configuration section for snmpd.conf
```

```
that you wish to create:

   1:  Trap Destinations
   2:  System Information Setup
   3:  Access Control Setup
   4:  Extending the Agent
   5:  Monitor Various Aspects of the Running Host
   6:  Agent Operating Mode

Other options: finished

Select section: finished

I can create the following types of configuration files for you.
Select the file type you wish to create:
(you can create more than one as you run this program)

   1:  snmpd.conf
   2:  snmp.conf
   3:  snmptrapd.conf

Other options: quit

Select File: quit
```

Esto generara un archivo snmpd.conf en el directorio en donde sea ejecutado, como se puede observar en la ejecución anterior se configuro la versión de SNMP a usar (en donde se configuro la comunidad SNMP) y se habilito el modo agentx.

Por ultimo se debe agregar la siguiente línea al final del archivo snmpd.conf recientemente generado:

```
agentxsocket localhost:705
```

Copiar el dicho archivo a /usr/share/snmp:

```
cp snmpd.conf /usr/share/snmp
```

Y Finalmente reinicial el demonio SNMP, para que tome las nuevas configuraciones:

```
/etc/init.d/snmpd restart
```

8.2. Instalacion del VLC-AUDIT

Una vez compilado el VLC 0.8.5 y instalada la librería Net-SNMP podremos pasar a instalar el VLC-AUDIT.

Llamaremos VLC_INSTALL_DIR al directorio en donde se ubico el código fuente del VLC.

Se debe ejecutar el parche audit.patch quien realizara las diferentes modificaciones al código fuente del VLC 0.8.5. Para esto de debe ejecutar el siguiente comando:

```
cd VLC_INSTALL_DIR
patch -p1 < ../audit.patch
```

Antes de realizar la compilación del VLC-AUDIT se deben modificar las siguientes variables de entorno para lograr la compilación del VLC contra la librería Net-SNMP:

```
CPPFLAGS="-I/usr/local/include":`net-snmp-config --cflags`:$CPPFLAGS
LDFLAGS=`net-snmp-config --agent-libs`
export CPPFLAGS LDFLAGS
```

Luego se debe seguir con la compilación normal del VLC, para lo cual se requerirá realizar un bootstrap, ejecutar el configure con las mismas opciones que se ejecuto al instalar el VLC, ejecutar el make y finalmente el make install.

A continuación se presenta un ejemplo de estos pasos:

```
./bootstrap
./configure --enable-x11 --enable-xvideo --disable-gtk --enable-sdl --enable-
ffmpeg \
--with-ffmpeg-mp3lame --enable-mad --enable-libdvbpsi --enable-a52 --enable-
dts \
--enable-libmpeg2 --enable-dvdnav --enable-faad --enable-vorbis --enable-ogg -
-enable-theora \
--enable-faac --enable-mkv --enable-freetype --enable-fribidi --enable-speex -
-enable-flac \
--enable-livedotcom --with-livedotcom-tree=/usr/lib/live --enable-caca --
enable-skins \
--enable-skins2 --enable-alsa --disable-kde --disable-qt --enable-wxwidgets --
enable-ncurses \
--enable-release
make
make install
```

Seguido de esto, se deberá realizar que el demonio SNMP reconozca la MIB VLC-AUDIT-MIB, lo cual se logra con los siguientes comandos:

```
cd VLC_INSTALL_DIR
cp VLC-AUDIT-MIB.txt /usr/share/snmp/mibs
export MIBS=+ VLC-AUDIT-MIB
```

Para chequear que el demonio este renociendo la MIB se puede ejecutar el siguiente comando, que el si todo se encuentra bien mostrara la estructura de la MIB en forma arborescente.

```
snmptranslate -M+. -mVLC-AUDIT-MIB -Tp -IR vlcAuditMib
```

Finalmente para verificar la correcta instalación del VLC-AUDIT se podrá ejecutar el siguiente comando:

```
vlc -l
```

En donde se mostraran todos los módulos incluidos en el VLC, donde tendrían que aparecer todos los módulos pertenecientes a nuestra aplicación (es decir el qos, snmp, loggerAudit y qos-duplicate).