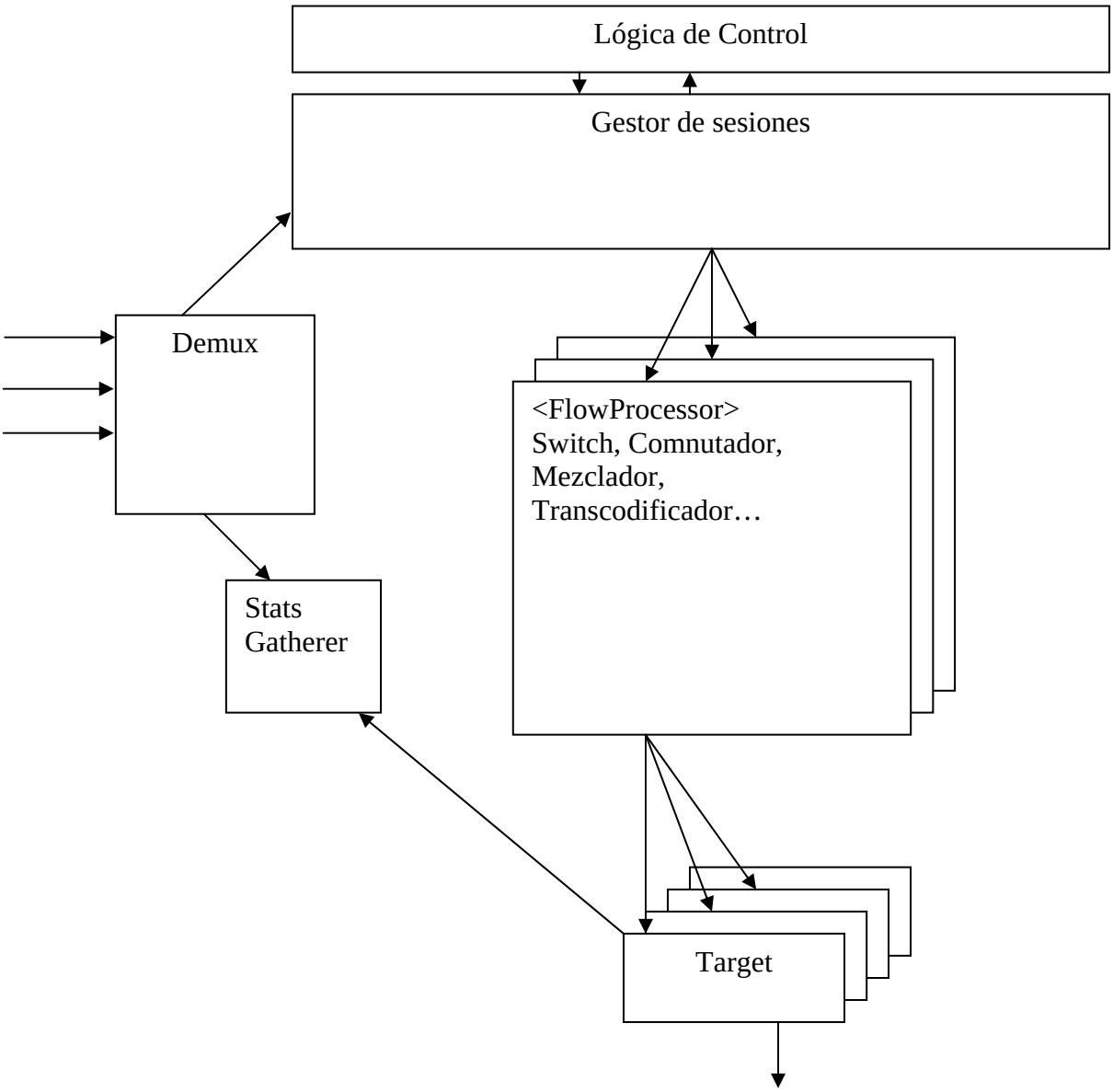
SAMURAI MCU

Índice

1) Arquitectura general



Básicamente la MCU se subdivide en varios módulos que contrendrán a su vez, submódulos que realicen funciones más específicas. La **lógica de control** será la encargada de recibir comándos desde el **DTE** y de la **aplicación SIP**, dicha lógica de control únicamente se comunicará con el **gestor de sesiones**, que será el encargado de mantener bajo control en todo momento las sesiones existentes en la MCU. El **DEMUX** es el encargado de recibir datos desde la red y pasar dichos datos al **gestor de sesiones**, además, el **DEMUX** se comunicará con el recolector de estadísticas (**stats Gatherer**) y será capaz de recuperar paquetes en flujos que se envíen con protección FEC.

El módulo **FlowProcessor** representa la superclase de la cual derivarán los conmutadores de datos, transcodificadores, sumadores, y cualquier otra funcionalidad que se quiera poder añadir en el futuro.

De manera independiente existirá un módulo **target** por destino y por PT conectado a cualquier sesión en la MCU (es decir, un target para 138.4.24.19 con PT 96 y otro distinto para 138.4.24.19 con PT 100), de tal manera que cada **flowProcessor** mantendrá una relación en todo momento de los **targets** a los cuales deba replicar los datos con los que trabaja, dichos destinos serán unicamente impuestos por el control superior a la MCU a través de la **lógica de control** que se lo comunicará a su vez al **gestor de sesiones**.

Como requisitos generales de la MCU debemos cumplir:

- Diseño modular y abierto a futuras mejoras
- Comportamiento simple, todo el trabajo lógico para nivel superior.
- Posibilidad de trabajar en Ipv4/Ipv6
- Mono-hilo (mejora de eficiencia)
- Obtener una MCU tan completa como para que pueda ser usada por Ares/Isabel/Samurai/Daidalos
- Soporte SSM

a) Actores

Participantes:

Los participantes se identifican por su ID, que en principio tiene que ser único dentro de la sesión a la que pertenezcan, la información extra que se necesita para poder enviar y recibir datos a un participante son <IP dest, rx1, rx2…,<pt1,tx1>,<pt2,tx2> > , por tanto, siguiendo la filosofía de dejar el control lógico a la entidad superior, esta debería ser quien informe a la MCU de todos los datos del participante.

Debe de poderse cambiar tanto el **BW** como el módulo de **FEC** dinámicamente, no se puede cambiar <IP,puertos rx>.

Un mismo participante, no puede pertenecer a varias sesiones simultaneamente. Para cambiar un participante de sesión, se elimina el participante y se crea uno nuevo en una sesión diferente.

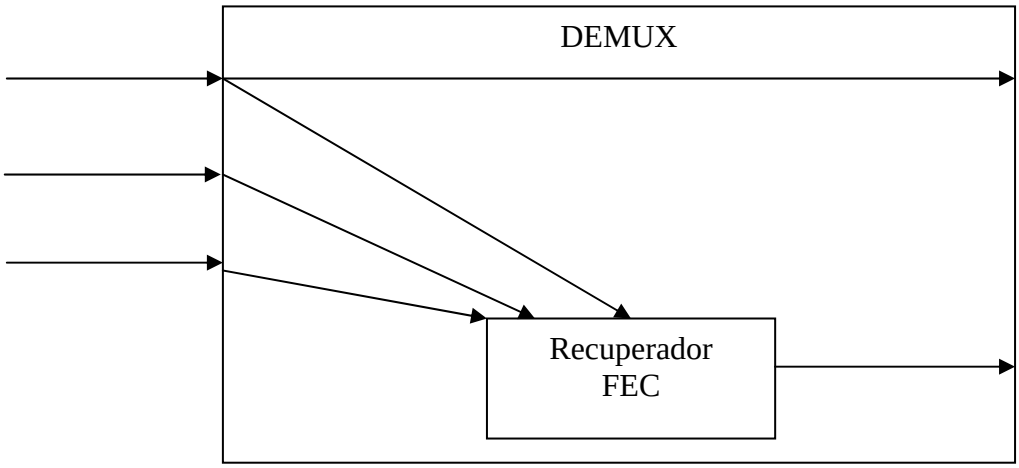
Cada participante mantiene una información:

· Lista de <RX port>
· Lista de duplas <TX port,PT>
· Modo de video + informacion modo
· Modo audio + información modo

b) Casos de uso

2) ARQUITECTURA POR MÓDULOS

a) Demux



El DEMUX es el encargado de recibir TODOS los datos que le llegan desde los distintos destinos de la red, se trata únicamente de un receptor tonto, no mantiene ninguna relación de los destinos posibles ni nada parecido, sus funciones principales serán:

- Recibir datos del exterior (RTP,RTCP)
- Recuperar datos FEC
- Comunicar la llegada de datos al recolector de estadísticas
- No reordena (lo hace la APP superior)
- No elimina duplicados (no suelen aparecer duplicados, poco impacto)
- Reenviar los datos al gestor de sesiones (paso de referencias, no copia en memoria, ademas informa de ip origen y puerto por el que recibe el paquete)

El DEMUX debe trabajar únicamente con información RTP, es decir, tenemos que darle la posibilidad de que varios flujos (audio, video , …) del mismo o distintos SSRCs sean recibidos por el mismo o distintos puertos.

El DEMUX, necesitará información en el momento de su contrucción (puntero al gestor de sesiones), y durante el resto de la ejecución debe de ofrecer métodos para abrir puertos de escucha y cerrarlos dinámicamente.

El DEMUX, contendrá un número arbritario de sockets de escucha que introducirá en el planificador de tareas para atender eventos de llegada de datos.

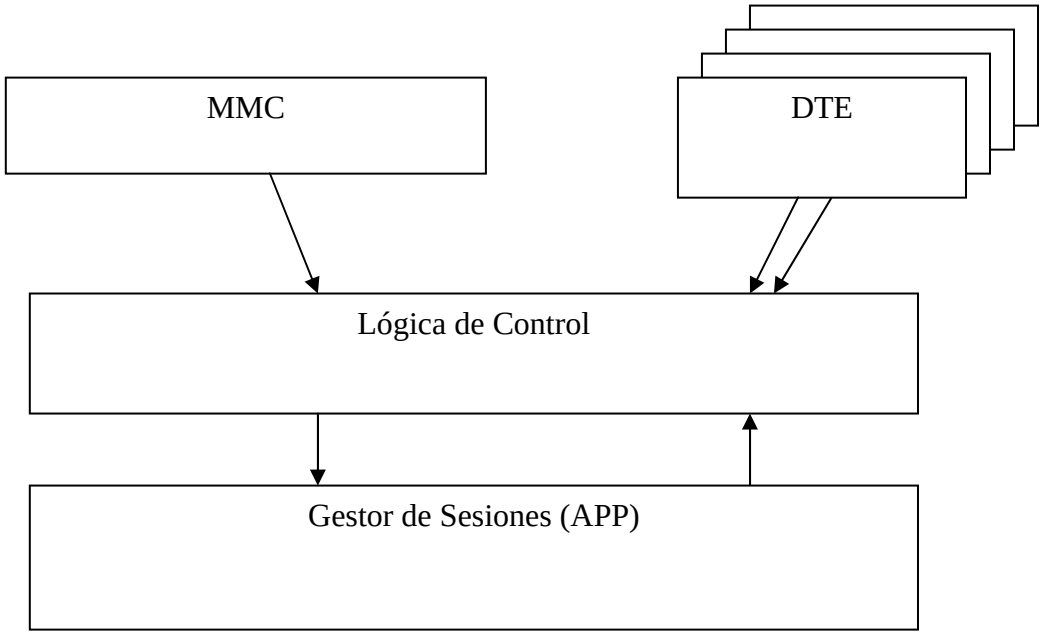
El resto de información debe ser desconocida para el DEMUX.

b) Lógica de control

La lógica de control se implementará de la misma manera que se implementó en el irouter, ofrecerá en un puerto conocido un servidor TCP donde se podrán conectar un número arbritario de aplicaciones de control, aceptará comandos tanto del control SIP como del DTE. Detalles de comandos de control en el (Apéndice A).

La lógica de control utilizará para ejecutar los comandos una referencia al gestor de sesiones (APP) como se lleva haciendo en todos los controles que hacen uso de la ICF.

El socket servidor se introducirá como tarea en el planificador, para poder atender los eventos de peticiones de conexión. Una vez creada la conexión, el socket que atienda a cada petición también será introducido en el planificador para atender la llegada de datos (comandos).



c) Gestor de sesiones

Se trata del módulo principal y más importante de la aplicación, debe llevar una relación de todas las sesiones existentes en cada momento, y de todo el trabajo que hay que realizar sobre los distintos flujos. En principio, es la clase candidata a derivar de application_t y por tanto todas las tareas serán introducidas dentro del planificador del gestor de sesiones.

A la hora de su creación, será el encargado a su vez de crear el módulo de lógica de control, el recolector de estadísticas, el DEMUX e introducirlos como tareas en su planificador.

El gestor de sesiones debería de ser accesible como APP (variable globlal) como se ha venido haciendo en otras aplicaciones, de tal manera que todo el mundo pueda comunicarse con el gestor de sesiones.

Como se puede observar en el esquema, el gestor de sesiones debe poder controlar todos los comandos que son recibidos por la lógica de control. Asi pués, el gestor de sesiones debe poder comunicarse con distintos módulos y realizar distintas tareas:

- Funciones propias:

  · Crear Sesión
  · Eliminar Sesión
  · Crear Participante (en sesión)
  · Eliminar participante (de sesión)
  · Crear target
  · Eliminar target
  · Crear flowProcessor
  · Eliminar flowProcessor
  · Información **completa** de qué debe recibir cada participante de cada sesión.
  · Debe saber que flujo debe enviar a cada flowProcessor: En principio los modos de audio y video se han dispuesto de manera que uno excluye al otro, de tal manera que un flujo únicamente deberá tener un flowProcessor destino.

- Comunicación con DEMUX:

  · Crear DEMUX
  · Abrir puerto de escucha
  · Cerrar puerto de escucha
  · Recibir paquete RTP: el DEMUX de pasar IP origen y puerto por el que ha recibido el paquete para que el gestor obtenga la sesión y el participante del que procede el paquete.
  · Eliminar DEMUX

- Comunicación con flowProcessor

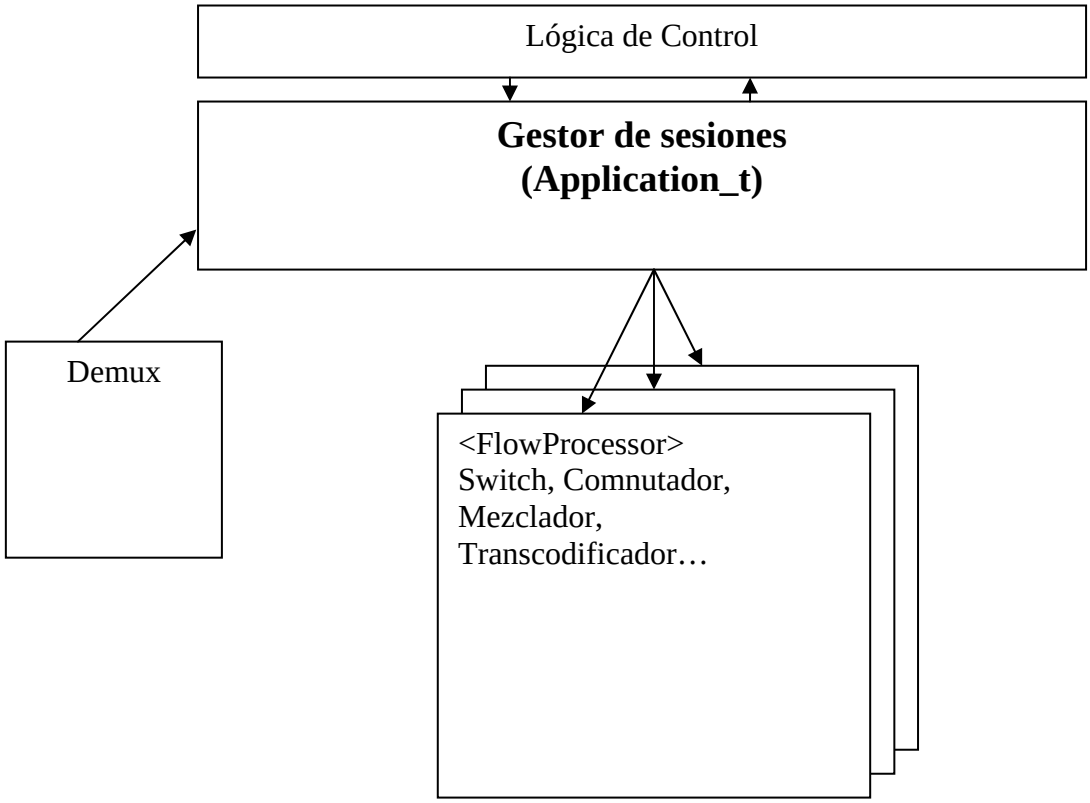
  · Creación y destrucción: Con todo lo que esto conlleva, especificar si es conmutador, transcodificador, mezclador, etc… y todos los argumentos que se necesiten.

  · Paso de datos (paquete RTP) a los flowProcessors: el gestor de sesiones únicamente debe saber a qué flowProcessor pertenece cada paquete, o tirarlo si no pertenece a ninguno, pero no debe trabajar para **nada** con los datos.

El gestor de sesiones trabajará con referencias a ‘flowProcessors’ utilizando ‘casting’ para apuntar a sumadores, trancodificadores, etc…

El gestor de sesiones debe mantener también una referencia a todos los posibles **targets,** es el encargado de informar a cada flowProcessor de los **targets** a los que debe enviar la información de salida.

El gestor de sesiones debe mantener una estructura eficaz y **muy meditada** para mantener sesiones, participantes dentro de las sesiones y toda la información que necesitase cada participante.



d) FlowProcessors

Los flowProcessors representan la superclase encargada de trabajar con los fllujos, mantendrá ciertos métodos, variables y contenedores comunes, y dependiendo de la función que queramos derivar de un flowProcessor sobreescribiremos ciertos métodos y añadiremos, en caso necesario métodos nuevos. En un principio, de este módulo deben derivar los mezcladores de audio, los transcodificadores de audio/video, los conmutadores y el generador de video en mosaico.

Los flowProcessor se comunican con el gestor de sesiones, de donde reciben los paquetes RTP, y con los targets, que representan los destinos.

Los flowProcessor deben mantener una referencia a todos los targets que debe enviar el flujo de salida, que dependiendo del caso, no tiene por que ser el de entrada, ya que en la clase se trabaja con los datos.

Las funciones que debe ofrecer al gestor de sesiones son :

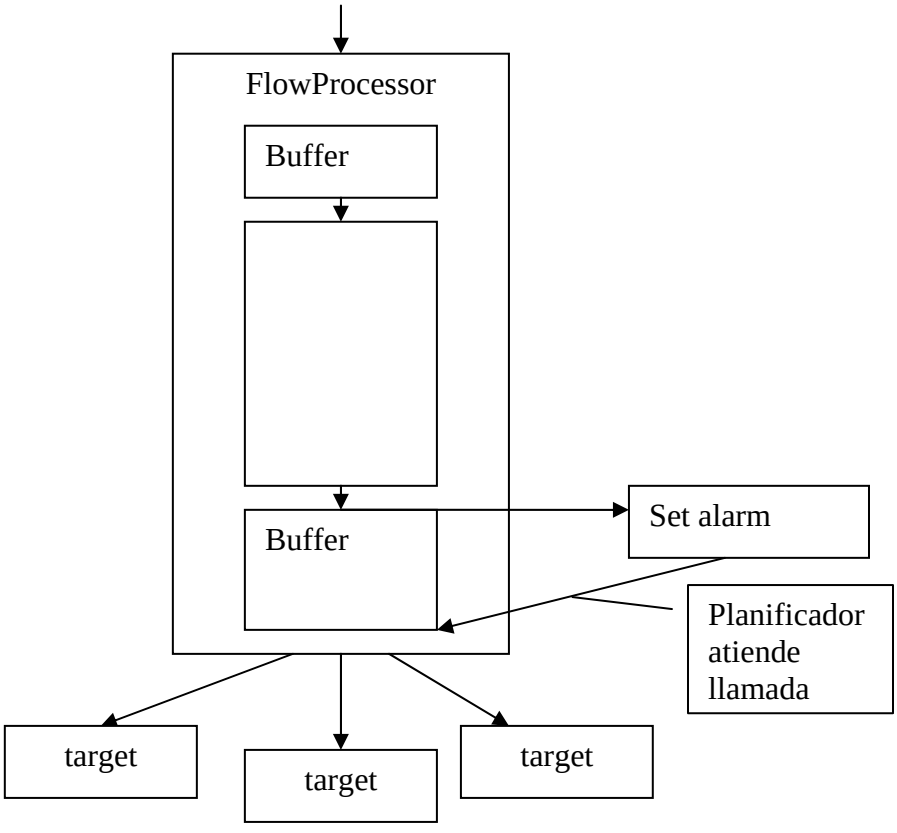
· Creación y destrucción
· Añadir y descartar destino
· Recibir paquete dato

El flowProcessor únicamente se comunicará con los targets para darles paquetes de datos en la salida.

De los flowProcessor derivaremos los servicios principales que ofrece la MCU, tales como conmutación de paquetes, transcodificación, mezclado, etc…

La idea es que la MCU sea mono-hilo, pero el uso de un hilo independiente dentro de cada flowProcessor liberaría de mucha complejidad en cuanto al código. De tal manera que el gestor de sesiones simplementa debería ir llenando un buffer de recepción en el flowProcessor y se olvidaría del paquete. A la hora de programar la aplicación podremos hacerlo de manera mucho más modular. Aconsejo que los búfferes de entrada y salida de datos, el trabajador del flowProcessor y las zonas críticas necesarias se encuentren definidas en la propia clase padre **flowProcessor.**

Existirá clases búffer con métodos **push( )** y **pop( )** a la entrada y salida de los flowProcessor. El hilo del gestor de sesiones llamará a push, el hilo del flowProcessor a pop , a la salida, el hilo del flowProcessor llamará a push y generará un evento (colocar periodo de 1ms a heartBeat) para que sea atendido por el planificador el buffer de salida llamando a pop y distribuya el paquete por los distintos targets. El flowProcessor debe ser tan amplio como para permitir que dentro de la clase se pueda realizar cualquier tipo de operación.



e) Recolector de estadísticas

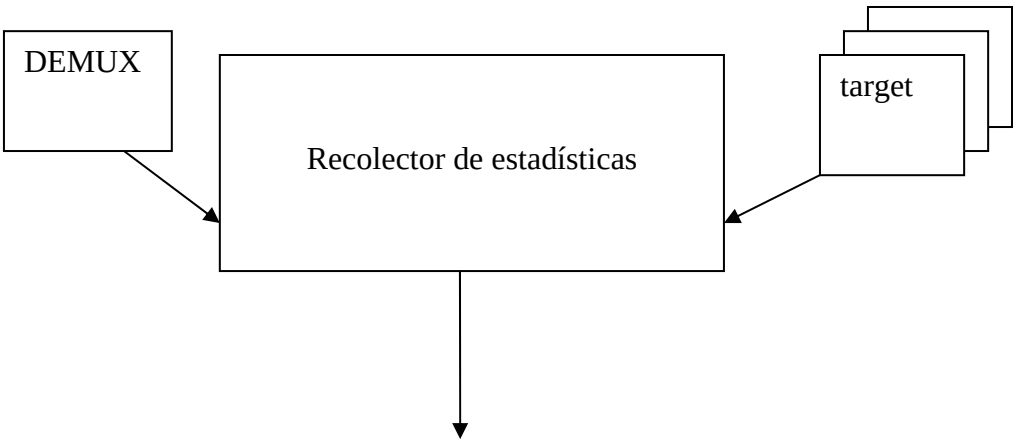
El recolector de estadísticas es básicamente del mismo estilo que el usado por otros programas ya realizados, debe ofrecer métodos para que tanto el DEMUX, como los distintos targets puedan informar sobre el paso de información, este módulo se encargará de calcular todo tipo de estadísticas tales como:

· Llegada de Datos por destino:

- Desorden de paquetes
- Duplicados de paquetes
- Pérdidas de paquetes
- Payloads
- Paquetes recuperados
- Ancho de banda de datos
- Ancho de banda FEC
- Jitter

· Salida de Datos por destino:

- Ancho de banda datos
- Ancho de banda FEC
- Payloads
- Paquetes descartados

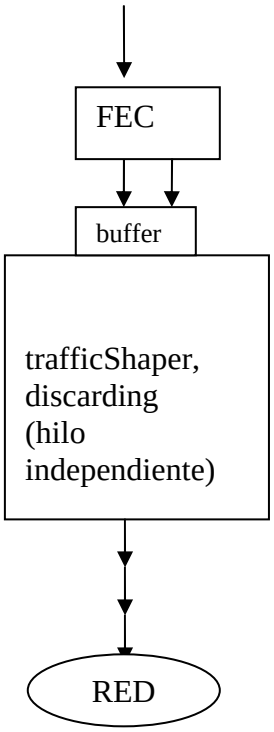


f) targets

Los targets representan los distintos destinos por flujo existentes. Los target se caracterízan por la IP destino, el puerto destino, y el PT de los datos a enviar, es decir, manejaremos el mismo target independientemente del flowProcessor que haya trabajado con el dato. Un flowProcessor tendrá a su salida un paquete RTP, con una IP destino y

un puerto. Se buscará si existe el target adecuado y se enviará a través de dicho target, en caso contrario se descartará el paquete. Únicamente puede darse esta situación si se ha mandado el comando de enviar a un participante un video sin especificar antes el puerto al que debe enviarse dicho video, es decir, debido a un uso erroneo de la MCU.

El target tiene como funciónes principales, proteger el flujo, y realizar descarte y aplanado de tráfico. El modo de trabajo , despues de mucha meditación creo que no debería ser mediante un pipeline, sino siguiendo el siguiente esquema:



El hilo que manda el paquete RTP al target, ejecuta las operaciones de FEC necesarios, se pasa el paquete al buffer(y el de paridad en caso de haberse generado) de recepción del trafficShaper, el cual trabaja en un hilo diferente, realizando el aplanado de tráfico y el descarte de paquetes, este último es el último a realizar y se utilizara un sistema básico de asignación de fichas por intervalo de tiempo, cuando se acaban las fichas, el resto de paquetes son descartados hasta el próximo intervalo.

El modo de trabajar del trafficShaper es midiendo un ancho de banda medio en intervalos de tiempo, y procesando cada paquete con dicha frecuencia, hay que ver con que frecuencia se calcula el ancho de banda medio y el rango de histéresis con el que trabajar para no ser demasiado restrictivos ni demasiado permisivos.

Apéndice A

Los comandos de control que debe soportar :

Primitivas de sesión

- **create_session**(n_sesion):
- **remove_session**(n_session):

Primitivas de participante:

- **new_participant**(n_session,
  ip,
  rx_port1,
  [,rx_port2,rx_port3,…])

Se crea un participante y se definen los puertos que debe escuchar el DEMUX. Cuando el DEMUX reciba un paquete mediante la IP origen, y el puerto por el que se recibe podremos saber el participante del que se trata.

Devuelve : ID del participante si no existe la IP en dicha sesión.
ERROR :: participante ya existe.
ERROR :: sesión no existe.

- **bind_rtp** (n_session,
  id_part,
  pt,
  tx_port)

Mediante este comando le indicamos a la MCU que en caso de enviar un flujo a un participante con cierto PT, debemos enviarlo al puerto tx_port.

Devuelve :: ID único, deberá utilizarse en el caso de unbind

- **unbind_rtp** (n_session,
  id_part,
  ID)

Dejamos de enviar el flujo a dicho puerto.

- **bind_rtcp** (n_session,
  id_part,
  tx_port)

Mediante este comando le indicamos a la MCU a qué puerto debe enviarse el flujo RTCP.

Devuelve :: ID único, deberá utilizarse en el caso de unbind

- **unbind_rtcp** (n_session,

id_part,
ID)

Dejamos de enviar flujo RTCP a dicho puerto.

- **Configure_participant**(n_session,
  ID,
  PT,
  FEC,
  BW)

  Con este comando configuramos el pipeline de salida del participante para cierto flujo, es decir, si queremos proteger el flujo con FEC (k = 0 para fec off) y el ancho de banda máximo que soporta (traffic shaper y discarding). Por el PT podremos saber el target al que nos referimos, ya que cada participante tiene un número de targets que se diferencian por el PT.

- **Remove_participant**(n_session, ID):: Elimina el participante de la sesión.
  Cuestión :: ¿Que pasa si una sesión se queda vacía?
  Mi respuesta :: Creo que se debería de dejar la sesión abierta a nivel de MCU, es decir que si le mandan crear un nuevo participante en dicha sesión lo haga, no devuelva error. Pero que se cierren los puertos en el DEMUX que ya no se vayan a escuchar.

- **Get _codecs**( ) :: devuelve lista de codecs soporados (PTs) → Creo que deberíamos de dejar de hablar con cadenas (MPEG4:: MPEG4, etc ,…) como utilizamos en otros protocolos de control y hablar directamente Payloads, más o menos como hace SIP, volviendo a la cuestión anterior, creo que ofreciendo los códecs que conocemos el nivel superior debería llegar a un acuerdo con cada destino y no dejarle esa tarea a la MCU.

- **Receive_video_mode** (n_session,
  id_part,
  mode
  [,PT,BW,SSRC])

Con este comando decidimos el modo en el que el participante 'id_part' debe ver los video que recibe, argumentos:

- mode : SWITCH_MODE → Modo por defecto, la MCU reenvia los videos
  GRID_MODE → Modo mosaico, la MCU envia un mosaico con los videos que desea el participante recibir.

Únicamente para GRID_MODE tienen valor los argumentos PT, BW y SSRC (si dejamos SSRC=-1 la MCU elije SSRC) de salida a la donde indicamos el códec de salida mediante el Payload y el ancho de banda que tiene disponible dicho video.

- **Receive_video**(n_session,
  Id_participante_receptor,
  Id_participante_envia1
  [,PT,]

[BW])

El comando actuaría de la siguiente manera:

Si estamos en SWITCH_MODE, se envía el video especificado al participante que así lo desee, si se desea que se cambie el tipo de codec , debe especificarse el payload en PT, si se desea limitar el máximo ancho de banda debe especificarse en BW. Si deseamos que el codec sea el del vídeo original, debemos indicarlo con un -1. Si indicamos un PT debemos indicar un BW **obligatoriamente**.

En caso de encontrarnos en GRID_MODE, se rehace el mosaico añadiendo el video especificado. Los argumentos PT y BW no se tienen en cuenta en este modo.

- **discard_video**(n_session,
  id_participante_receptor,
  id_participante_envia)

SWITCH_MODE : Se elimina el participante de los video que recibe, si se trata de conmutación, simplemente deja de conmutarlo, si se trata de transcodificación, elimina del flowProcessor transcodificador dicho destino, si un flowProcessor transcodificador se queda sin destinos se elimina.

GRID_MODE : Debe rehacerse el mosaico, dejando de utilizar el vídeo indicado.

- **receive_audio_mode** (n_session,
  id_part,
  mode
  [,PT])

Cuestión :: ¿Siempre se suma? Cuando simplemente se quiere conmutar, ¿como se especifica?

Con este comando decidimos si el participante desea recibir el audio por separado (SWITCH_MODE) o una suma de todos los audios (MIX_MODE), en caso de especificar MIX_MODE , debemos indicar el PT de salida (Codec). En caso de SWITCH_MODE podemos opcionalmente indicar PT si deseamos cambiar el codec de salida, si simplemente deseamos reenviar el flujo, lo indicamos con PT = -1.

- **receive_audio**(n_session,
  id_participant_receptor,
  id_participant_envia)

Si nos encontramos en SWITCH_MODE, enviaremos dicho flujo alparticipante, sin más modificación en caso de que se haya configurado PT = -1, se transcodifica en caso de PT distinto de -1. En caso de encontrarnos en MIX_MODE, añadimos el flujo a la hora de mezclarlos.

- **discard_audio(** n_session,
  id_participant_receptor,
  id_participant_envia)

SWITCH_MODE : deja de conmutarse el flujo de audio
MIX_MODE : deja de sumarse el flujo de audio

En cuanto a la comunicación con el DTE:

Tenemos que acostumbrarnos a pensar que trabajamos en MPEG-4, no en MJPEG, que los códecs tienen la capacidad de trabajar en CBR (que es un gran invento), que ellos mismos se encargan de ajustar la calidad al máximo permitido por el BW según las variaciones de la imagen y que por jugar manualmente con el grab_size , el FR, y Q, **jamás** conseguiremos un resultado ni lejanamente comparable con lo que se consigue imponiendo al codec un cierto CRB. En mi opinión, jugar con estos parámetros supone estancarse en la manera que se tenía de trabajar con MJPEG, con la falsa creencia de que funcionará igual de bien en MPEG-4. Dicho esto:

- **Set**(PT1,PT2,Size,FR,Q) :
  PT1 : Códec de audio
  PT2 : Códec de video
  Size : tamaño de captura (aaaxbbb)
  FR : Frec. de refresco
  Q : Calidad de códec de video

  Este comando (existente por compatibilidad con Murcia) no acabo de encajarlo, no entiendo a que participante se refiere, a que sesión, a qué video de los que recibe alguien, a qué audio, en caso de que no se sumen qué pasa... (DISCUTIR EN REUNION QUÉ HACER CON ESTO)

  Hay que encontrar primitiva que encaje con el nuevo esquema.

El resto de primitivas se pueden mantener intáctas:

get_audio_losses( ): devuelve el porcentaje de perdidas en audio desde startcount.
get_video_losses( ): devuelve el porcentaje de perdidas en video desde startcount.
get_losses get_losses( ): devuelve el porcentaje de perdidas desde start count.
start_count start_count( ): resetea el contador de perdidas.