UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería de Telecomunicación

Implementación de un software de monitorización de QoE para la transmisión de video en streaming para los servicios de video de próxima generación

TRABAJO FIN DE ESTUDIOS

GRADO EN INGENIERÍA TELEMÁTICA

Autor: García Casanova, Alejandro

Director: Cano Baños, María Dolores

Cartagena, 6 de febrero de 2021

Índice

Tabla de elementos

Capítulo 1 - Introducción

1.1 Contexto y motivos del proyecto

Los servicios de retransmisión de video en la red llevan años creciendo constantemente, y a grandes velocidades, estudios de Parks Associates muestran que más de 310 millones de hogares conectados tendrán al menos un servicio OTT (Over The Top) para 2024. (1) Estos servicios ya conforman una gran porción de todo el tráfico de Internet hoy en día, y van a suponer un gran reto en el futuro si no se toman las medidas adecuadas, como se ha podido ver en el primer cuarto de 2020 con la aparición global del *SARS-CoV-2.* Esto ha supuesto un gran aumento en el número de consumidores, Netflix durante este periodo ha aumentado un 22% el número de clientes globales llegando a los 182.9 millones (2). Al mismo tiempo se ha puesto la línea de red a su máximo de ocupación durante los periodos de confinamiento, tanto en Europa como en otros países las autoridades han pedidos a proveedores de servicios como YouTube o Netflix que reduzcan la calidad de los contenidos para reducir el ancho de banda que consumen. (3) Por ello las grandes operadoras e investigadores de todo el mundo se centran en reducir el ancho de banda utilizado por estos servicios, sin disminuir la calidad que esperan recibir los usuarios finales.

Las operadoras tratan estos problemas como problemas de baja eficiencia en el uso del ancho de banda, el cual está libre durante gran parte del día a excepción de las horas de máxima carga. Para afrontarlo, primero se debe monitorizar la red a través de QoS (Quality of Service) para extraer datos exactos de la red y poder priorizar diferentes contenidos, en segunda instancia, y teniendo de base los datos previamente obtenidos, hay que saber cómo afectan estos posibles problemas al usuario, mediante QoE (Quality of Experience) midiendo los problemas que perciben y cómo se ven afectados por ellos.

El comportamiento de los usuarios frente a los nuevos servicios ha cambiado recientemente, ahora se espera que los contenidos se reproduzcan de forma casi inmediata (con la menor carga inicial), con pocas (o ninguna) parada de carga durante la reproducción, de la menor duración posible, y con la mayor calidad posible en cada momento, incluso en las franjas horarias de mayor estrés de la red (de 20:00 a 23:00). (4)

Para ajustar y estandarizar los procesos para cumplir estos objetivos, la ITU-T (International Telecommunication Union-Telecommunication Standardization Sector) publica diferentes recomendaciones de algoritmos y modelos a usar.

1.2 Descripción del Proyecto

Este documento aborda una implementación por software para monitorizar la QoE de los servicios de Streaming (retransmisión) de video tanto Live (en directo) como On-Demand (servicios de video a la carta), basado en modelos paramétricos (con datos extraídos de la red) y usando tecnologías de Machine Learning, propuestos en la recomendación ITU-T P.1203.

1.3 Estructura del Trabajo de Fin de Estudios

Este trabajo está dividido en cuatro capítulos principales que abordan el proyecto desde distintos ámbitos, para darle una forma organizada y completa.

Comienza con una introducción en la que se pone en contexto al lector sobre la situación actual respecto a los servicios de streaming multimedia, y se le indica lo que puede esperar a continuación en este proyecto.

En el capítulo dos se explica en detalle la recomendación ITU-T P.1203 sobre modelos de medición de calidad paramétricos con descarga progresiva y adaptativa sobre transportes fiables, primero hablando de la institución encargada de la recomendación, y posteriormente analizando los documentos, desde un punto de vista enfocado en su implementación.

A continuación, en el capítulo tres se disecciona el código utilizado para implementar la recomendación anterior, explicando en detalle todas las librerías utilizadas, sus funciones y el uso que han tenido en el programa.

Al finalizar, en el capítulo cuatro se presentará una conclusión del proyecto y se propondrán diferentes líneas de investigación para el proyecto según lo aprendido durante el desarrollo de este.

Capítulo 2 - Recomendación ITU-T P.1203

2.1 Explicación de la recomendación

La ITU-T P.1203 es una recomendación por parte de la International Telecommunication Union-Telecommunication Standardization Sector (ITU-T), órgano cuya función es estudiar los diferentes aspectos necesarios para la estandarización de las telecomunicaciones. (5)

Ilustración 1 – Logotipo de la ITU

En este escrito se presentan varios módulos de medición de calidad, que en conjunto forman un modelo que es capaz de calcular la calidad de experiencia en la que un individuo percibe un documento audiovisual transmitido por la red.

El modelo también está dividido en cuatro modos de funcionamiento, dependiendo de los datos disponibles, y da como resultado una puntuación sobre cinco en distintos aspectos y una puntuación sobre cinco global. Los modos de funcionamiento con más datos dan resultados más contrastados.

Cabe destacar que la puntuación obtenida puede discrepar con respecto a una puntuación calculada por evaluación personal, debido a impedimentos tales como niveles de audio, colores en la imagen, o el dispositivo de visualización.

2.2 Objetivo de la recomendación

Su uso está destinado al control de calidades en servicios audiovisuales en Demanda/Directo basados en encriptación HTTP/TCP para los modelos 0 y 1 y en servicios audiovisuales de Demanda/Directo basados en HTTP/TCP no encriptado para todos los modelos.

El modelo ha sido validado para los siguientes factores:

- Compresión de video: ITU-T H.264/AVC perfil alto, 75 kbit/s – 12.5 Mbit/s.
  Especificado en el módulo Pv de la recomendación [ITU-T P.1203.1].
- Compresión de audio: AAC-LC baja complejidad, 32-196 kbit/s.
  Especificado en el módulo Pa de la recomendación [ITU-T P.1203.2].
- Contenido de video: Videos con contenido de distinta complejidad espacio temporal.
- Carga inicial y eventos de parada: Especificado en el módulo Pq de la recomendación [ITU-T P.1203.3].
- Resolución de pantalla: Full HD (1920x1080 pixeles)
- Dispositivo de reproducción: Pantallas de PC/TV, Smartphone (Samsung Galaxy S5)
- Adaptación de calidad: Variación de calidad por cambio entre diferentes niveles. Especificado en la recomendación [ITU-T P.1203.1].
- Ratio de fotogramas: Entre 8 y 30 fotogramas por segundo.

2.3 Disección de la recomendación

MODO	ENCRIPTACION	ENTRADA	COMPLEJIDAD
0	Encriptación de datos de medios audiovisuales y de cabecera de paquete	Metadatos	Baja
1	Encriptación de datos de medios audiovisuales	Metadatos e información de tipo y tamaño del paquete	Baja
2	Sin encriptación	Metadatos y hasta un 2% de la transmisión	Media
3	Sin encriptación	Metadatos y cualquier otra información de la transmisión	Ilimitada

Tabla 1 – Modos de operación

Esquema de los distintos módulos que forman el modelo

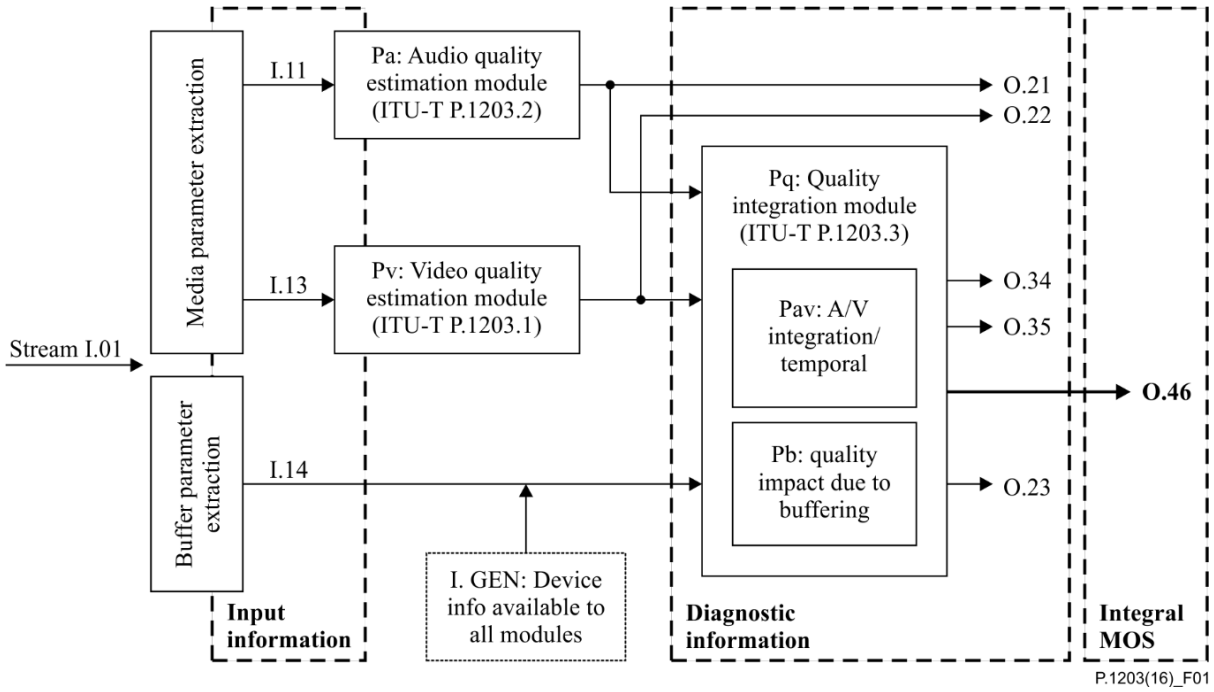

Ilustración 2 - Esquema del modelo

2.3.1 Datos de entrada del modelo

I.GEN: Resolución y tipo de dispositivo. Tipo de dispositivo como:
PC/TV: Pantallas iguales o entre 24 y 100 pulgadas.
Móvil: Pantallas iguales o menores a 10 pulgadas.

- **I.11:** Información de codificación de audio.
- **I.13:** Información de codificación de video.
- **I.14:** Información de retraso de carga y eventos de parada.

Tabla de descripción de datos de entrada:

ID	Descripción	Valores	Frecuencia	Disponible para los modos
I.GEN				
0	Resolución de la imagen mostrada al usuario	Número de pixeles (WxH) mostrados	Para cada sesión	Todos
1	Tipo de dispositivo donde se reproduce el contenido	"PC" o "mobile"	Para cada sesión	Todos
I.11				
2	Bitrate del audio	Bitrate en kbit/s	Para cada segmento	Todos
3	Duración del segmento	Duración en segundos	Para cada segmento	Todos
4	Número de ventana de audio	Número entero, empezando por 1	Para cada segmento	1,2,3
5	Tamaño de la ventana de audio	Tamaño de la ventana en bytes	Para cada ventana	1,2,3
6	Duración de la ventana de audio	Duración en segundos	Para cada ventana	1,2,3
7	Codificación de audio	Uno de los siguientes: AAC-LC, AAC-HEv1, AAC-HEv2, AC3	Para cada segmento	Todos
ID	Descripción	Valores	Frecuencia	Disponible para los modos

8	Frecuencia de muestreo de audio	En Hz	Para cada segmento	Todos
9	Número de canales de audio	2	Para cada segmento	Todos
10	Transmisión de bits de audio	Bytes de audio codificados en la ventana	Para cada ventana	2,3
I.13				
11	Bitrate del video	Bitrate en kbit/s	Para cada segmento	Todos
12	Fotogramas por segundo del video	Número de Fotogramas por segundo	Para cada segmento	Todos
13	Duración del segmento	Duración en segundos	Para cada segmento	Todos
14	Resolución de la codificación de video	Número de píxeles (WxH) transmitidos por video	Para cada segmento	Todos
15	Codificación del video y perfil	H264-Alto	Para cada segmento	Todos
16	Número de fotograma del video	Número entero, empezando en 1, en orden siguiendo los fotogramas	Para cada ventana	1,2,3
17	Duración de la ventana de video	Duración de la ventana en segundos	Para cada ventana	1,2,3
18	Marca de tiempo de los fotogramas mostrados	La marca de tiempo de los fotogramas al mostrarse	Para cada ventana	1,2,3
19	Marca de tiempo de los fotogramas descodificados	La marca de tiempo de los fotogramas descodificados	Para cada ventana	1,2,3
20	Tamaño de la ventana de video	El tamaño de la ventana de video codificado en bytes	Para cada ventana	1,2,3
21	Tipo de cada imagen	“I” o “Non-I” para el modo 1 “I”/”P”/”B” para los modos 2 y 3	Para cada ventana	1,2,3
22	Transmisión de bits de video	Bytes de video codificados en la ventana	Para cada ventana	2,3
I.14				

23	Parada/Carga inicial	Tiempo de carga en segundos, respecto al inicio del clip Vale 0 si no hay retardo de carga	Por cada evento de parada	Todos
24	Duración del evento	Tiempo del evento de parada en segundos	Por cada evento de parada	Todos

Tabla 2 – Datos de entrada

Con respecto a los eventos de parada, solo están contemplados los ocurridos al cargar el contenido o producidos fortuitamente, no los producidos por el usuario, como al pausar, buscar un contenido o cambiar de calidad manualmente.

2.3.2 Datos de salida del modelo

Para poder calificar las diferentes características del documento audiovisual transmitido por red, se utiliza un sistema de puntuación entre 1 y 5, donde 1 significa malo y 5 significa excelente.
Características:

- **O.21:** Calidad de codificación de audio por intervalo de salida de muestras
  - Una puntuación por cada segundo de una sesión.

- **O.22:** Calidad de codificación de video por intervalo de salida de muestras.
  - Una puntuación por cada segundo de una sesión.

- **O.23:** Percepción de parada
  - Puntuación única para la sesión
- **O.34:** Calidad de codificación del segmento audiovisual por intervalo de salida de muestras.
  - Una puntuación por cada segmento de una sesión.
  - Tamaño de sesión sincronizado con O.21 y O.22

- **O.35:** Calidad de codificación final del documento audiovisual.
  - Una puntuación por sesión
  - Incluye la integración temporal

- **O.46:** Calidad final de la sesión del documento audiovisual.
  - Una puntuación por sesión
  - Incluye los eventos de carga inicial y parada

2.3.3 Módulo de ventana de medición

Definida como *"Información de video o audio de uno o más segmentos, pertenecientes a un nivel de calidad específico, usado como entrada para los módulos de Pv o Pa para cierto tiempo de salida ts."*

Son utilizadas por los módulos Pv ([ITU-T P.1203.1]) y Pa ([ITU-T P.1203.2]) para calcular una puntuación en dicho intervalo, con los datos presentes dentro de ese intervalo únicamente y con segmentos de la misma calidad.

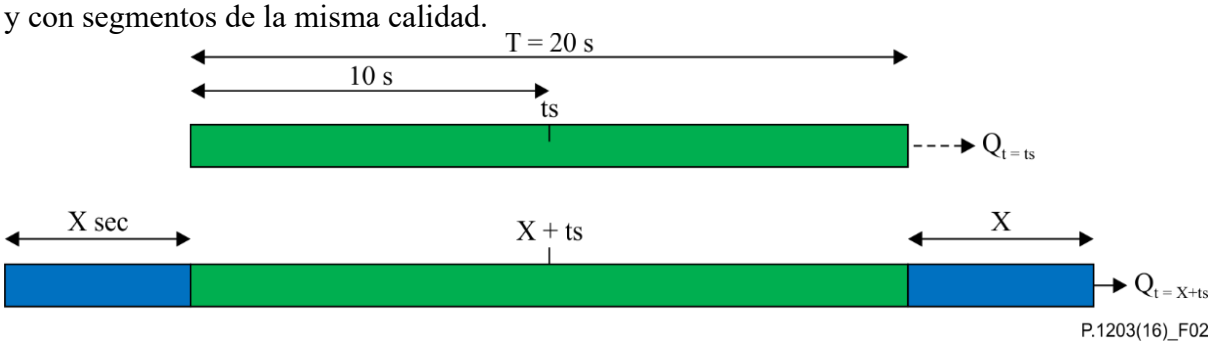

Ilustración 3 - Ventana de medición

Cada ventana debe utilizar la información expuesta en la tabla previamente mostrada, dependiendo del modo de funcionamiento del modelo.

Si las muestras no pueden ser obtenidas del bitstream o de los metadatos, como en el modo 0, se crearán artificialmente S*R muestras, sin información de carga, siendo S la longitud del segmento y R la ratio de muestra. Cada muestra tendrá una duración y una marca temporal de decodificación("DTS").

El código completo utilizado para resolver este procedimiento lo podemos encontrar dentro del paquete *"itu-p1203"*, en el script *"measurementwindow.py"*.

Aquí explico los principales conceptos y su pseudo ejecución.

Primero se crea una lista vacía que se llenará con muestras.

```
self._frames = [ ]  # actual measurement window
```

Si la duración total de las muestras es mayor a 20 segundos, se elimina la primera muestra de la lista.

```
if self._acc_frame_dur + frame["duration"] > self.max_size:
    removed_frame = self._frames.pop(0)
    self._removed_frames.append(removed_frame)
    self._acc_frame_dur -= removed_frame["duration"]
```

Se van añadiendo muestras.

```
self._frames.append(frame)
self._acc_frame_dur += frame["duration"]
self._acc_pvs_dur += frame["duration"]
```

Se comprueba si se puede calcular una puntuación.

Si el tiempo de salida es 0 y la duración acumulada es menor de 11, ningún resultado es obtenido.

```
    if self._last_score_output_at == 0 and round(self._acc_pvs_dur, 5) <
self._half_window_size + 1:
```

Pero si la duración acumulada menos 10, es mayor o igual al tiempo de salida más 1 se calculará la calidad, y el tiempo de salida será aumentado en 1.

```
    if self._acc_pvs_dur - self._half_window_size >= self._last_score_output_at + 1:
        next_score_output_at = self._last_score_output_at + 1
        self._last_score_output_at = next_score_output_at
        return True
```

Al completarse el procedimiento anterior, una marca temporal de salida es calculada redondeando hacia abajo respecto a la duración total, y es incrementada en 1.

```
    final_sample_timestamp = math.floor(self._acc_pvs_dur)
    output_sample_timestamp = self._last_score_output_at + 1
```

Las muestras que no son necesarias se eliminan de la lista, siendo estas aquellas cuales su "DTS" es menor que la actual marca de tiempo menos 10. Y con las restantes se va calcula una puntuación

```
while output_sample_timestamp <= final_sample_timestamp:

        while round(self._frames[0]["dts"], 5) < output_sample_timestamp -
self._half_window_size:
                removed_frame = self._frames.pop(0)
                removed_duration += removed_frame["duration"]
                self._removed_frames.append(removed_frame)
                self._acc_frame_dur -= removed_frame["duration"]

        if self._score_callback:
                self._score_callback(output_sample_timestamp, self._frames)

        output_sample_timestamp += 1
return
```

Cuando las listas llegan a los módulos, estos se encargan de discernir qué muestras son válidas para el cálculo de la puntuación. Para ello utilizan el siguiente código el cual podemos encontrar en el script *"utils.py"* (nombre de la función).

```
target_frame = frames[output_sample_index]
target_ql = get_chunk_hash(target_frame, type)
window = [output_sample_index]
if get_chunk_hash(frames[output_sample_index - 1], type) == target_ql:
        i = output_sample_index - 1
        window = [i] + window
        if i-1 != -1:
                i -= 1
                while get_chunk_hash(frames[i], type) == get_chunk_hash(frames[i+1],
type):
                        window = [i] + window
                        if i-1==-1:
                                break
                        else:
                                i-=1
if output_sample_index + 1 != len(frames):
        if get_chunk_hash(frames[output_sample_index + 1], type) == target_ql:
                i = output_sample_index + 1
                window.append(i)
        if i+1 != len(frames):
                i += 1
                while get_chunk_hash(frames[i], type) == get_chunk_hash(frames[i-1], type):
                        window.append(i)
                        if i+1 == len(frames):
                                break
                                else:
                                        i+=1

return [frames[w] for w in window]
```

Así obtenemos la lista final de la ventana de muestras para que los módulos Pv y Pa puedan obtener una puntuación. (6)

2.3.4 Módulo de estimación de calidad de video - Pv

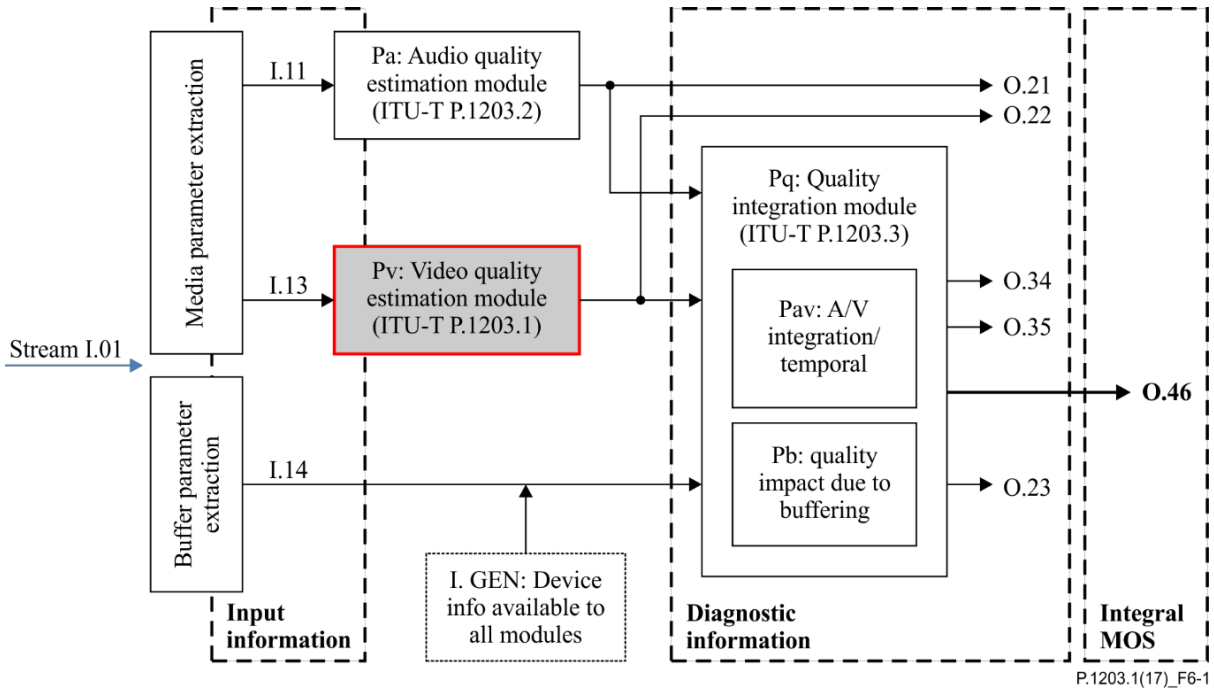

Ilustración 4 – Esquema del módulo Pv

Aquí vamos a ver en profundidad el módulo de cálculo de estimación de calidad de video.

Este módulo está destinado a monitorizar los servicios de video sobre TCP, tanto en servicios que funcionan en el más alto nivel (OTT - over the top), por ejemplo, YouTube, como en los servicios de video/audio gestionados por tcp utilizando los protocolos HTTP/TCP/IP y RTMP/TCP/IP.

Este módulo ha sido validado para los siguientes factores:

- Longitud del video: Máximo 20 segundos
- Contenedor del Bitstream: Formato MPEG-2 para la Trama de Transporte de los segmentos de video codificado
- Implementación del Codificador/Descodificador: El modelo ha sido ajustado usando ITU-T H.264/MPEG-4 AVC High profile: x264 (ffmpeg), a partir de estos ajustes se ha creado un marco común para los demás códecs.
- Tamaño de corte: Un corte por cada fotograma de video
- Detección de cambio de escena: No
- Configuración de x264: Mediana
- Resolución/Bitrate de video:
    - 240p: 75-150 kbit/s
    - 360p: 220-450 kbit/s
    - 480p: 375-750 kbit/s
    - 720p: 1,050-2,100 kbit/s
    - 1080p: 1,875-12,500 kbit/s
- Group of pictures (GOP): 1 segundo de duración, solo IBBBP
- Duración de segmento: Entre 1 y 9 segundos, determina cada cuanto se puede cambiar entre distintas calidades

2.3.4.1 Datos de entrada

La entrada de datos, según el modo de utilización, para este módulo está en la tabla 2 en el apartado I.13, como se puede apreciar en el esquema anterior.

2.3.4.2 Datos de salida

Como salida de datos este módulo solo tiene una salida: O.22 explicado anteriormente en el apartado 2.3.2.

2.3.4.3 Funcionamiento

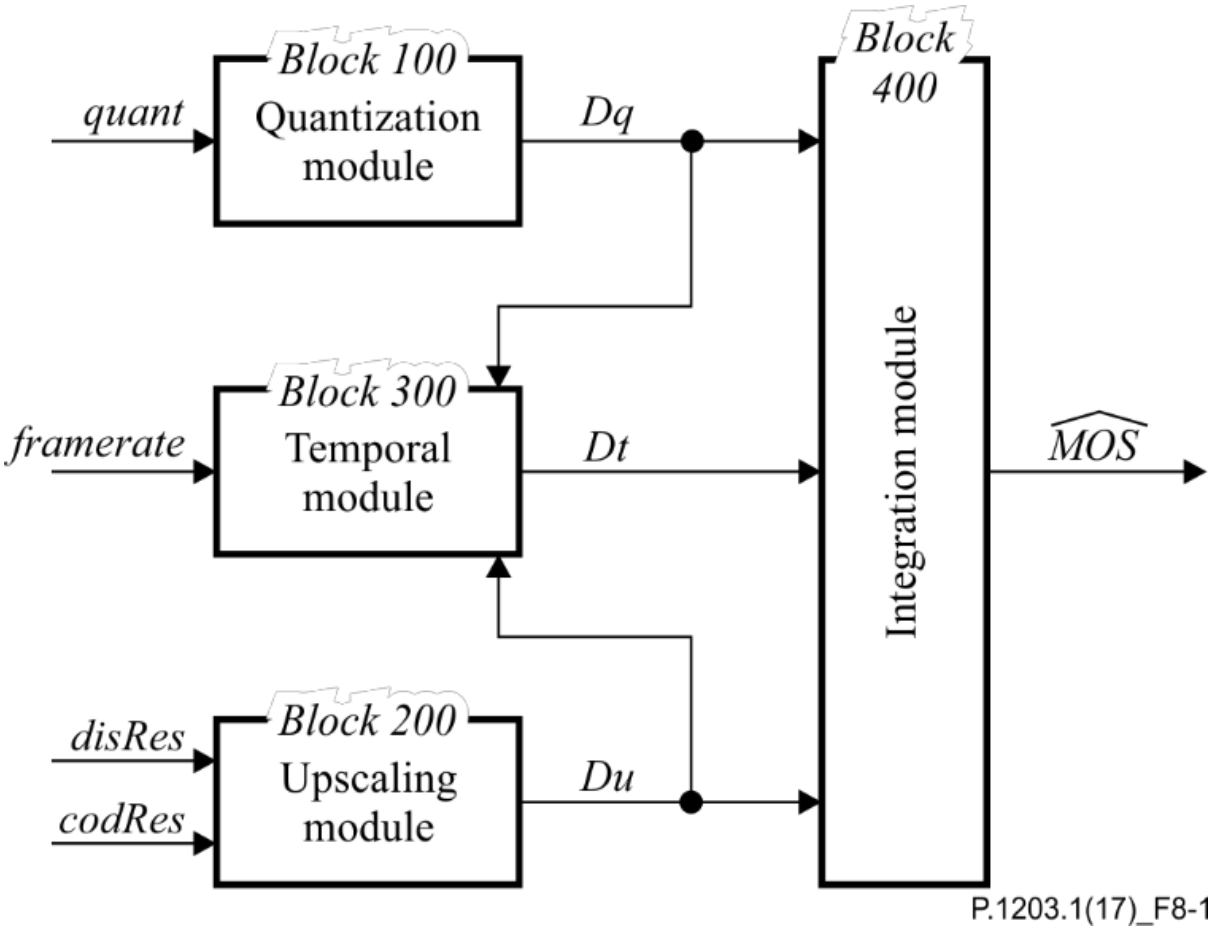

Ilustración 5 - Esquema conceptual de Pv

Antes de explicar el funcionamiento del modelo voy a definir algunos parámetros que veremos aplicados posteriormente.

- *quant* ∈ [0,1]: Parámetro que señala la degradación de cuantificación, su cálculo depende de la variante utilizada en el Quantization module
- *scaleFactor* ∈ [0,1]: Parámetro que señala la degradación de escalado (Upscaling module)
- *framerate(fr)*: Número de imágenes por segundo en el video
- *bitrate(br)***:** Bitrate del video en Kbits/s

- *disRes*: La resolución del video mostrado, en número de píxeles (Por ejemplo, en resolución HD -> *disRes* = 1920*1080 = 2073600 píxeles)
- *codRes*: La resolución de codificación del video, en número de píxeles (Por ejemplo, codRes = 854*480 = 409920 píxeles)
- *displaySize*: La longitud diagonal del video mostrado en centímetros o pulgadas
- *deviceType*: Parámetro para describir el dispositivo de reproducción como portátil ("handheld") o fijo ("TV"), si no es conocido se puede estimar utilizando el *displaySize*, si es igual o mayor de 33 cm (13") asumimos que es fijo, si es menor de 33 cm (13") asumimos que es portátil

Vamos ahora a definir los distintos bloques del módulo Pv de forma general, y después veremos cómo actúan en cada modo de funcionamiento.

2.3.4.4 Módulo de cuantificación

Tiene como objetivo calcular la degradación de cuantificación *Dq* (Degradation quantization)

Calculamos *Dq* como:

$$Dq = 100 - RfromMOS(\widehat{MOSq})$$

Siendo un máximo según la siguiente función:

$$Dq = max\,(min(Dq, 100)\,, 0)$$

$(\widehat{MOSq})$ se calcula como:

$$(\widehat{MOSq}) = q_1 + q_2 * exp\,(q_3 * quant)$$

Siendo también un máximo tal que:

$$(\widehat{MOSq}) = max\,(min(\,(\widehat{MOSq}), 5)\,, 1)$$

Donde q_1= 4.66, q_2= -0.07 y q_3= 4.06 son valores prefijados.

Ahora nos centramos en la función *RfromMOS*

RfromMOS es expresada como:

RfromMOS:$\Re \longmapsto \Re$
MOS $\longmapsto$ Q := RfromMOS (MOS)

Siendo que *RfromMOS* es la función inversa de *MOSfromR*, y al no estar $RfromMOS$ explícitamente definida, para poder obtener un valor Q dado un cierto MOS este se debe calcularse mirando las tablas calculadas con *MOSfromR,* y usando interpolación lineal.

El error absoluto de esta doble conversión debe ser menor de 0.01 en la escala de MOS dentro del rango $[MOS_{MIN}\,, MOS_{MAX}]$tal que:

$$|MOS - MOSfromR(RfromMOS(MOS))| < 0.01\ MOS \in [MOS_{MIN}, MOS_{MAX}]$$

MOSfromR es expresada como:

$$MOSfromR: \Re \longmapsto \Re$$
$$Q \longmapsto MOS := MOSfromR(Q)$$

Y se calcula como:

$$MOS = MOS\ MIN + (MOS\ MAX - MOS\ MIN) * \frac{Q}{100} +$$
$$Q * (Q - 60) * (100 - Q) * 0.000007$$

$$MOS = min(MOS_{MAX}, max(MOS, MOS_{MIN}))$$

Donde $MOS_{MAX} = 4.9$ y $MOS_{MIN} = 1.05$

Vamos ahora a ver un ejemplo de cómo construir una tabla que podamos utilizar para *MOSfromR*:

Primero definimos una tabla *h* con un conjunto de llaves *k* y un conjunto de valores *V* donde *h(k)* es el valor respectivo *v* para la llave *k*. Solo debe haber un único valor *v* para cada llave *k*.

Se permite $h(MOSfromR(v)) = v$ para cada $v \in \{0,0.25,0.5,\ldots,100\}$ siendo $h(MOS_{MIN}) = 0$.
Para obtener el valor de *Q* dado un valor *MOS* primero tenemos que asegurarnos que se encuentra en el rango:

$$MOS = min(MOS_{MAX}, max(MOS, MOS_{MIN}))$$

Y a continuación, calculamos *Q* utilizando la interpolación lineal:

$$Q = \frac{V_p(K_q - MOS) + V_q(MOS - K_p)}{K_q - K_p}$$

Donde K_p, K_q son los puntos de interpolación más cercanos en la tabla *h*, y V_p,V_q son los valores respectivos a esos puntos.

2.3.4.5 Módulo de escalado

Tiene como objetivo calcular la degradación de escalado *Du* (Degradation Upscaling).

Calculamos *Du* como:

$$Du = u_1 * log\ 10\ (u_2 * (scaleFactor - 1) + 1))$$

Siendo un máximo según la siguiente función:

$$Du = max\ (min(Du, 100), 0)$$

$scaleFactor$ se calcula como:

$$scaleFactor = max\left(\frac{disRes}{codRes}, 1\right)$$

Donde u_1= 72.61, u_2= 0.32 siendo valores prefijados.

2.3.4.6 Módulo temporal

Tiene como objetivo calcular la degradación temporal *Dt* (Degradation Temporal).

Calculamos *Dt* como:

$$Dt = \begin{cases} D_{t1} - D_{t2} - D_{t3}, framerate < 24 \\ 0, framerate \geq 24 \end{cases}$$

Siendo un máximo según la siguiente función:

$$D_t = max\left(min(D_t, 100), 0\right)$$

D_{t1}(Degradación temporal pura) se calcula como:

$$D_{t1} = \frac{100 * (t_1 - t_2 * framerate)}{t_3 + framerate}$$

D_{t2}(Variable de compensación 1 relacionada con la codificación) se calcula como:

$$D_{t2} = \frac{D_q * (t_1 - t_2 * framerate)}{t_3 + framerate}$$

D_{t3}(Variable de compensación 2 relacionada con la codificación) se calcula como:

$$D_{t3} = \frac{D_u * (t_1 - t_2 * framerate)}{t_3 + framerate}$$

Donde t_1= 30.98, t_2= 1.29, t_3= 64.65 siendo valores prefijados.

2.3.4.7 Módulo de Integración

En este módulo se calcula el $\widehat{MOS}$ perceptivo de la parte de video utilizando la siguiente fórmula:

$$\widehat{MOS} = MOSfromR(\hat{Q})$$

La función $MOSfromR$ ya ha sido explicada previamente en el apartado 2.3.4.4.

$\hat{Q}$ es la codificación de video estimada y se expresa como:
$\hat{Q} \in [0,100]$

$\hat{Q} = 100 - D =$

$$\begin{cases} 100 - max\left(min\left((100 - \widehat{Q_{max}}) + Dq - Dt2 - Dt3{,}100\right), 0\right), framerate < 24 \\ 100 - max\left(min\left((100 - \widehat{Q_{max}}) + Dq, 100\right), 0\right), framerate \geq 24 \end{cases}$$

Siendo $\widehat{Q_{max}}$ la máxima calidad respecto a *Du* y D_{t1} se calcula como:

$$\widehat{Q_{max}} = \begin{cases} 100 - Du - Dt1, framerate < 24 \\ 100 - Du, framerate \geq 24 \end{cases}$$

Y siendo *D* la degradación en general, en la representación de un video.

Con los valores obtenidos en los anteriores módulos (*Dq*, *Du*, *Dt*) podemos obtener el valor de *D* como:

$$D = max(min(Dq + Du + Dt, 100), 0)$$

Si el dispositivo es del tipo portátil (*deviceType* = "handheld") se debe hacer un ajuste con el resultado final como:

$$\widehat{MOS}_{handheld} = htv1 + htv2 * \widehat{MOS} + htv3 * \widehat{MOS}^2 + htv4 * \widehat{MOS}^3$$
$$\widehat{MOS} = max(min(\widehat{MOS}_{handheld}, 5), 1)$$

Donde htv1= −0.60293, htv2 = 2.12382, htv3 = −0.36936, htv4= 0.03409 son coeficientes de ajuste prefijados.

2.3.4.8 Modos de funcionamiento en el módulo Pv

Ahora vamos a ver en profundidad cómo afecta cada modo de funcionamiento al módulo de Pv.

2.3.4.8.1 Modo 0

En este modo 0 se ve afectado el cálculo de la variable *quant*, entrada del módulo de cuantificación.

$$quant = a_1 + a_2 * ln(a_3 + ln(br) + ln(br * bpp + a_4))$$

bpp es el número de Kbits por píxel, se calcula como:

$$bpp = \frac{br}{codRes * fr}$$

Donde a_1= 11.99835, a_2 = −2.99992, a_3 = 41.24751, a_4= 0.13183 son coeficientes de ajuste prefijados.

En el caso de que el formato de transporte este segmentado y el Bitrate no esté especificado este se obtendrá usando el tamaño del chunk, como:

$$br = brChunkSize = \frac{chunkSize * 8 - audioSize - tsHeader - pesHeader}{videoDur * 1000}$$

- *brChunkSize*: Estimación del Bitrate del video codificado en Kbits/s
- *chunkSize*: Tamaño del chunk de video y audio juntos en Bytes
- *videoDur*: Duración del chunk de video en segundos

audioSize se calcula como:

$$audioSize = audioBrTarget * audioDur * 1000$$

- *audioBrTarget*: Bitrate objetivo del audio codificado en Kbits/s
- *audioDur*: Duración del audio del chunk en segundos

tsHeader se calcula como:

$$tsHeader = 4 * 8 * \frac{chunkSize}{188}$$

pesHeader se calcula como:

$$pesHeader = 17 * 8 * (numVideoFrames + numAudioFrames)$$

- *numVideoFrames*: Número de muestras de video en el chunk del video
- *numAudioFrames*: Número de muestras de video y audio en el chunk del video

Y se calculan así:

$$numVideoFrames = int(ceil(videoDur * framerate))$$

$$numAudioFrames = int(ceil\,(audioDur * \frac{audioSampleRate}{samplesPerFrame}))$$

- *audioSampleRate*: Ratio de Hz por muestra de audio en Hz
- *samplesPerFrame*: Número de muestras por cada fragmento de audio

2.3.4.8.2 Modo 1

En este modo 1, el cálculo de *Dq* se ve mejorado por la información extra obtenida respecto a la complejidad de codificación del video, que afecta a la efectividad de la codificación. Las modificaciones para ajustar esta mejora se producen en la variable *quant* y en las ecuaciones dentro del quantization module, a diferencia de en los modos 0, 2 y 3 que solo afectan a la variable *quant*.

La variable *quant* se calcula como:

$$quant = a_1 + a_2 * ln(a_3 + ln(br) + ln(br * bpp + a_4))$$

Donde cambian a_1= 5.00012, a_2 = −1.19631, a_3 = 41.35850 como coeficientes de ajuste prefijados.

bpp cambia a:

$$bpp = \frac{brFrameSize}{codRes * fr}$$

- *brFrameSize*: Bitrate del video calculado desde el tamaño de las imágenes, en Kbits/s

brFrameSize se calcula como:

$$brFrameSize = \frac{\sum_{index=1}^{numVideoFrame} frameSizes(index) * 8}{frameDuration * numVideoFrames * 1000}$$

- *frameSizes*: Vector que contiene el tamaño de la imagen en bytes, para cada imagen del chunk
- *frameDuration*: Duración de una imagen en segundos
- *numVideoFrame*: Número de imágenes de video en un chunk

Respecto a *Dq* se calcula de la misma manera que lo explicado en 2.3.4.4, pero se ajusta el resultado $\widehat{MOS}$ obtenido de la siguiente forma:

$$\widehat{MOS} = \widehat{MOS}_{mode0} + sigmoid(k0, k1, k2, iFrameRatio)$$

$sigmoid(k0, k1, k2, iFrameRatio)$ se calcula como:

$$sigmoid(k0, k1, k2, iFrameRatio) = k0 - \frac{k0}{1 + e^{-scalex*(iFrameRatio-midx)}}$$

scalex se calcula como:

$$scalex = \frac{10}{k2 - k1}$$

midx se calcula como:

$$midx = \frac{k1 + k2}{2}$$

Donde k_0= −0.91562479, k_1 = −3.28579526, k_2 = 20.4098663 son coeficientes de ajuste prefijados.

iFrameRatio se calcula como:

$$iFrameRatio = \frac{\underline{I}}{\underline{I_n}}$$

- $\underline{I}$: Número medio de imágenes-I en un chunk
- $\underline{I_n}$: Número medio de imágenes-no-I en un chunk

Siendo también un máximo tal que:

$$(\widehat{MOSq}) = max\,(min(\,(\widehat{MOSq}), 5)\,, 1)$$

2.3.4.8.3 Modo 2

En este modo 2 se utiliza un 2% de los datos (refiriéndose a un 2% de la secuencia de bits en el payload del protocolo H.264) del video para mejorar el resultado del modelo.

Puede ocurrir el caso de que no se pueda obtener esta información extra de una imagen (imagen corrupta, error de codificación...), para ello el modelo presenta la compensación de imagen, la cual es usada después de analizar el bitstream de datos y cuya función es calcular la variable *quant* de forma más precisa.

Primero se analizan todas las imágenes, chunk por chunk, analizando si los datos estadísticos están presentes o no y el tipo de datos en cada caso. Presentes: "I", "P" o "B"; no presentes: "I" o "Non-I".

Una vez tengamos la clasificación se pueden dar 3 casos: Que todas las imágenes tengan la información buscada, que ninguna la tenga, o que la tengan algunas.

En el caso de que todas las imágenes tengan la información buscada se aplica directamente el modo 2 o 3, para calcular la variable *quant*.

Si ninguna imagen tiene la información se aplica el modo 1 para calcular la variable *quant*.

En el caso de que algunos tengan información se utiliza el algoritmo de compensación de imagen ("Frame Compensation"), este algoritmo funciona buscando las imágenes sueltas o consecutivas del vector sin estadísticas, y asignándoles las estadísticas más cercanas en proximidad dentro del vector, preferiblemente imágenes del mismo tipo.

La variable *quant* en el modo 2 se calcula como:

$$quant = \frac{\underline{QP_{PB}}}{51}$$

- $\underline{QP_{PB}}$: Es la media de valores del vector *qppb* con los parámetros de cuantificación de las imágenes (*qpValues*) en un chunk, el cual lo podemos encontrar dentro del paquete *"itu-p1203"*, en el script *"p1203Pv.py"*.

- *qpValues*: $QP \in [0, 51]$

```
types = [ ]
qp_values = [ ]
for frame in frames:
        qp_values.append(frame["qpValues"])
        frame_type = frame["type"]
        types.append(frame_type)

qppb = [ ]
for index, frame_type in enumerate(types):
            if frame_type in ["P", "B", "Non-I"]:
                    qppb.extend(qp_values[index])
avg_qp = np.mean(qppb)

quant = avg_qp / 51.0
```

2.3.4.8.4 Modo 3

El modo 3 es parecido al modo 2 excepto que utiliza todos los datos posibles del video para mejorar el resultado del modelo.

La variable *quant* se calcula como en el modelo 2, lo que cambia es el código con el que se obtiene *qppb*.

```
types = [ ]
qp_values = [ ]
for frame in frames:
            qp_values.append(frame["qpValues"])
        frame_type = frame["type"]
            types.append(frame_type)

qppb = [ ]
for index, frame_type in enumerate(types):
            if frame_type in ["P", "B", "Non-I"]:
                qppb.extend(qp_values[index])
            elif frame_type == "I" and len(qppb) > 0:
                if len(qppb) > 1:
                                # replace QP value of last P-frame before I frame with QP value of
previous P-frame if there
                                # are more than one stored P frames
                                qppb[-1] = qppb[-2]
                else:
                                # if there is only one stored P frame before I-frame, remove it
                                qppb = [ ]
avg_qp = np.mean(qppb)

quant = avg_qp / 51.0
```

(7)

2.3.5 Módulo de estimación de calidad de audio - Pa

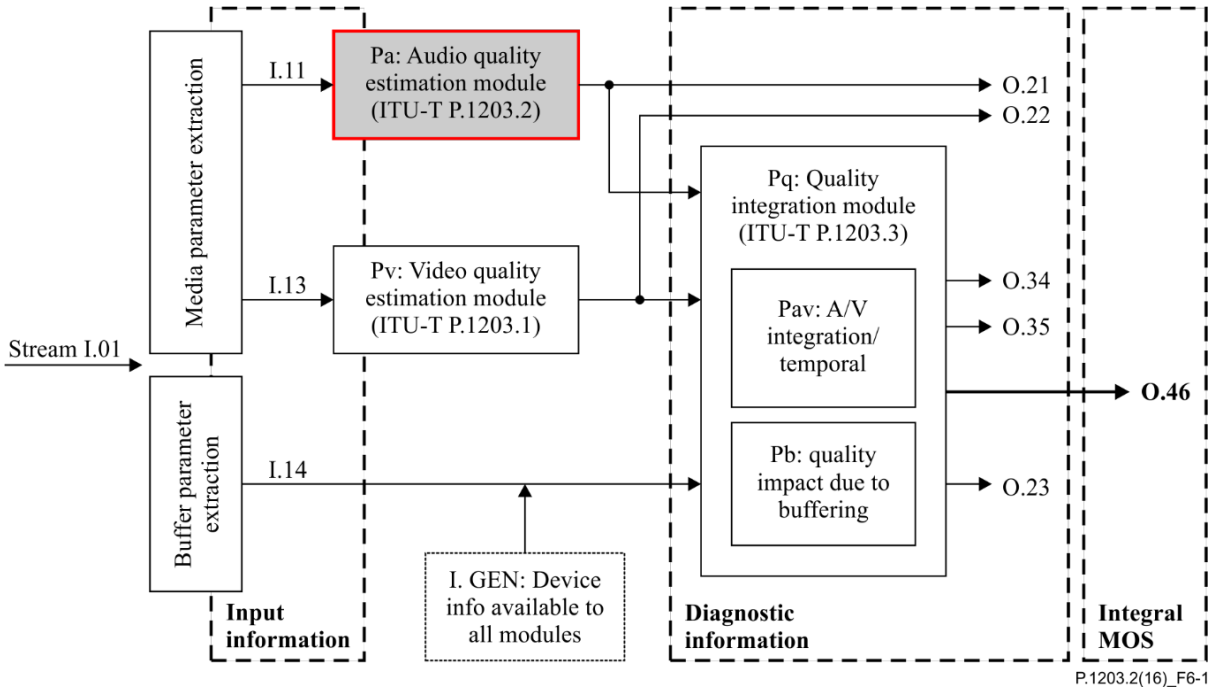

Ilustración 6 - Esquema del módulo Pa

Aquí vamos a ver en profundidad el módulo de cálculo de estimación de calidad de audio.

Este módulo está destinado a monitorizar los servicios de audio sobre TCP, tanto en servicios que funcionan en el más alto nivel (OTT - over the top), por ejemplo, YouTube, como en los servicios de video/audio gestionados por tcp utilizando los protocolos HTTP/TCP/IP y RTMP/TCP/IP.

Este módulo ha sido validado para los siguientes factores:

- Longitud del audio: Máximo 20 segundos
- Contenedor del Bitstream: Formato MPEG-2 para la Trama de Transporte de los segmentos de audio codificado
- Implementación del Codificador/Descodificador: El modelo ha sido ajustado usando AAC-LC: librería libfdk_aac, en modo de baja complejidad LC (ffmpeg), a partir de estos ajustes se ha creado un marco común para los demás códecs.
- Muestras por segundo del audio: 48,000 muestras/s
- Ratio de bits del audio: 16,32,64 y 98 Kbit/s/canal
- Canales de audio: 2 (estéreo)
- Duración de segmento: Entre 1 y 9 segundos, determina cada cuanto se puede cambiar entre distintas calidades

2.3.5.1 Datos de entrada

La entrada de datos para este módulo, el cual no varía con el modo de funcionamiento, está en la tabla 2 en el apartado I.11, como se puede apreciar en el esquema anterior.

2.3.5.2 Datos de salida

Como salida de datos este módulo solo tiene una salida: O.21 explicado anteriormente en el apartado 2.3.2.

2.3.5.3 Funcionamiento

Este módulo de calidad de audio funciona de la siguiente manera:

La salida con el resultado lo obtenemos de:

$$O.21 = MOSfromR(QA)$$

Donde *QA* se calcula como:

$$QA = 100 - QcodA$$

QcodA se calcula como:

$$QcodA = a1A * exp(a2A * Bitrate) + a3A$$

La función $MOSfromR$ ya ha sido explicada en el apartado 2.3.4.4.

Los coeficientes a1A, a2A y a3A varían en función de los códecs de audio utilizados.

Códec de Audio	a1A	a2A	a3A
MPEG1 L2	100.0	–0.02	15.48
AC3	100.0	–0.03	15.70
AAC-LC	100.0	–0.05	14.60
HE-AAC v2	100.0	–0.11	20.06

Tabla 3 - Coeficientes de códecs de audio

(8)

2.3.6 Módulo de calidad de integración - Pa

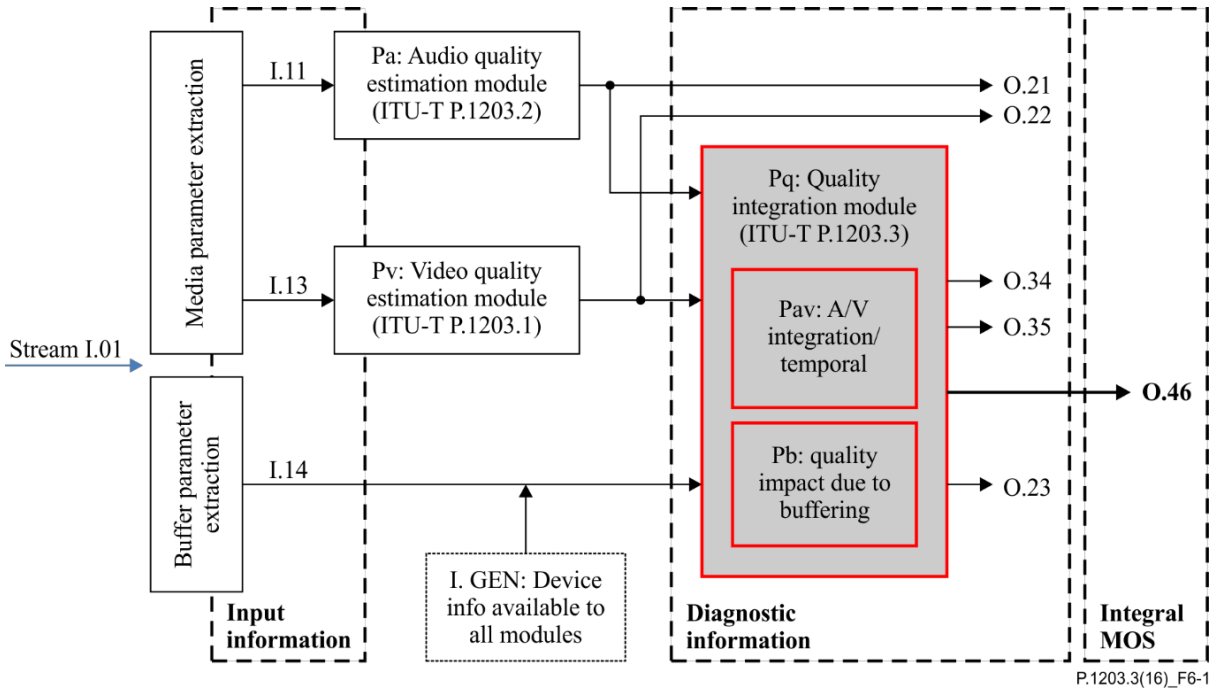

Ilustración 7 - Esquema del módulo Pq

Aquí vamos a ver en profundidad el módulo de cálculo de calidad de integración.

Este es el módulo final que tiene como objetivo calcular las puntuaciones finales tanto con los resultados obtenidos en los módulos anteriores, como con otros valores en la misma escala obtenidos por otros posibles modelos. Las puntuaciones finales son los datos de salida de este módulo, los cuales veremos más adelante.

Este módulo ha sido validado para los siguientes factores:

- Longitud de la secuencia de video: Entre 60 segundos y 5 minutos
- Duración de la carga inicial: Entre 0 y 10 segundos
- Número máximo de eventos de parada: 5
- Duración máxima de un evento de parada: 15 segundos
- Duración máxima de todos los eventos de parada: 30 segundos
- Otros detalles: Sin eventos de parada en los 5 primeros segundos al inicio del video

2.3.6.1 Datos de entrada

La entrada de datos para este módulo, el cual no varía con el modo de funcionamiento, son: O.21 del módulo de estimación de calidad de audio, O.22 del módulo de estimación de calidad de video y las variables de la tabla 2 en el apartado I.14, como se puede apreciar en el esquema anterior.

2.3.6.2 Datos de salida

Como salida de datos este módulo tenemos: O.23, O.34, O.35 y O.46 explicado anteriormente en el apartado 2.3.2.

2.3.6.3 Módulo de parada

Primero se llama a una rutina para calcular los siguientes parámetros relacionados con los eventos de parada:

- *numStalls*: Número de eventos de parada.
- *totalStallLen*: Importancia total de los eventos de parada ponderados.
- *avgStallInterval*: Tiempo medio entre dos eventos de parada.

numStalls lo obtenemos a partir de los datos de I.14.

totalStallLen para obtener este valor total, primero hay que asignar un valor de importancia a cada parada w_satll_i dependiendo del momento en el que se produce la parada respecto a la duración del video

w_satll_i se calcula como:

$$w\_satll_i = c_{ref7} + (1 - c_{ref7}) * e^{-(stallPositionFromEnd_i * (\frac{log(0.5)}{-c_{ref8}}))}$$

$stallPositionFromEnd_i$ se calcula como:

$$stallPositionFromEnd_i = T - startTime_i$$

T es la duración del clip y $startTime_i$es el tiempo de inicio del *i* evento de parada

totalStallLen se calcula como:

$$totalStallLen = \sum_{i=1}^{numStalls} w_{stall_i} * stallLen_i$$

Donde c_{ref7}= 0.48412879 y c_{ref8} = 10 son coeficientes de ajuste prefijados.

avgStallInterval indica el tiempo medio entre 2 eventos de parada.

Se calcula con el tiempo entre el inicio de dos eventos de parada y sumándole la duración de la parada del primer evento, como se puede apreciar en la siguiente imagen.

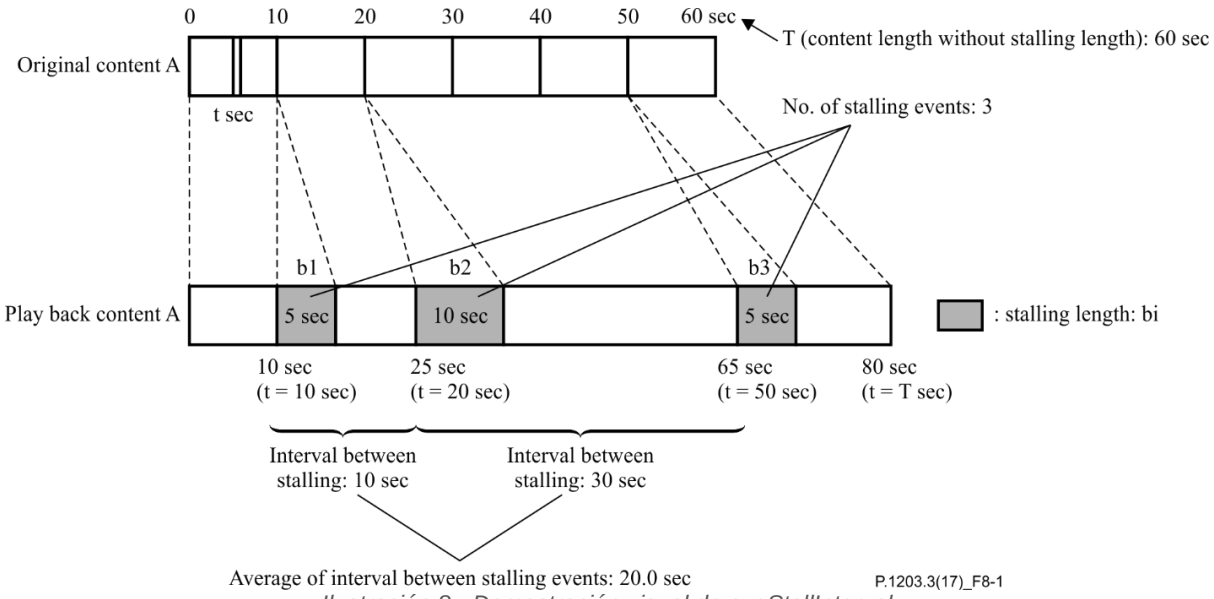

Ilustración 8 - Demostración visual de avgStallInterval

2.3.6.3.1 Salida O.23

Percepción de parada, obtenida a partir del módulo de parada como:

$$O.23 = 1 + 4 * SI$$

Donde *SI* se calcula como:

$$SI = exp(-\frac{numStall}{s1}) * exp(-\frac{\frac{totalStallLen}{T}}{s2}) * exp(-\frac{\frac{avgStallInterval}{T}}{s3})$$

Donde $s1$= 9.35158684, $s2$ = 0.91890815 y $s3$= 11.0567558 son coeficientes de ajuste prefijados.

2.6.3.4 Módulo de calidad audiovisual

La siguiente rutina se centra en los siguientes parámetros de calidad audiovisual:

- *negativeBias*: Es el percentil 10 de la diferencia por segundo entre el valor de O.34 y O.35 ($O.34_{diff}$), si este valor es negativo, sino es 0.
- *vidQualSpread*: Es la diferencia entre la calidad máxima y mínima de video en O.22
- vidQualChangeRate: Es la cantidad de veces que la diferencia entre un valor de O.22 y el valor anterior de O.22 es mayor de 0.2, dividido entre la longitud del video.
- *qDirChangesTot*: Indica la cantidad de cambios de dirección de calidad en el video.
- *qDirChangesLongest*: Es la duración en segundos del periodo más largo del vídeo sin cambios de calidad.

negativeBias se calcula como:

$$negativeBias = max(0, -negPerc) * c23$$

negPerc es el percentil 10 de $O.34_{diff}$

$O.34_{diff}$ se calcula como:

$$O.34_{diff}[t] = (O.34[t] - O.35_{baseline}) * w_{diff}[t]$$

w_{diff} se calcula como:

$$w_{diff}[t] = c1 + (1 - c1) * e^{-((T-t)*\frac{log(0.5)}{-c2})}$$

T es la duración total del clip y *t* es la posición temporal.

Donde $c1$ = 1.87403625, $c2$ = 7.85416481 y $c23$= 0.01853820 son coeficientes de ajuste prefijados.

Veremos cómo se calcula $O.35_{baseline}$ más adelante en el apartado de Salida O.35.

vidQualSpread se calcula como:

$$vidQualSpread = max(O.22) - min(O.22)$$

vidQualChangeRate se calcula como podemos ver en el siguiente pseudocódigo:

```
for i in range(1, duration):
    diff = O.22[i] - O.22[i-1]
if diff > 0.2 or diff < -0.2:
        vidQualChangeRate += 1
vidQualChangeRate = vidQualChangeRate / duration
```

qDirChangesTot se calcula creando una lista con todos los cambias de calidad por segmento de O.22 (*QC*) y a partir de ella obtenemos los cambios de dirección. Primero tenemos que ampliar la lista de resultados de O.22 con márgenes al principio y al final de esta, estos márgenes tendrán el primer y último valor de O.22 repetido varias veces (4) respectivamente, de forma que podamos ir comparando valores entre distintos puntos del video, con una separación de 3 escalones entre los 2 puntos. Para guardar las comparaciones necesitamos una escala de 5 niveles empezando en el nivel intermedio como 0 y apuntando en *QC* si se produce un cambio significativo o no, como: +1 si aumenta la calidad, -1 si disminuye o 0 si no varía.

El siguiente código muestra el algoritmo:

```
QC = [ ]

ma_order = 5
ma_kernel = np.ones(ma_order) / ma_order
padding_beg = np.asarray([self.O22[0]] * (ma_order - 1))
padding_end = np.asarray([self.O22[-1]] * (ma_order - 1))
padded_O22 = np.append(np.append(padding_beg, self.O22), padding_end)
ma_filtered = signal.convolve(padded_O22, ma_kernel, mode='valid').tolist()

step = 3
for current_score, next_score in zip(ma_filtered[0::step],
ma_filtered[step::step]):
thresh = 0.2
if (next_score - current_score) > thresh:
        QC.append(1)
elif -thresh < (next_score - current_score) < thresh:
        QC.append(0)
else:
        QC.append(-1)
```

Aún falta revisar la lista *QC* eliminando los 0 y juntando las regiones con el mismo valor, pues no muestran un cambio de dirección. El algoritmo para ello se encuentra junto al algoritmo para calcular *qDirChangesLongest* y lo veremos a continuación.

qDirChangesLongest funciona recorriendo *QC*, creando una lista con la posición en la lista donde empieza un cambio y su valor positivo (+1) o negativo (-1), con forma de dupla [posición de inicio] [positivo/negativo]. A continuación, se le añade al principio y al final de la lista una dupla con cambio 0 y se calcula las distancias entre cada posición de la lista. El valor deseado es la distancia más larga obtenida multiplicado por el escalón de separación (cuyo valor es 3) entre las diferencias de valores de O.22.

El siguiente código muestra el algoritmo:

```
lens = [ ]
for index, val in enumerate(QC):
if val != 0:
        if lens and lens[-1][1] != val:
        lens.append([index, val])
        if not lens:
        lens.append([index, val])
if lens:
lens.insert(0, [0, 0])
lens.append([len(QC), 0])
distances = [b[0] - a[0] for a, b in zip(lens, lens[1:])]
longest_period = max(distances) * step
else:
longest_period = len(QC) * step
q_dir_changes_longest = longest_period

q_dir_changes_tot = sum(1 for k, g in groupby([s for s in QC if s != 0]))
```

2.6.3.4.1 Salida O.34

Calidad audiovisual por intervalo de muestras, lo obtenemos a partir de los valores de O.21 y O.22, calidad de audio y video por intervalo de muestras respectivamente.

Se calcula como:

$$O34_t = max(min(av_1 + av_2 * O21_t + av_3 * O22_t + av_4 * O21_t * O22_t, 5), 1)$$

Donde av_1 = –0.00069084, av_2 = 0.15374283, av_3= 0.97153861 y av_4= 0.02461776 son coeficientes de ajuste prefijados, y t indica indica la numeración de la muestra.

2.6.3.4.2 Salida O.35

Salida con la calidad audiovisual final de codificación, toma como datos O.34 y datos temporales del video.

O.35 se calcula como:

$$O.35 = O.35_{baseline} - negBias - oscComp - adaptComp$$

$O.35_{baseline}$se calcula como:

$$O.35_{baseline} = \frac{\sum_t w1(t)*w2(t)*O.344[t]}{\sum_t w1(t)*w2(t)}$$

$$w1(t) = t_1 + t_2 * e^{t/(T/t_3)}$$
$$w2(t) = t\,4 - t\,5 * O.34[t]$$

Donde t_1 = 0.00666620027943848, t_2 =0.0000404018840273729, t_3= 0.156497800436237, t_4= 0.143179744942738 y t_5= 0.0238641564518876 son coeficientes de ajuste prefijados.

negativeBias ya ha sido explicada previamente en el apartado anterior 2.6.3.4.

oscComp depende de la condición de *oscTest*, si esta se cumple:

$$oscComp = max\,(0, min(qDiff * e^{c1*qDirChangesTot+c2}, 1.5))$$

$$qDiff = max(0, 1 + log\ 10\,(vidQualSpread + 0.001))$$

Donde c_1 = 0.67756080 y c_2 = −8.05533303, son coeficientes de ajuste prefijados.

En el caso de que no se cumpla *oscComp* vale 0.
La condición *oscTest* se calcula como:

$$oscTest = (\frac{qDirChangesTot}{duration} < 0.25) \text{ and } (qDirChangeLongest < 30)$$

addaptComp depende de la condición de adaptTest, si esta se cumple:

$$adaptComp = max(0, min(c3 * vidQualSpread * vidQualChangeRate + c4, 0.5))$$

En el caso de que no se cumpla $adaptComp$ vale 0.

La condición adaptTest se calcula como:

$$adaptTest = (\frac{qDirChangesTot}{duration} < 0.25)$$

Donde c_3= 0.17332553 y c_4 = −0.01035647, son coeficientes de ajuste prefijados.

2.6.3.4.3 Salida O.46

Calidad final de sesión, toma como datos: los datos del módulo de parada, O.35 del módulo de calidad, los resultados del modelo predictivo de machine learning Random Forest y la duración del video.

Primero se calcula un valor temporal de O.46 como:

$$O.46\_temp = 0.75 * (1 + (O.35 - 1) * SI) + 0.25 * RFPrediction$$

SI ya ha sido calculado previamente en el apartado anterior 2.3.6.3.1.

RFPrediction es el resultado del algoritmo Random Forest.

Por último, para ajustar las diferencias entre los distintos laboratorios y sus respectivos test se ha realizado una regresión lineal de primer orden entre el valor subjetivo MOS y la salida O.46 de cada laboratorio, y a continuación se ha realizado la media de los resultados obtenidos, llegando a unos coeficientes que ajustan el valor temporal como:

$$O.46 = 0.02833052 + 0.98117059 * O.46\_temp$$

Anexo - Modelo de predicción con ML - Random Forest

Para el cálculo de estimación del MOS se utiliza un mecanismo de machine learning en el que 14 características, con su valor correspondiente, son introducidas a un árbol de decisión de 20 ramas y hasta 6 niveles de profundidad. El vector de características pasa por cada rama obteniendo una puntuación MOS en cada una de ellas, y la puntuación final es la media de los 20 resultados obtenidos.

El mecanismo de cada nodo es analizar una característica respecto a un valor prefijados, si el valor de la característica es mayor a un cierto margen se continua en una dirección, si es menor continúa en otra dirección, al llegar al siguiente nodo ocurre lo mismo, cada nodo analiza una característica distinta hasta llegar a la hoja final, obteniendo un resultado MOS.

Un nodo está formado por 5 valores: ID del nodo, ID de la característica, margen a superar, ID del siguiente nodo si no supera el margen e ID del siguiente nodo si lo supera.

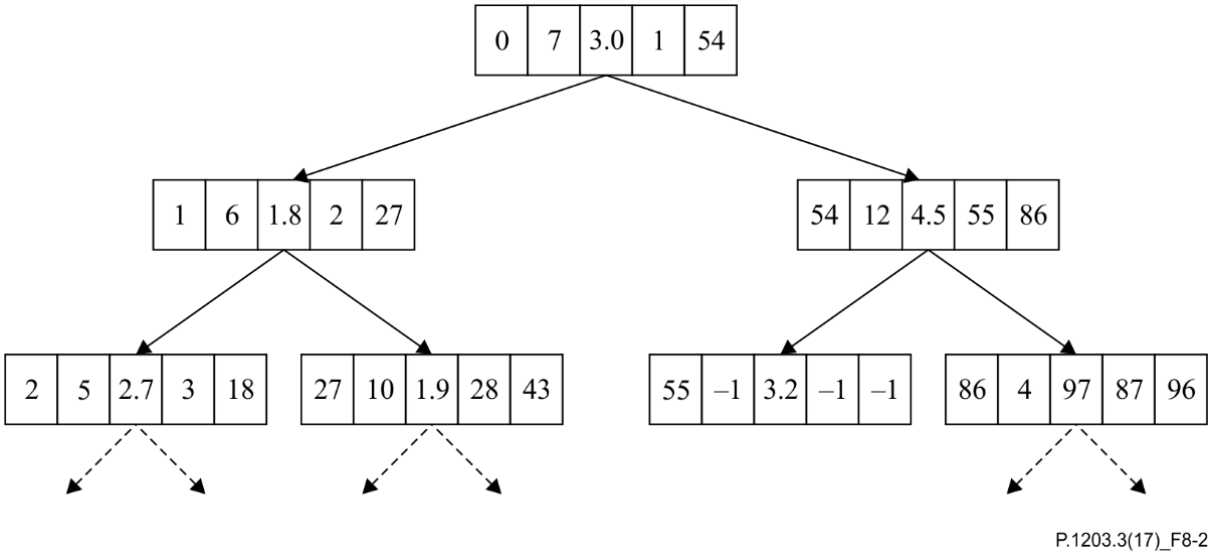

Ilustración 9 - Esquema de nodos Random Forest

Características para el modelo de predicción, en orden:

- *stallCountWithoutInitial:* Número de eventos de parada sin contar la carga inicial.
- *stallDur:* Duración total de todos los eventos de parada, la carga inicial cuenta como un tercio de su duración.
- *stallFreq:* Frecuencia de eventos de parada, número de eventos de parada, sin contar la carga inicial, entre la duración total del video.
- *stallRatio:* Ratio de la duración de parada, se divide *stallDur* entre la duración del video.
- *timeLastStallToEnd:* Tiempo transcurrido entre el inicio del último evento de parada y el final del video, en caso de no haber ningún evento de parada su valor es T.
- *averagePvScoreOne:* Media del primer tercio de valores de O.22.
- *averagePvScoreTwo*: Media del segundo tercio de valores de O.22.
- *averagePvScoreThree:* Media del tercer tercio de valores de O.22.
- *1PercentilePvScore:* Percentil 1 de O.22.
- *5PercentilePvScore:* Percentil 5 de O.22.
- *10PercentilePvScore:* Percentil 10 de O.22.
- *averagePaScoreOne:* Media de la primera mitad de valores de O.21.
- *averagePaScoreTwo:* Media de la segunda mitad de valores de O.21.
- *T:* Duración del video.

(9)

Capítulo 3 - Software de evaluación.

En este capítulo vamos a ver el software utilizado para el desarrollo del proyecto, además de las librerías adicionales necesarias para su correcto funcionamiento.

3.1 Python

Para el desarrollo de este proyecto he decidido utilizar Python por ser un lenguaje de programación de alto nivel, orientado a objetos, de propósito general. Está diseñado para que el código sea legible por humanos, por lo que utiliza notablemente espacios en blanco para que los programadores puedan tener una estructura clara y lógica tanto como para proyectos pequeños como grandes. (10) (11)

Ilustración 10 - Logotipo Python

3.2 Librería itu_p1203

Esta es la librería principal utilizada para el cálculo de los resultados, a continuación, la analizo en detalle pues es la responsable de implementar las indicaciones del ITU-T P.1203 explicadas en el capítulo 2. (12) (13) (14)

3.2.1 Estructura

Trees	Carpeta con archivos de datos ".csv" necesarios para poder utilizar el algoritmo Random Forest
__init__.py	Archivo de inicialización de Python.
__main__.py	Script de entrada en caso de ejecutar el paquete, realiza el procedimiento usando los demás scripts.
errors.py	Define la clase P1203StandaloneError ayudándose del Script log.py
extractor.py	Crea un informe a partir de un archivo de video con la información necesaria para el paquete P.1203.
log.py	Crea un registro con un cierto nombre y un nivel (debug o info).
measurementwindow.py	Implementa una ventana de medición deslizante para ir calculando puntuaciones por tramos.
p1203Pa.py	Script encargado del módulo que calcula las calificaciones de audio.
p1203Pv.py	Script encargado del módulo que calcula las calificaciones de video.

p1203Pq.py	Script encargado del módulo de integración que calcula los resultados finales.
p1203_standalone.py	Script encargado de implementar los distintos scripts en conjunto para poder obtener un resultado final a partir de un archivo JSON de entrada.
rfmodel.py	Script encargado de implementar el algoritmo Random Forest en el cálculo de los resultados.
utils.py	Script con funciones auxiliares utilizadas por los demás scripts.

Tabla 4 - Tabla de contenidos de la librería itu_P1203

3.2.2 Disección del código

3.2.2.1 __init__.py

Este archivo es necesario para que Python trate las carpetas con scripts como paquetes, podría estar vacío, pero en nuestro caso contiene la versión del paquete, e importa los scripts más importantes: P1203Pa, P1203Pv, P1203Pq y P1203Standalone.

3.2.2.2 __main__.py

Compuesto por 2 funciones principales y algunas secundarias.

Las funciones principales son:

- **extract_from_single_file**

Argumentos:

input_file {str} -- archivo de entrada (JSON o archivo de video, o STDIN si “-”)
mode {int} -- 0, 1, 2, 3 elección del modo de actuación del modelo
debug {bool} – Ejecuta el modo debug, por defecto: False.
only_pa {bool} -- solo ejecutar el módulo de audio, por defecto: False.
only_pv {bool} -- solo ejecutar el módulo de video, por defecto: False.
print_intermediate {bool} -- mostrar los valores intermedios O.21/O.22
modules {dict} -- puedes especificar otros módulos de Pa, Pv y Pq distintos de los predeterminados
quiet {bool} -- Silenciar los mensajes del registro, por defecto: False.
amendment_1_audiovisual {bool} -- habilitar la corrección 1 de la cláusula 8.2, por defecto: False.
amendment_1_stalling {bool} -- habilitar la corrección 1 de la cláusula 8.4, por defecto: False.

Extrae el informe con los datos a partir de un solo archivo (JSON o video) usando utils.py para los JSON, o usando el extracort.py para obtener el JSON del archivo de video. Luego crea una variable de la clase P1203Standalone (*itu_p1203)* con esos datos *,* la cual los distintos módulos usan para calcular sus resultados.

Esta función devuelve el archivo de entrada, y la variable *output* con los resultados de los distintos módulos.

- **main**

Argumentos:
modules {dict} -- puedes especificar otros módulos de Pa, Pv y Pq distintos de los predeterminados
quiet {bool} -- Silenciar los mensajes del registro

Esta función permite ejecutar el Script p1203_standalone.py desde la línea de comandos, funciona usando múltiples parser para todas las posibilidades que vimos en la función principal anterior.

Esta función devuelve un print() con el JSON *output_report* que contiene los resultados de los distintos módulos.

Las funciones secundarias se encargan de asegurarse de que el usuario ha firmado que comprende cuales son los autores y el uso de copyright del paquete.

3.2.2.3 errors.py

Sirve para obtener un registro global de los errores. Para ello define una clase que maneja las excepciones obteniendo el error en el registro.

```
logger = log.setup_custom_logger('main')#Registro global

class P1203StandaloneError(Exception):#Manejador de excepciones
        def __init__(self, message):
        logger.error(message)
        super().__init__(message)
```

3.2.2.4 Extractor.py

Compuesto por la clase Extractor con sus funciones internas, la función main y otras funciones auxiliares.

- **Class Extractor(Object)**

Extrae la información relevante del video para luego introducir los datos en el modelo, está basado en los paquetes ffmpeg y ffprobe.

Funciones internas:

- **__init__**

Argumentos:
self{Extractor}
input_files {list} -- lista con los videos a analizar

mode {int} -- el modo de funcionamiento del modelo, para saber la información necesaria a extraer.

qp_logfile {NoneType} -- Añade un registro al archivo, solo disponible si es un único video

use_average{bool} -- Se decide si se utiliza una media con los valores qp de los frames, por defecto: False.

Inicializa un nuevo objeto (self) de la clase Extractor, con las características seleccionadas.

- **extract**

Argumentos:

self{Extractor}

Función principal de la clase Extractor. Empieza creando dos listas para la información de los segmentos de video y audio de forma separada, a continuación, itera por la lista de videos y se aplica la función *get_segment_info_lines*, que veremos próximamente en este apartado, y obtenemos 3 valores: *segment_info_video, segment_info_audio* y la duración de cada video. Esto dato se van añadiendo a las listas previamente creadas y junto a datos predeterminados del método de visualización (IGEN) se crea un informe.

Este informe se añade a la clase (*self.report*) y también es lo que devuelve la función *extract(self)*.

- **get_tempfilename()**

Sin argumentos.

Método que devuelve un nombre de archivo temporal.

- **_file_line_gen**

Argumentos:

filename {NoneType/str} -- Nombre completo del archivo

Método para identificar si la extensión del archivo es .gz y abrirlo con el método *open* del paquete *gzip* (*gzip.open*) o abrirlo con el método *open* directamente y devolver con *yield* una iteración de las líneas de texto del archivo argumento.

- **_parse_qp_data**

Argumentos:

logfile {NoneType/str} -- Variable con el registro del archivo de video

use_average{bool} -- Se decide si se utiliza una media con los valores qp de los frames, por defecto: False.

Este método es una forma eficiente de traspasar los datos de cada frame. Empieza utilizando la función anterior *_file_line_gen* para obtener el texto del registro, decodificar desde utf-8 y dejarlo mejor estructurado con:

```
for line in Extractor._file_line_gen(logfile):
    line = line.decode("utf-8").strip()
```

A partir de aquí se utilizan varias declaraciones *if* para ir removiendo las líneas no relevantes, hasta llegar a la línea con "*New frame*" de donde obtendremos toda la información necesaria como: Tipo de frame, tamaño del frame y los valores de qp.

Este método devuelve con *yield* la información de los "*New frame*" de forma iterable.

- **parse_qp_data**

Argumentos:

logfile {NoneType/str} -- Variable con el registro del archivo de video

use_average {bool} -- Se decide si se utiliza una media con los valores qp de los frames, por defecto: False.

Este método traspasa la información de los registros qp generados por el paquete ffmpeg-debug-qp utilizando la función anterior *_parse_qp_data* y lo devuelve en forma de list:

```
return list (Extractor.  parse  qp  data(logfile, use  average))
```

- **get_video_frame_info_ffmpeg_debug_qp**

Argumentos:

segment -- Archivo de video

qp_logfile {NoneType} -- Añade un registro al archivo, solo disponible si es un único video

use_average {bool} -- Se decide si se utiliza una media con los valores qp de los frames, por defecto: False.

Método que obtiene la información de un segmento(vídeo) usando el script "ffmpeg-debug-qp"

Primero intenta utilizar el método anterior "parse_qp_data", si no funciona intenta obtener la dirección del script, tanto en el directorio "ffmpeg-debug-qp" como en el directorio actual, y ejecutarlo con la dirección del video(*segment*).

Devuelve la lista de los datos ofrecidos por el script si ha podido ejecutarlo.

- **get_video_frame_info_ffprobe**

Argumentos:

segment -- Archivo de video

info_type {str} -- Indica si se trata de un paquete (*packet*) o de una imagen (*frame*), por defecto: packet.

Dependiendo si se trata de un paquete o de una imagen se ejecuta el comando de "ffprobe" apropiado para obtener: *frame_type (tipo de imagen: I/NO-I o I/P/B), DTS (marca de tiempo de decodificación), PTS (marca de tiempo de presentación), size (tamaño en bytes) y duration (duración en milisegundos).*

Esta información se guarda en un JSON.

```
stdout, _ = run_command(cmd)
info = json.loads(stdout)[info_type + "s"]
```

A continuación, se recorre el JSON y se va añadiendo cada paquete o imagen a un diccionario que recuerda el orden de entrada (*OrderedDict*) y cada uno de estos se guarda en una lista (*ret*).

El método devuelve la lista *ret.*

- **get_format_info**

Argumentos:

segment -- Archivo de video

Método de funcionamiento similar a *get_video_frame_info_ffprobe,* se ejecuta el comando de "ffprobe" con el argumento -*show_format* para obtener: *nb_streams* (número de elementos en el stream), *nb_programs* (identificación del códec), *format_name* (nombre del formato del archivo), *format_long_name* (nombre del formato del archivo sin abreviación)*, start_time* (posición de la primera imagen del compuesto en segundos fraccionados por *AV_TIME_BASE*), *duration* (duración del stream en segundos fraccionados por *AV_TIME_BASE*), *size* (tamaño del stream en bytes), *bit_rate (*ratio de bit/s en el total del stream) y *probe_score* (puntuación de sondeo del formato).

Estos datos se guardan en un JSON y posteriormente se traspasan a un diccionario (*info*), y esto es lo que devuelve el método.

- **get_segment_info**

Argumentos:

segment -- Archivo de video

Método de funcionamiento similar a *get_video_frame_info_ffprobe,* se ejecuta el comando de "ffprobe" con el argumento -*show_streams* para obtener: *segment_filename* (nombre del archivo), *file_size* (tamaño del archivo en bytes), *video_duration* (duración del video en milisegundos), *video_frame_rate* (ratio de imágenes en hercios), *video_width* (anchura en píxeles), *video_height* (altura en píxeles), *video_codec* (tipo de codificación del video: h264, HEVC o vp9), *audio_duration* (duración del audio en milisegundos) y *audio_bitrate* (ratio de Kbit/s del audio).

Esta información se guarda en un JSON el cual se recorre y se separa entre audio y video para luego añadirse a un diccionario que recuerda el orden de entrada (*OrderedDict*) y cada uno de estos se guarda en una lista (*ret*).

El método devuelve la lista *ret.*

- **get_stream_size**

Argumentos:

segment -- Archivo de video

stream_type {str} -- Indica si se trata de "video" o "audio", por defecto: video.

Método que devuelve el tamaño del stream en bytes, sumando el tamaño de cada imagen a través del script *ffprobe.*

- **get_segment_info_lines**

Argumentos:

segment -- Archivo de video

mode {int} -- 0, 1, 2, 3 elección del modo de actuación del modelo, por defecto: 0.
timestamp {int} -- Marca de tiempo de inicio para los segmentos, por defecto: 0.
qp_logfile {NoneType} -- Añade un registro al archivo, solo disponible si es un único video.
use_average {bool} -- Se decide si se utiliza una media con los valores qp de los frames, por defecto: False.

Este método utiliza las funciones *get_segment_info* y *get_format_info* para obtener la información del segmento y pasarlo a un JSON de video o audio respectivamente. A continuación, comprueba el modo de funcionamiento y según si es modo 1 o modo 2-3 utiliza las funciones *get_video_frame_info_ffprobe* o *get_video_frame_info_ffmpeg_debug_qp* para añadir información adicional al JSON de video.

Devuelve el JSON de video y el de audio con su información correspondiente y la duración como valor independiente.

Función main:

- **main()**

Sin argumentos.

Crea un informe con la información necesaria para el paquete P.1203 tomando como entrada de datos los archivos de video y usando la clase Extractor, al finalizar muestra el JSON con el informe.

3.2.2.5 Log.py

Compuesto por una única función y una variable global (*loggers {Dictionary}*) que guarda los registros.

- **setup_custom_logger**

Argumentos:
Name {str} -- Nombre del registro
Debug {bool} -- Selecciona si se ejecuta como debug (*True*) o como info (*False*) por defecto: False.

Devuelve un registro (*logger*) con un cierto nombre y un nivel (debug o info), para ello hace uso del paquete logging.

3.2.2.6 Mesurementwindow.py

Compuesto por la clase Measurement Window y sus funciones internas.

- **Class MeasurementWindow()**

Implementa una ventana de medición deslizante que solo puede mantener cierta cantidad de imágenes dependiendo de la suma de la duración de estas.

Funciones internas:

- **__init__**

Argumentos:
self {MeasurementWindow}

Inicializa las variables necesarias para la clase.

- **set_score_callback**

Argumentos:
self
callback -- Argumento en el que se introduce una función y de forma asíncrona se realiza esa función y devuelve el resultado cuando termina.

Ajusta la función que debería ejecutarse para cierto modelo, cuando se necesita calcular una puntuación.

El *callback* toma dos argumentos:
{float} -- Marca de tiempo de la muestra a calcular
{list} -- Lista de las imágenes a las que calcular una puntuación

```
self._score_callback = callback
```

- **_should_calculate_score()**

Argumentos:
self

Este método se llama cada vez que se añade una nueva imagen a la *MesurementWindows* para decidir si hace falta calcular una puntuación de salida.

Se puede empezar a calcular la primera puntuación (t=1) cuando hay 11 frames añadidos a la *MesurementWindows*, y se calcula el siguiente cuando se añade otra imagen hasta terminar.

Se devuelve un boolean *True* si se puede calcular una puntuación o un *False* en caso contrario.

- **add_frame**

Argumentos:
self
frame {dict} -- Variable cuyas claves son la duración y la DTS (*marca de tiempo de decodificación*) de una imagen.

Añade un *frame* a la *MesurementWindows*, eliminando uno antiguo si se supera el máximo de 20 segundos de duración total y se llama al método self.*_should_calculate_score()*.

- **stream_finished**

Argumentos:
self

Función que se ejecuta cuando el flujo de imágenes acaba, disminuyendo frames hasta terminar la *MesurementWindows* y obteniendo una puntuación final.

3.2.2.7 P1203Pa.py

Compuesto por la clase P1203Pa y sus funciones internas.

- **Class P1203Pa(object)**

Contiene los datos de las puntuaciones de audio.

Funciones internas:

- **audio__model_function**

Argumentos:
self
codec {str} -- Indica la codificación del audio, debe ser uno de *mp2, ac3, aaclc* o *heaac*.
bitrate {int} -- Indica la ratio de Kbit/s usado en el audio.

Esta función empieza comprobando que el *codec* usado sea válido, continúa calculando el valor de *qa* valor que depende de unos parámetros fijos del códec usado y del bitrate del audio.

El valor *qa* se utiliza en la función *mos_from_r* del script *utils.py* y da como resultado el MOS del audio y se devuelve como *mos_audio*.

- **model_callback**

Argumentos:
self
output_sample_timestamp {int} -- Marca de tiempo de inicio para las muestras de salida (1, 2, ...).
frames {list} -- lista con las imágenes dentro de la ventana de medición.

Función que recibe frames de la ventana de medición y utilizando la función *get_chunk* del script *utils.py* se obtiene un conjunto de frames de la misma calidad y se calcula su calificación (*score*) utilizando la función previamente vista *audio__model_function*.

Al final añade el ese valor a la variable *o21.*

```
self.o21.append(score)
```

- **calculate**

Argumentos:
self

Esta función calcula el MOS (puntuación de opinión media) del audio.

Empieza utilizando la función *check_segment_continuity* del script *utils.py* la cual comprueba que los segmentos a analizar sean contiguos a los previamente analizados.

Continúa creando la variable *measurementwindow* a través de la clase *MeasurementWindow()* y ejecutando su método *set_score_callback*, luego genera 100 muestras de audio para cada segundo y crea un bucle que recorre estas muestras añadiéndolas a una variable *frame (dictionary)* con los siguientes valores ajustados: *duration, DTS, Bitrate* y *codec*, y posteriormente introduciéndolos en la variable *measurementwindow* mediante el método *add_frame(frame)* hasta terminar el bucle.

Al terminar el bucle se ejecuta la función *stream_finished()* de la clase *measurementwindow* y se devuelve el *stream_id* y la puntuación o21 en formato JSON.

- **__init__**

Argumentos:
self
segments -- {list} Lista de archivos de video.
stream_id -- {str} Identifica la procedencia de los datos, por defecto: None.

Inicializa los valores: *segments, stream_id* y o21, del modelo Pa, con los datos JSON de entrada.

3.2.2.8 P1203Pv.py

Compuesto por la clase P1203Pv y sus funciones internas.

- **Class P1203Pv(object)**

Contiene los datos de las puntuaciones de vídeo.

Funciones internas:

- **degradation_due_to_upscaling**

Argumentos:
self
coding_res {int} -- Indica el número de píxeles en la resolución codificada.
display_res {int} -- Indica el número de píxeles en la resolución visualizada.

Se quiere calcular la degradación de reescalado de visualización. Para ello se divide *display_res* entre *coding_res* y se obtiene el máximo con respecto a 1 para aplicarlo a una fórmula que utiliza coeficientes previamente definidos en la clase *P1203Pv.*

El resultado de la fórmula (*deg_scal_v*) es lo que devuelve la función.

- **degradation_due_to_frame_rate_reduction**

Argumentos:
self
deg_cod_v {int} -- Variable que indica la degradación de cuantificación.
deg_scal_v {int} -- Variable que indica la degradación de escalado.
framerate {float} -- Ratio de frames/s en el total del stream.

Se quiere calcular la degradación debida a la reducción de la ratio de frames/s. Para ello primero se comprueba si el valor de *framerate* es menor de 24, si este es el caso se aplica una fórmula para calcular *deg_frame_rate_v,* para finalizar se utiliza la función *constrain* del script *utils* para calcular el valor final de *deg_frame_rate_v* y devolverlo.

```
        deg_frame_rate_v = 0
        if framerate < 24:
        deg_frame_rate_v = (100 - deg_cod_v - deg_scal_v) * (t1 - t2 * framerate) / (t3 +
framerate)
        deg_frame_rate_v = utils.constrain(deg_frame_rate_v, 0.0, 100.0)
        return deg_frame_rate_v
```

- **degradation_integration**

Argumentos:

self

mos_cod_v {int/float} – MOS (Mean Opinion Score) de cuantificación calculado a partir de valores predeterminados y la variable *quant*.

deg_cod_v {int} -- Variable que indica la degradación de cuantificación.

deg_scal_v {int} -- Variable que indica la degradación de escalado.

deg_frame_rate_v {int} -- Variable que indica la degradación temporal de frames.

Función que integra las tres degradaciones, para ello primero calcula la degradación total (*deg_all*) utilizando la función *constrain* del script *utils* con la suma de las degradaciones como valor de entrada. Después se calcula la variable *qv* (codificación del video estimada) restando la degradación total a 100, y por último se devuelve el MOS final, resultado de la función *mos_from_r* del script *utils* que toma como entrada de datos *qv.*

- **video_model_function_mode0**

Argumentos:

self

coding_res {int} -- Indica el número de píxeles en la resolución codificada.

display_res {int} -- Indica el número de píxeles en la resolución visualizada.

bitrate_kbps_segment_size {float} -- Ratio de bits en Kbit/s en el total del segmento.

framerate {float} -- Ratio de frames/s en el total del stream.

Calcula la puntuación final del modelo, en el modo 0 a partir de los argumentos de esta función, además de algunos valores predeterminados ajustados al modo de funcionamiento. Para ello tiene que calcular las diferentes degradaciones.

Empieza con la degradación de compresión calculando la variable *quant*:

```
        quant = a1 + a2 * np.log (a3 + np.log(bitrate_kbps_segment_size) +
np.log(bitrate_kbps_segment_size * bitrate_kbps_segment_size / (coding_res * framerate) +
a4))
```

Continua con el cálculo del MOS de cuantificación (*mos_cod_v*) utilizando la variable *quant* y la función *constrain* del script *utils*:

```
mos_cod_v = q1 + q2 * np.exp (q3 * quant)
mos_cod_v = utils.constrain(mos_cod_v, 1.0, 5.0)
```

A partir de *mos_cod_v* calculamos la degradación de cuantificación (*deg_cod_v*) utilizando las funciones *r_from_mos* y *constrain* del script *utils*:

```
deg_cod_v = 100.0 - utils.r_from_mos(mos_cod_v)
deg_cod_v = utils.constrain(deg_cod_v, 0.0, 100.0)
```

Ahora calcula la degradación de escalado (*deg_scal_v*) y la de reducción de la ratio de frame (*deg_frame_rate_v*) utilizando las funciones previamente vistas *degradation_due_to_upscaling* y *degradation_due_to_frame_rate_reduction*.

```
deg_scal_v = self.degradation_due_to_upscaling(coding_res, display_res)
deg_frame_rate_v = self.degradation_due_to_frame_rate_reduction(deg_cod_v,
deg_scal_v, framerate)
```

Por último, calcula la puntuación final (*MOS*) utilizando la función anterior *degradation_integration* y se devuelve el resultado (*score*).

- **video_model_function_mode1**

Argumentos:

self
coding_res {int} -- Indica el número de píxeles en la resolución codificada.
display_res {int} -- Indica el número de píxeles en la resolución visualizada.
bitrate_kbps_segment_size {float} -- Ratio de bits en Kbit/s en el total del segmento.
framerate {float} -- Ratio de frames/s en el total del stream.
frames {list} -- lista con las imágenes dentro de la ventana de medición.
iframe_ratio {float} -- ratio de imágenes del tipo "I", usado en modo debug, por defecto: *None.*

Calcula la puntuación final del modelo, en el modo 1 a partir de los argumentos de esta función, además de algunos valores predeterminados ajustados al modo de funcionamiento. Para ello tiene que calcular las diferentes degradaciones.

Empieza con la degradación de compresión calculando la variable *quant*:

```
quant = a1 + a2 * np.log(a3 + np.log(bitrate_kbps_segment_size) +
np.log(bitrate_kbps_segment_size * bitrate_kbps_segment_size / (coding_res * framerate) +
a4))
```

Continua con el cálculo del MOS de cuantificación (*mos_cod_v*) utilizando la variable *quant* y la función *constrain* del script *utils*:

```
mos_cod_v = q1 + q2 * np.exp(q3 * quant)
mos_cod_v = utils.constrain(mos_cod_v, 1.0, 5.0)
```

Ahora se comprueba si tenemos un valor para *iframe_ratio* (modo debug) o si hay que calcularlo, esta es la parte que se diferencia del modo 0.

Si *iframe_ratio* no tiene valor, un bucle recorre la lista *frames* y la función *calculate_compensated_size* del script *utils* obtiene el tamaño del frame y añade ese tamaño a

la lista adecuada: una lista con los frames tipo “I” (*i_sizes*), o una lista con los demás tipos de frames (*noni_sizes*).

Si hay frames en las dos listas se calcula la ratio de frames “I” (*iframe_ratio*) dividiendo la media de tamaños de la lista *i_sizes* entre la media de tamaños de la lista, si no hay frames en las dos listas el valor de *iframe_ratio* es 0.

A continuación, se calcula el valor de complejidad (*complexity*) que se tiene que añadir al valor de *mos_cod_v* previamente calculado, (Aquí termina la parte que se diferencia del modo 0). Este valor se calcula aplicando la función *sigmoid* del script *utils* que utiliza como datos de entrada: valores predeterminado y el valor de *iframe_ratio*.

A partir de *mos_cod_v* calculamos la degradación de cuantificación (*deg_cod_v*) utilizando las funciones *r_from_mos* y *constrain* del script *utils*:

```
deg_cod_v = 100.0 - utils.r_from_mos(mos_cod_v)
deg_cod_v = utils.constrain(deg_cod_v, 0.0, 100.0)
```

Ahora calcula la degradación de escalado (*deg_scal_v*) y la de reducción de la ratio de frame (*deg_frame_rate_v*) utilizando las funciones previamente vistas *degradation_due_to_upscaling* y *degradation_due_to_frame_rate_reduction*.

```
deg_scal_v = self.degradation_due_to_upscaling(coding_res, display_res)
deg_frame_rate_v = self.degradation_due_to_frame_rate_reduction(deg_cod_v,
deg_scal_v, framerate)
```

Por último, calcula la puntuación final (*MOS*) utilizando la función anterior *degradation_integration* y se devuelve el resultado (*score*).

- **video_model_function_mode2**

Argumentos:

self
coding_res {int} -- Indica el número de píxeles en la resolución codificada.
display_res {int} -- Indica el número de píxeles en la resolución visualizada.
framerate {float} -- Ratio de frames/s en el total del stream.
frames {list} -- lista con las imágenes dentro de la ventana de medición.
quant {float} -- Parámetro que señala la degradación de cuantificación, usado en modo debug, por defecto: None.
avg_qp_per_noni_frame {list} -- Media de cuantificación para los frames no “I”, usado en modo debug, por defecto: [ ].

Calcula la puntuación final del modelo, en el modo 2 a partir de los argumentos de esta función, además de algunos valores predeterminados ajustados al modo de funcionamiento. Para ello tiene que calcular las diferentes degradaciones.

Empieza con la degradación de compresión, parte que se diferencia de los modos anteriores, comprobando si *quant* no tiene valor, si este es el caso, se comprueba que *avg_qp_per_noni_frame* tampoco lo tenga, si esto ocurre crea un bucle que recorre la lista *frames* para obtener los tipos de frame, y los valores qp asociados a los frames, y guarda cada dato en su array correspondiente (*types* y *qp_values*). Continúa creando otro bucle que recorre la lista *types* localizando el índice de los frames distintos del tipo “I” para posteriormente poder

obtener el valor qp de dichos frames y guardarlos en *qppb*, esta lista de datos (*qppb*) es equivalente a la lista *avg_qp_per_noni_frame*.

Ahora se calcula la media de la lista *qppb,* o de su homóloga *avg_qp_per_noni_frame,* si ya se tuviera, y dicho valor (*avg_qp*) se divide entre 51 para obtener el valor de *quant.*

Continua con el cálculo del MOS de cuantificación (*mos_cod_v*) utilizando la variable *quant* y la función *constrain* del script *utils*:

```
mos_cod_v = q1 + q2 * np.exp(q3 * quant)
mos_cod_v = utils.constrain(mos_cod_v, 1.0, 5.0)
```

A partir de *mos_cod_v* calculamos la degradación de cuantificación (*deg_cod_v*) utilizando las funciones *r_from_mos* y *constrain* del script *utils*:

```
deg_cod_v = 100.0 - utils.r_from_mos(mos_cod_v)
deg_cod_v = utils.constrain(deg_cod_v, 0.0, 100.0)
```

Ahora calcula la degradación de escalado (*deg_scal_v*) y la de reducción de la ratio de frame (*deg_frame_rate_v*) utilizando las funciones previamente vistas *degradation_due_to_upscaling* y *degradation_due_to_frame_rate_reduction*.

```
deg_scal_v = self.degradation_due_to_upscaling(coding_res, display_res)
deg_frame_rate_v = self.degradation_due_to_frame_rate_reduction(deg_cod_v,
deg_scal_v, framerate)
```

Por último, calcula la puntuación final (*MOS*) utilizando la función anterior *degradation_integration* y se devuelve el resultado (*score*).

- **video_model_function_mode3**

Argumentos:

self
coding_res {int} -- Indica el número de píxeles en la resolución codificada.
display_res {int} -- Indica el número de píxeles en la resolución visualizada.
framerate {float} -- Ratio de frames/s en el total del stream.
frames {list} -- lista con las imágenes dentro de la ventana de medición.
quant {float} -- Parámetro que señala la degradación de cuantificación, usado en modo debug, por defecto: None.
avg_qp_per_noni_frame {list} -- Media de cuantificación para los frames no “I”, usado en modo debug, por defecto: [ ].

Calcula la puntuación final del modelo, en el modo 3 a partir de los argumentos de esta función, además de algunos valores predeterminados ajustados al modo de funcionamiento. Para ello tiene que calcular las diferentes degradaciones.

Empieza con la degradación de compresión, empezando igual que en el modo 2, comprobando si *quant* no tiene valor, si este es el caso, se comprueba que *avg_qp_per_noni_frame* tampoco lo tenga, si esto ocurre crea un bucle que recorre la lista *frames* para obtener los tipos de frame, y los valores qp asociados a los frames, y guarda cada dato en su array correspondiente (*types* y *qp_values*). Continúa creando otro bucle, cuyo contenido es distinto al creado en el modo 2, que recorre la lista *types* localizando el índice de los frames distintos del tipo “I” para posteriormente poder obtener el valor qp de dichos frames y guardarlos en *qppb*, y también

localizando los frames del tipo “I” para reemplazar el valor qp del último frame distinto del tipo “I” por su valor anterior a este, si es posible, esta lista de datos (*qppb*) es equivalente a la lista *avg_qp_per_noni_frame*.

Ahora se calcula la media de la lista *qppb,* o de su homóloga *avg_qp_per_noni_frame,* si ya se tuviera, y dicho valor (*avg_qp*) se divide entre 51 para obtener el valor de *quant.*

Continua con el cálculo del MOS de cuantificación (*mos_cod_v*) utilizando la variable *quant* y la función *constrain* del script *utils*:

```
mos_cod_v = q1 + q2 * np.exp(q3 * quant)
mos_cod_v = utils.constrain(mos_cod_v, 1.0, 5.0)
```

A partir de *mos_cod_v* calculamos la degradación de cuantificación (*deg_cod_v*) utilizando las funciones *r_from_mos* y *constrain* del script *utils*:

```
deg_cod_v = 100.0 - utils.r_from_mos(mos_cod_v)
deg_cod_v = utils.constrain(deg_cod_v, 0.0, 100.0)
```

Ahora calcula la degradación de escalado (*deg_scal_v*) y la de reducción de la ratio de frames (*deg_frame_rate_v*) utilizando las funciones previamente vistas *degradation_due_to_upscaling* y *degradation_due_to_frame_rate_reduction*.

```
deg_scal_v = self.degradation_due_to_upscaling(coding_res, display_res)
deg_frame_rate_v = self.degradation_due_to_frame_rate_reduction(deg_cod_v,
deg_scal_v, framerate)
```

Por último, calcula la puntuación final (*MOS*) utilizando la función anterior *degradation_integration* y se devuelve el resultado (*score*).

- **handheld_adjustment**

Argumentos:

self

score {float} -- Puntuación final con el MOS (puntuación de opinión media) del video procesado.

Esta función hace un ajuste a la puntuación final porque el visionado del video ha sido en un móvil, utilizando valores predeterminado y el valor de *score*, y lo devuelve como resultado

```
return max(min(htv_1 + htv_2 * score + htv_3 * score**2 + htv_4 * score**3, 5), 1)
```

- **Model_callback**

Argumentos:

self

output_sample_timestamp {int} -- Marca de tiempo de inicio para las muestras de salida (1, 2, ...).

frames {list} -- lista con las imágenes dentro de la ventana de medición.

Función que recibe frames de la ventana de medición y utilizando la función *get_chunk* del script *utils.py* obtiene un conjunto de frames del mismo tipo (”video”) y lo guarda en la variable *frames*.

A partir de aquí según el modo de funcionamiento elegido se calculan valores necesarios y después se aplica la función *video_model_function_mode* adecuada.

En el modo 0, se calcula el *bitrate* utilizando la media del valor "Bitrate" de los frames escogidos.

En el modo 1, se calcula el *compensated_size* (tamaño del frame sin la cabecera) utilizando la función *calculate_compensated_size* del script *utils.py*, se calcula la duración (*duration*) sumando el valor *duration* de los frames escogidos y se calcula el *bitrate* utilizando la suma de los *compensated_size* entre el valor de *duration*, y pasando el resultado de bit/s a Kbyte/s.

En el modo 2 y 3, no se realizan cálculos previos a la función.

Si el dispositivo de visionado ha sido un móvil, se aplica el método *handheld_adjustment* a la puntuación final (*score*) y se obtiene un nuevo score ajustada

Al final se añade el valor de *score* a la variable *o21.*

- **check_codec**

Argumentos:
self

Función que comprueba que se están usando los códecs adecuados, en este modelo sólo es válidos los códecs de h264.

```
codecs = list(set([s["codec"] for s in self.segments]))
for c in codecs:
if c != "h264":
        raise P1203StandaloneError("Unsupported codec: {}".format(c))
```

- **calculate**

Argumentos:
self

Esta función calcula el MOS (puntuación de opinión media) del video.

Empieza utilizando la función *check_segment_continuity* del script *utils.py* la cual comprueba que los segmentos a analizar sean contiguos a los previamente analizados.

Continua creando la variable *measurementwindow* a través de la clase *MeasurementWindow()* y ejecutando su método *set_score_callback*, luego comprueba el tipo de modo en el que se ejecuta el modelo, empieza comprobando el modo 0, para ello comprueba si está el valor "frames" dentro de la variable *segment*, si se encuentra este valor el modo no es 0, continua comprobando si está el valor "qpValues" en la variable *frame*, si es así entonces ejecuta el modelo en modo 3, si este valor no está lo ejecuta en modo 1.

Después ejecuta la función *check_codec* para asegurarse que no hay fallo con los códecs.

En el siguiente paso se distingue entre el modo 0, u otro modo.
En el modo 0 será necesario crear frames falsos para luego añadirlos a la ventana de medición (*measurementwindow*). Para crear los frames artificiales primero calcula el número total de frames (*num_frame*): multiplicando la duración del segmento (*segment["duration"]*) por el

número de fotogramas por segundo (*segments["fps"]*). Luego calcula la duración de los frames (*frame_duration*): dividiendo 1 entre el número de fotogramas por segundo.
Luego un bucle de tamaño *num_frame* crea variables {dict} *frame* con los siguientes claves: "duration" con la duración del frame (*frame_duration*), "dts" con su marca de tiempo adecuada (empieza en 0 y antes de acabar el bucle se aumenta añadiendo la duración de un frame) y las claves "bitrate", "codec", "fps" y "resolution" con su valor procedente de *segment*.
Antes de acabar el bucle se añade a la ventana de medición (*add_frame*) el frame creado y cuando se acaba el bucle se ejecuta el método *stream_finished* de la clase *measurementwindow.*

Si es otro modo, empieza creando una variable *num_frame_assumed* del mismo modo que creó la variable *num_frame* en el modo 0 y la compara con la longitud de *segment["frames"]*, si existe una discrepancia se envía una alerta de advertencia y se continúa, se crea la variable *frame_duration* de la misma manera que en el modo 0 y empieza el mismo bucle que en el modo 0. Crea la variable *frame* {dict} con las mismas claves que en el modo anterior además de las siguientes: "resolution", "size", "type" para el modo 1 y 3, y "qpValues" solo para el modo 3, siendo sus valores procedentes de la variable *segment* ajustada a su frame relativo. El bucle termina de la misma forma que el anterior, añadiendo los frames a la ventana de medición , aumentando el valor de *dts* y al cerrar el bucle ejecutando el método *stream_finished*.

Por último, está función devuelve el *stream_id*, el modo (*mode*) y la puntuación o22 en formato JSON.

- **__init__**

Argumentos:
self
segments -- {list} Lista de archivos de video.
display_res -- {str} Resolución de visionado en píxel de ancho x alto, por defecto: "1920x1080".
device -- {str} Tipo de dispositivo de visualización, por defecto: "pc".
stream_id -- {str} Identifica la procedencia de los datos, por defecto: None.
coeffs -- {dict} Coeficientes usados en el modelo, sobrescribe a los predeterminados si usan la misma clave, por defecto: {}.

Inicializa los valores: *segments*, *display_res*, *device*, *stream_id, o22, mode* y *coeffs*, del modelo Pv, con los datos JSON de entrada.

3.2.2.9 P1203Pq.py

Compuesto por la clase P1203Pq y sus funciones internas.

- **Class P1203Pq(object)**

Contiene los datos de las puntuaciones finales.

Funciones internas:

- **__init__**

Argumentos:
self

021 {list} -- Lista con las puntuaciones MOS de audio.
022 {list} -- Lista con las puntuaciones MOS de video.
l_buff {list} -- Lista con la duración de los eventos de parada, por defecto: [ ].
p_buff {list} -- Lista con la posición (marca de tiempo en segundos) de los eventos de parada, por defecto: [ ].
device {str} -- Dispositivo de visualización, "pc" o "mobile".
coeffs {dict} -- Coeficientes usados en el modelo, sobrescribe a los predeterminados si usan la misma clave, por defecto: {}.
amendment_1_audiovisual {bool} -- Habilita el arreglo para la enmienda 1, de la cláusula 8.2, ajuste para el caso de muy baja calidad de audio, por defecto: False.
amendment_1_stalling {bool} -- Habilita el arreglo para la enmienda 1, de la cláusula 8.4, ajuste para el caso de eventos de parada muy largos, por defecto: False.

Inicializa los valores: *O21*, *O22*, *device*, *amendment_1_audiovisual*, *amendment_1_stalling*, *has_audio*, *has_video*, *l_buff* y *p_buff*, con los datos JSON de entrada.

- **_calc_stalling_impact**

Argumentos:
self
num_stalls {int} -- Número de eventos de parada.
total_stall_len {int} -- Duración total en segundos de la suma de la ponderación de las paradas, según su posición en el segmento.
duration {int} -- Duración en segundos del segmento.
avg_stall_interval {int/float} -- Duración media en segundos de los eventos de parada.

Función que calcula el impacto de los eventos de parada (*stalling_impact*), para ello hace uso de los parámetros de entrada, además de coeficientes predeterminados, se devuelve *stalling_impact*.

```
stalling_impact = np.exp(-num_stalls / self.coeffs["s1"]) * \
np.exp(-total_stall_len / duration / self.coeffs["s2"]) * \
np.exp(-avg_stall_interval / duration / self.coeffs["s3"])
```

- **_calc_stalling_features**

Argumentos:
self
duration {int} -- Duración en segundos del segmento.

Función que calcula varios parámetros de los eventos de parada, para ello hace uso de los parámetros de entrada, además de coeficientes predeterminados.

El primer parámetro *total_stall_len* es el sumatorio de las duraciones (en segundos) de los eventos de parada, siendo estas ponderadas según su posición en el segmento.

```
total_stall_len = sum(
[l_buff * utils.exponential(1, self.coeffs["c_ref7"], 0, self.coeffs["c_ref8"],
duration - p_buff)
for p_buff, l_buff in zip(self.p_buff, self.l_buff)]
```

El segundo parámetro *num_stalls* es el número total de eventos de parada.

```
num_stalls = len(self.l_buff)
```

El tercer y último parámetro *avg_stall_interval* es el tiempo medio entre los eventos de parada, si hay 1 o 0 eventos de parada este valor es 0.

```
if num_stalls > 1:
    avg_stall_interval = sum([b - a for a, b in zip(self.p_buff, self.p_buff[1:])]) /
(len(self.l_buff) - 1)
```

La función devuelve los tres parámetros anteriores.

- **_calc_video_quality_change_rate**

Argumentos:
self
duration {int} -- Duración en segundos del segmento.

Función que calcula la ratio de cambio de calidad (*vid_qual_change_rate*), para ello desde el segundo valor de O22, se cuenta el número de veces que la diferencia entre dos valores consecutivos es mayor a 0.2, y ese valor es dividido por último entre la duración del segmento, se devuelve el resultado como *vid_qual_change_rate*.

- **_calc_034_035_baseline**

Argumentos:
self
duration

Función que calcula la calidad audiovisual por intervalo de muestras (*O34*) y un valor base de la calidad audiovisual final de codificación (*O35_baseline*).

Empieza creando un bucle que recorre cada segundo (*t*) de *duration*, durante este bucle se rellena la lista *O34* con un valor ajustado para cada segundo, utiliza coeficientes fijados y los valores de *O21 y O22* ajustados al segundo a tratar. Si el valor de *amendment_1_audiovisual* es True durante el bucle se ajusta el valor de *O34* con coeficientes fijados. También durante el bucle se van formando las variables *O35_numerator* y *O35_denominator* de la siguiente forma:

```
temp = O34[t]
w1 = self.coeffs["t1"] + self.coeffs["t2"] * np.exp((t / float(duration)) / self.coeffs["t3"])
w2 = self.coeffs["t4"] - self.coeffs["t5"] * temp

O35_numerator += w1 * w2 * temp
O35_denominator += w1 * w2
```

Al terminar el bucle se calcula *O35_baseline* dividiendo *O35_numerator* entre *O35_denominator*, y la función devuelve *O34* y *O35_baseline*.

- **_calc_and_test_osc**

Argumentos:
self
duration {int} -- Duración en segundos del segmento.

q_dir_changes_longest {int} -- Duración en segundos del mayor período sin cambios positivos o negativos de la calidad del segmento.
q_dir_changes_tot {int} -- Número total de cambios de dirección de la calidad.
vid_qual_spread {int} -- Es la diferencia entre el máximo valor y el mínimo valor de *O22*.

Primero comprueba 2 condiciones, si *q_dir_changes_longest* entre *duration* es menor que 0.25 y si *q_dir_changes_longest* es menor a 30, y guarda el valor en el boolean *osc_test*.

Si *osc_test* es False, la variable *osc_comp* vale 0, si es True entonces hay que calcular primero una variable *q_diff*, utilizando *vid_qual_spread* y asegurándose que obtiene un valor positivo, y después se calcula osc_comp como:

```
osc_comp = np.maximum(0.0, np.minimum(
        q_diff * np.exp(self.coeffs["comp1"] * q_dir_changes_tot +
self.coeffs["comp2"]), 1.5))
```

Al final se devuelve *osc_comp*.

- **_calc_qdir**

Argumentos:
self

Esta función es la encargada de aplicar el algoritmo necesario para calcular el mayor período sin cambios positivos o negativos de la calidad del segmento (*q_dir_changes_longest)* y el número total de cambios de dirección de la calidad (*q_dir_changes_tot)*.

Primero crea un relleno al principio (*padding_beg)* y al final (*padding_end)* de la lista *O22*, este relleno está compuesto por el primer valor de *O22* repetido 4 veces más antes del valor real, y del último valor de *O22* repetido cuatro veces más después del valor real, y lo guarda en *padded_O22*.

Esta lista con relleno es ahora transformada por medio de una convolución con un filtro pasa bajo FIR (Respuesta finita al impulso) de tamaño 5, el resultado es la lista *ma_filtered*.

Ahora crea un bucle que recorre ma_filtered calculando la diferencia entre el valor 3 pasos en adelante y el valor actual, de modo que si el resultado es mayor que 0.2, se añade a la lista *QC* un 1, si el valor es menor que 0.2, se añade a la lista *QC* un -1, y si el resultado está entre 0.2 y -0.2 se añade a *QC* un 0.

Ahora crea un bucle que recorre *QC* tomando el índice (*index*) y el valor de ese índice (*val*), entonces comprueba que cuando la variable val sea distinta de 0: guarda el primer índice y su valor, en la lista *lens*, y a partir de entonces guarda la dupla de [*index*, *val*] en la lista *lens* cuando el valor de *val* sea distinto del de la última dupla guardada, de forma que lens se convierte al final en una lista con las posiciones y los cambios de dirección de calidad.

Si no tiene contenido la variable lens *q_dir_changes_longest* se calcula multiplicando la longitud de *QC* por 3 (el número de pasos utilizados en la formación de *QC*).

Si la variable *lens* tiene contenido, continúa añadiendo al principio y al final de esta lista una dupla [0,0], para seguidamente calcular la variable *distances*. Para ello crea un bucle que recorre la lista lens y va restando el índice del siguiente elemento, al índice actual, y guardando los resultados en la lista *distances*, guardando así las distancias entre cambios de dirección de calidad. Como último paso para calcular *q_dir_changes_longest* multiplica el valor máximo de *distances* por 3 (el número de pasos utilizados en la formación de *QC*).

Para calcular *q_changes_tot* hace una sumatoria con un bucle que recorre *QC* primero quitando los valores iguales a cero y luego sumando 1 por cada conjunto de valores iguales, dando como resultado el número total de cambios de dirección.

Devuelve los calores de *q_dir_changes_longest* y *q_dir_changes_tot*.

- **calculate**

Argumentos:

self

Esta función calcula el MOS (puntuación de opinión media) total de la sesión (*O46*), la percepción de parada (*O23*), la calidad audiovisual por intervalo de muestras (*O34*) y la calidad audiovisual final de codificación (*O35*) .

Empieza calculando la longitud de O21 (O21_len) y de O22 (O22_len) contando el número de valores en sus respectivas listas, luego comprueba si hay video y audio comprobando las variables *has_video* y *has_audio*, si no hay vídeo la función se para con un error por falta del vídeo, por otro lado, si no hay audio: salta una advertencia, supone que la calidad del audio es alta rellenando la variable O21 con el máximo valor (5.0) y asigna a la variable de duración (*duration*) el valor de *O22_len*. Si *has_video* y *has_audio* son "True" entonces el valor de *duration* es igual al valor más pequeño entre *O21_len* y *O22_len*.

Ahora sigue calculando varias variables intermedias y finales, algunas de forma inmediata, y otras utilizando una función dentro de este script:

total_stall_len, *num_stalls* y *avg_stall_interval* utilizando la función vista previamente *_calc_stalling_features*.

vid_qual_spread es la diferencia entre el valor máximo de *O22* (*max(self.O22)*) y el valor mínimo de O22 (*min(self.O22)*).

vid_qual_change_rate utilizando la función vista previamente *_calc_video_quality_change_rate*.

Q_dir_changes_longest y *q_dir_changes_tot* utilizando la función vista previamente *_calc_qdir*.

O34 y *O35_baseline* utilizando la función vista previamente *_calc_034_035_baseline*.

O34_diff es una lista con la diferencia entre cada valor de *O34* y el valor de *O35_baseline*, multiplicado por una lista de ponderaciones (*w_diff*), cuyo valor se calcula utilizando coeficientes predeterminados, las posiciones temporales de la lista *O34* y la duración del segmento.

neg_perc es el percentil 10% de *O34_diff*.

negative_bias es *neg_perc* multiplicado por un coeficiente predeterminado, asegurándose que *neg_perc* no es un valor negativo, en cuyo caso vale 0.

stalling_impact utilizando la función vista previamente *_calc_stalling_impact*.

23 (el valor de percepción de parada) se calcula multiplicando por 4 el valor de *stalling_impact* y sumando 1 al resultado.

osc_comp utilizando la función vista previamente _calc_and_test_osc.

adapt_test es un boolean que comprueba que *q_dir_changes_longest* entre *duration* sea menor que 0.25

adapt_comp depende del valor de *adapt_test*, si es False vale 0, si es True vale:

```
adapt_comp = np.maximum(0.0, np.minimum(self.coeffs["comp3"] *
vid_qual_spread * vid_qual_change_rate + self.coeffs["comp4"], 0.5))
```

O35 (la calidad audiovisual final de codificación) se calcula restando a *O35_baseline* las siguientes variables, *negative_bias*, *osc_comp* y *adapt_comp*.

mos es un valor intermedio, se calcula restando 1 a O35, multiplicando el resultado por el de *stalling_impact* y por último sumando 1 al último valor obtenido.

rf_score es la predicción de MOS por Machine Learning utilizando el método Random Forest, se calcula utilizando la función *calculate* dentro del script *rfmodel.py*.

O46 (la calidad final de sesión) se calcula primero un valor temporal utilizando la siguiente fórmula:

```
O46 = 0.75 * np.maximum(np.minimum(mos, 5), 1) + 0.25 * rf_score
```

Ahora comprobamos si el valor de *amendment_1_stalling* es True, en este caso aplicamos el siguiente ajuste al valor de *O46*:

```
q_fac = min(max(self.coeffs["amd_1_a1"] * total_stall_len + self.coeffs["amd_1_a2"],
0), 1)
O46 = 1 + (O46 - 1) * q_fac
```

Por último, el valor de *O46* es ajustado para compensar las diferencias subjetivas de evaluación entre los distintos laboratorios.

```
O46 = self.coeffs["f1"] + self.coeffs["f2"] * O46
```

Está función devuelve *O23*, *O34*, *O35* y *O46* en formato JSON.

3.2.2.9 P1203_standalone.py

Compuesto por la clase P1203Standalone y sus funciones internas.

- **Class P1203Standalone()**

Clase utilizada para calcular de forma autónoma el modelo P1203 tomando como datos de entrada archivos en formato JSON.

Funciones internas:

- **__init__**

Argumentos:
self
input_report {Dict} -- Informe de entrada de datos en formato JSON.
debug {bool} -- Habilita la visualización de los informes de debug, por defecto: False.
Pa {} -- Indica el módulo de estimación de calidad de audio, por defecto: P1203Pa.
Pv {} -- Indica el módulo de estimación de calidad de video, por defecto: P1203Pv.
Pq {} -- Indica el módulo de integración audiovisual, por defecto: P1203Pq.
quiet {bool} -- Silenciar los mensajes del registro, por defecto: False.
amendment_1_audiovisual {bool} -- habilitar la corrección 1 de la cláusula 8.2, por defecto: False.
amendment_1_stalling {bool} -- habilitar la corrección 1 de la cláusula 8.4, por defecto: False.

Inicializa los valores: *input_report*, *debug*, *Pa*, *Pv*, *Pq,* amendment_1_audiovisual y amendment_1_stalling, con los datos JSON de entrada. Y los valores *audio*, *video*, *integration* y *overall_result*, como None (vacíos).

- **calculate_pa**

Argumentos:
self

Esta función calcula el resultado del módulo de estimación de calidad de audio (*audio*).

Empieza comprobando si el dato de entrada *input_report* contiene la clave
'I11', si es así intenta rellenar las variables *segments* y *stream_id*, y con estos datos como argumentos ejecuta la función *calculate* de *Pa* para obtener los resultados para rellenar la variable *audio*. Si no contenía 'I11' comprueba si tiene la clave 'O21' y de esta forma rellenar la variable *audio* con los datos de 'O21' y poniendo como "streamId" el valor -1. Si no tenía ninguna de las claves anteriores manda un mensaje de error utilizando *P1203StandaloneError*.

Ahora comprueba si el boolean debug es True para imprimir por consola un JSON con los datos de *audio*.

Al final devuelve el valor de *audio*.

- **calculate_pv**

Argumentos:
self

Esta función calcula el resultado del módulo de estimación de calidad de video (*video*).

Empieza comprobando si el dato de entrada *input_report* contiene la clave
'I13', si es así intenta rellenar las variables *segments*, *display_res*, *stream_id* y *device*, y con estos datos como argumentos ejecuta la función *calculate* de *Pv* para obtener los resultados para rellenar la variable *video*. Si no contenía 'I13' comprueba si tiene la clave 'O22' y de esta forma rellenar la variable *audio* con los datos de 'O22' y poniendo como "streamId" el valor -1. Si no tenía ninguna de las claves anteriores manda un mensaje de error utilizando *P1203StandaloneError*.

Ahora comprueba si el boolean debug es True para imprimir por consola un JSON con los datos de *audio*.

Para terminar, devuelve el valor de *video*.

- **calculate_integration**

Argumentos:
self

Esta función calcula el resultado del módulo de integración (*integration*).

Empieza comprobando si el dato de entrada *input_report* contiene la clave
'I23' y si dentro de esta hay una clave 'stalling' no vacía, si es así, rellena la variable *stalling*, sino, *stalling* vale: [ ].

Continúa intentando añadir el valor de *input_report*["IGen"]["device"] a *device*, si no puede *device* es, por defecto: "pc".

Ahora realiza un bucle que recorre *stalling* rellena *l_buff* con los valores de las duraciones de los eventos de parada, y hace lo mismo con las marcas de tiempo de los eventos de parada rellenando *p_buff,* excepto en el caso de no haber un evento de parada en el segundo 0, de modo que primero habría que eliminar el primer valor *stalling* antes de rellenar *p_buff*.

Con estos datos como argumentos, además de *O21*, *O22*, *amendment_1_audiovisual* y *amendment_1_stalling*, ejecuta la función *calculate* de *Pq* para obtener los resultados para rellenar la variable *integrate* y por último devuelve el valor de *integrate*.

- **calculate_complete**

Argumentos:
self
print_intermediate {bool} -- Indica si muestra por consola los resultados intermedios del cálculo del modelo, por defecto: False.

Esta función calcula los resultados del módulo p1203 (*overall_result*).

Primero ejecuta las 3 funciones previamente vistas, *calculate_pa*, *calculate_pv* y *calculate_integration*.

Continúa intentando extraer los valores de *stream_id* y *mode* de la variable *video* en la clave 'video', si no puede les asigna: -1.

Ahora igual el valor de *overall_result* con el de *integration*, calculado con *calculate_integration*, y le añade las claves de 'streamId', 'mode' y 'date' con sus valores correspondientes.

Si el valor de *print_intermediates* es True también se añade a *overall_result* las claves 'O21' y 'O22' con sus valores de *audio* y *video* calculados en *calculate_pa* y *calculate_pv* respectivamente.
Por último, se devuelve *overall_result*.

3.2.2.10 rfmodel.py

Conjunto de funciones para la ejecución del algoritmo Random Forest.

Funciones:

- **execute_trees**

Argumentos:
features {} -- Son las características (datos de entrada) necesarios para ejecutar el algoritmo Random Forest.
path {} -- Dirección del directorio *trees* donde se encuentran los datos necesarios para ejecutar el algoritmo Random Forest.

Primero crea un bucle que recorre el directorio *path* en busca de archivos que empiezan por "tree" y terminan en ".csv". Cada vez que encuentra un archivo que cumple los requerimientos, lo guarda como *tree_matrix*, y ejecuta la función *execute_tree*, con sus argumentos ajustados, añadiendo el resultado a la lista *resallo*.
Cuando termina el bucle se hace la media de los valores de la lista *res_all* y se devuelve el resultado, *res_mean.*

- **execute_tree**

Argumentos:
features {} -- Son las características (datos de entrada) necesarios para ejecutar el algoritmo Random Forest.
tree_matrix {} -- Es la variable generada a partir del texto con los datos necesarios para aplicar el algoritmo Random Forest.

Función compuesta por otra función interna *recurse_execute* que toma los datos de *execute_tree*.

La devolución de datos es una llamada a la función *execute_tree* dándole como argumento a *node_id* el valor 0.

- **recurse_execute**

Argumentos:
node_id {int} -- Identifica el nodo del que extraer información.

Esta función empieza extrayendo los 4 valores del nodo indicado (node_id), en orden:

El identificador de la característica, feature_id.
El valor umbral que debe traspasar para pasar al siguiente nodo, *feature_thres*.

El identificador del siguiente nodo si no supera el umbral, *left_child*.
El identificador del siguiente nodo si supera el umbral, *right_left*.

Ahora comprueba si es el final del algoritmo (*feature_id* == -1), si este es el caso devuelve el último valor de *feature_thres*, si no es el final, comprueba el valor de *features*[*feature_id*] respecto a *feature_thres*, y vuelve a ejecutar *recurse_execute* tomando como *node_id*, el valor de *left_child* si es menor, y el valor de *right_child* si es mayor.

- **scale_moses**

Argumentos:

sec_mos {list} -- Lista con las puntuaciones de *O21* u *O22* redondeadas a 3 decimales.

num_splits {int} -- Número de particiones a los datos de *sec_mos*, *O21* se divide en 2 partes y *O22* se divide en 3.

Función que calcula la media de puntuación de O22 u O21 por partes.
Primero calcula la duración total (*total_duration*) con la longitud de *sec_mos* , y a continuación la duración de cada parte (*split_duratioin*) dividiendo *total_duration* entre *num_split*.

Continúa haciendo un bucle que recorre *total_duration*, calculando recursivamente *previous_mos* e incrementando *previous_time* hasta que termina una parte (*split_duration*), entonces utiliza otra fórmula que calcula *mos* utilizando los datos recursivos entre otros. Este valor se añade a *mos_samples* y se vuelve a poner a 0 a *previous_time*, para calcular las demás partes si quedan.

Al terminar el bucle se comprueba que el tamaño de *mos_samples* sea igual al número de partes (*num_splits*), si no son iguales añade el último valor de *previous_mos* a *mos_samples* y por último devuelve la variable *mos_samples*.

- **get_rebuf_stats**

Argumentos:

l_buff {list} -- Lista con la duración de los eventos de parada, por defecto: [ ].

p_buff {list} -- Lista con la posición (marca de tiempo en segundos) de los eventos de parada, por defecto: [ ].

duration {int} -- Duración en segundos del segmento.

Esta función calcula los parámetros relacionados con los eventos de parada.

Primero comprueba que *l_buff* y *p_buff* contienen datos, si están vacíos devuelven [0, 0, 0, 0, *duration*].

Si contienen datos crea un bucle que recorre *p_buff* y va guarda la dupla de datos [*p_buff*, *l_buff*] en la variable *events*, a excepción del evento de parada inicial, si hubiera.

Los parámetros que calcula son los 5 siguientes:
La longitud de *events*, el número de eventos de parada sin contar la parada inicial, *num_rebuf*.
El sumatorio de los valores *l_buff* de *events*, longitud total de los eventos de parada, *len_rebuf*.
El valor de *num_rebuf* entre *duration*, la ratio de eventos de parada por la duración total, *num_rebuf_per_length*.
El valor de *len_rebuf* entre *duration*, la ratio de la duración de eventos de parada por la duración total, *len_rebuf_per_length*.

El valor de *duration* menos la marca de tiempo del último evento de parada, es el tiempo transcurrido desde el inicio del último evento de parada hasta el final, *time_of_last_rebuf*.

Se devuelven los 5 parámetros dentro de un array.

- **calculate**

Argumentos:

021 {list} -- Lista con las puntuaciones MOS de audio.
022 {list} -- Lista con las puntuaciones MOS de video.
l_buff {list} -- Lista con la duración de los eventos de parada, por defecto: [ ].
p_buff {list} -- Lista con la posición (marca de tiempo en segundos) de los eventos de parada, por defecto: [ ].
duration {int} -- Duración en segundos del segmento.

Calcula los pasos intermedios y la puntuación final de mos a través del algoritmo Random Forest.

Empieza comprobando que *l_buff* y *p_buff* contienen datos y si existe evento de parada inicial, si lo hay, guardan la duración en *initial_buffering_length*, en caso contrario *initial_buffering_length* vale 0.

Después ejecuta la función *get_rebuf_stats* y guarda el resultado en *rebuf_stats,* y modifica los valores de *rebuf_stats*[1] (*len_rebuf*) y de *rebuf_stats*[3] (*len_rebuf_per_length*) sumando un tercio de *initial_buffering_length* y sumando un tercio de *initial_buffering_length* entre *duration* respectivamente, para añadir la información de la parada inicial ajustada.

Ahora redondea los valores de *O21* y *O22* a 3 decimales y ejecuta la función *scale_moses* para ambos valores guardando el resultado de *O21* en *sec_moses_feature_audio* y el de *O22* en *sec_moses_feature_video*.

Continúa calculando los percentiles de 1 , 5 y 10 de *O22_rounded* y guarda el resultado en *sec_mos_stats*.

Guarda la dirección del directorio “trees” a través de *__file__* en la variable *tree_path*

Y por último ejecuta la función *execute_trees* tomando como argumentos los parámetros que ha ido calculando previamente, devuelve el resultado *rf_score*.

3.2.2.11 utils.py

Conjunto de funciones auxiliares, utilizadas por los demás scripts.

Funciones:

- **which**

Argumentos:

program {str} -- Variable con la dirección y el nombre del programa.

Función compuesta por otra función interna *is_exe* que toma los datos de *which*.

Su función es comprobar si existe un programa ejecutable.

Primero divide la variable *program* en dos partes, la dirección hasta la última "/" (*fpath*) y el nombre del archivo, para ello utiliza el script *split* del módulo *os.path*.

Si existe *fpath* se ejecuta *is_exe* para saber si es ejecutable el programa, y si es así se devuelve *program*. Si no existe *fpath* se utiliza la variable global *path* la cual primero se prepara, y luego se une a la variable *program*, formando la variable *exe_file*, para luego ejecutar *is_exe* con esta variable como argumento, si *is_exe* da positivo, se devuelve *exe_file*. En otro caso se devuelve *None*.

- **is_exe**

Argumentos:

fpath {str} - Variable con la dirección y el nombre del programa.

Función que devuelve un boolean afirmando si el programa de argumento es ejecutable.

Primero comprueba dos estados: que existe el archivo con la función *isfile* del módulo *os.path* y que es accesible con el módulo *os.access* utilizando como argumentos *fpath* y *os.X_OK* que comprueba que *fpath* es ejecutable. Si los dos son True, la variable *found* es True.

En el caso de que *found* sea False y el sistema operativo sea "win32" se intenta añadir ".exe" al final de *fpath* y se hace la misma comprobación que al principio.

Se devuelve *found*.

- **calculate_compensated_size**

Argumentos:

frame_type {str} -- Indica el tipo de imagen*: I/NO-I o I/P/B*.
size {float} -- Tamaño en bytes dado por el informe de ffmpeg.
dts {float} -- Marca de tiempo de decodificación, por defecto None.

Compensa los valores de tamaño erróneos obtenidos de ffmpeg, los cuales contienen información adicional como: SPS(Sequence Parameter Set), PPS(Picture Parameter Set) y AUD(Access Unit Delimiter).

Si *dts* tiene un valor y este valor es 0 , la variable *size_copmensated* vale *size* - 800.

Si no ocurre esto se comprueba el tipo de frame, si es "I" el valor de *size_compensated* es *size* - 55, si no es del tipo "I" el valor es *size* - 11.

Se devuelve el valor de *size_compensated* si este es positivo, sino se devuelve 0.

- **mos_from_r**

Argumentos:

Q {int} -- Calidad estimada de codificación de video.

Función que calcula un resultado MOS (Puntuación de opinión media del video) dado un valor de Q y utilizando los parámetros MOS_MAX y MOS_MIN previamente definidos.

```
        MOS = MOS_MIN + float(MOS_MAX - MOS_MIN) * float(Q) / 100.0 + float(Q) * \
float(Q - 60.0) * float(100.0 - Q) * 0.000007
```

Devuelve el valor de MOS si está entre el valor máximo y mínimo, o en su defecto el valor máximo si lo sobrepasa o el mínimo si es inferior.

- **r_from_mos**

Argumentos:

MOS {int/float} -- Puntuación de opinión media del video.

Función que calcula Q (Calidad estimada de codificación de video) dado un valor de MOS y utilizando interpolación lineal revisando los listados de resultados creados a partir de *mos_from_r* (*R_FROM_MOS_KEYS* y *R_FROM_MOS_VALUES*).

Si el valor de MOS está fuera de los márgenes, obtiene el valor del margen más cercano.

Para realizar la interpolación primero se verifica si el valor de MOS se encuentra en *R_FROM_MOS_KEYS*, si es así, se obtiene el índice y se busca el valor con el mismo índice en *R_FROM_MOS_VALUES* y obtenemos Q.
Si el valor de MOS no se encuentra en *R_FROM_MOS_KEYS* se utiliza la función *interp* del módulo *NumPy* que realiza la interpolación lineal tomando como argumentos: *MOS*, *R_FROM_MOS_VALUES* y *R_FROM_MOS_KEYS*. El resultado se guarda en la variable Q.

Se devuelve la variable Q.

- **constrain**

Argumentos:

x {list} -- Vector de valores.
minimum {float} -- Valor mínimo, por defecto “0.0”.
maximum {float} -- Valor máximo, por defecto “100.0”.

Función que constriñe un vector de entrada “x” entre un cierto mínimo y máximo. Para ello utiliza las funciones *maximum* y *minimum* del módulo NumPy. Devuelve el vector resultado.

- **sigmoid**

Argumentos:

min_x {float} -- Valor máximo de y.
min_y {float} -- Valor mínimo de y.
sat_bottom {float} -- Valor mínimo de x para la pendiente de la curva.
sat_top {float} -- Valor máximo de x para la pendiente de la curva.
x {list} -- Vector de valores para la función.

Función sigmoide de tipo Boltzmann utilizada para describir comportamientos en los que se pasa de un valor estable, a otro de una magnitud diferente, en nuestro caso se aplica para calcular la variación del MOS respecto al modo 0, en el modo 1.

- **exponential**

Argumentos:

a {float} -- Valor fijo del eje y, en el margen izquierdo
b {float} -- Define la pendiente de la curva.

c {float} -- Desplaza el punto fijo de la curva en el eje x, por defecto es 0.
d {float} -- Define la pendiente de la curva.
x {list} -- Vector de valores para la función.

Función exponencial con la siguiente forma:

$$y = b + (a - b) \cdot exp(-(x - c) \cdot \frac{log(0.5)}{(-(d - c))})$$

Utilizada para calcular la longitud total de parada ponderada en el módulo de parada dentro del módulo de cálculo de calidad de integración.

- **resolution_to_number**

Argumentos:
string {str} -- Resolución de visionada dado en formato "AnchoxAlto" (número de píxeles), por ejemplo "1920x1080".

Calcula el número total de píxeles dividiendo el string en dos valores del tipo int y devolviendo el resultado de la multiplicación.

- **check_segment_continuity**

Argumentos:
segments -- {list} Lista de archivos de video.
type {str} -- Indica si se trata de "video" o "audio", por defecto: video.

Empieza con un bucle que recorre la variable *segments* comprobando que la suma de las claves ["*start*"] y ["*duration*"] de cada archivo redondeada al decimal coincide con el valor de la clave ["*start*"] del siguiente archivo, en caso de no coincidir salta un mensaje de advertencia a través del script *logger.py*.

- **get_chunk_hash**

Argumentos:
frame {dict} -- Variable cuyas claves son la duración y la dts (*marca de tiempo de decodificación*) de una imagen.
type {str} -- Indica si se trata de "video" o "audio", por defecto: video.

Función que devuelve un valor hash que identifica cada frame según su nivel de calidad. Para ello utiliza la clave ["representation"] si el frame la tiene, si no es el caso hay que distinguir entre video y audio, en video se devuelve la suma de los valores de las siguientes claves: ["bitrate"], ["codec"] y ["fps"].
Para audio es la suma de las siguientes claves: ["bitrate"] y ["codec"].
Si no es nada de lo anterior salta un mensaje de error con tipo de frame invalido.

- **get_chunk**

Argumentos:
frame {list} -- Lista con los diccionarios frame, contando cada uno de ellos con las claves: ["bitrate"], ["codec"] y ["framerate"].
output_sample_index {int} -- Índice de la marca de tiempo utilizada como ejemplo para obtener los frames de la misma calidad.
type {str} -- Indica si se trata de "video" o "audio", por defecto: video.

Función que devuelve una lista de frames con la misma calidad que la del frame indicado por el valor *output_sample_index*.
Para ello primero obtenemos el frame elegido y lo guardamos en la variable *target_frame*, a continuación, obtenemos el "hash" artificial por medio de la función *get_chunk_hash* la cual he explicado anteriormente y buscamos los demás frames con ese mismo hash con un bucle que recorre la lista *frames* primero decrementando en uno hasta que sea distinto el hash y luego aumentando el hash hasta llegar a uno que es distinto. De esta forma se crea un rango de frames (*window*) que incluye todos los frames con el mismo hash y se devuelve *frames*[w] siendo *w* los índices de los frames dentro de *window*.

- **read_json_without_comments**

Argumentos:
input_file {str} -- archivo de entrada (JSON o archivo de video, o STDIN si "-")

Función que abre el archivo obtenido en *input_file*, lo lee y elimina los comentarios escritos en el mismo. Devolviendo el resultado como un objeto.

3.3 Librería Pyshark

Ilustración 11 - Logotipo de Pyshark

Librería encargada de traspasar la línea de comandos del programa *Wireshark* (*tshark*) para su uso en Python, a diferencia de librerías similares no traspasa ningún paquete, simplemente utiliza tshark y su función para exportar los archivos en formato XML. (15)

3.3.1 Funciones

- **FileCapture**

Argumentos:
input_file {str} -- Dirección del archivo de entrada con formato *packet capture* (.pcap).
display_filter {str} --Filtro utilizado para discernir los archivos de video de la captura.
use_json {bool} -- Indica si se utiliza el formato JSON, por defecto: True.
include_raw {bool} -- Indica si se añade el contenido del paquete sin modificar, por defecto: True.

Función que extrae los paquetes de video deseados de un archivo (*input_file*) mediante un filtro (*display_filter*) y los guarda tanto en formato JSON, como sin modificar (RAW).

3.4 Framework FFmpeg

Framework líder en la manipulación de archivos multimedia, capaz de descodificar, codificar, transcodificar, multiplexar, demultiplexar, filtrar y

Ilustración 12 - Logotipo de FFmpeg

reproducir todo tipo de formatos, desde los más antiguos y obscuros hasta los más novedosos y complejos. (16)

3.4.1 Funciones utilizadas

- **input**

Argumentos:

filename {str} -- Dirección del archivo de entrada URL.

Función para añadir un archivo al proceso actual de ffmpeg, se puede utilizar más de una vez para añadir distintos archivos al proceso.

- **output**

Argumentos:

stream {list} -- Dirección del archivo de entrada con formato *packet capture* (.pcap)

filename {str} -- Indica el nombre del archivo de salida.

shortest {bool} -- Indica si utiliza el archivo de entrada (stream) más corto como tiempo máximo, por defecto: None.

vcode {str} -- Indica la codificación del video de salida, si se utiliza 'copy' copia la misma codificación que la de entrada.

y {bool} -- Indica si se sobrescribe el archivo de salida, por defecto: None.

Función que indica cómo tratar el archivo de salida del actual proceso de ffmpeg.

- **run()**

Sin argumentos.

Ejecuta el proceso actual de ffmpeg con las distintas indicaciones especificadas previamente mediante otras funciones, devolviendo un resultado final.

3.4.2 ffprobe

Es una herramienta dentro del framework ffmpeg utilizada para recolectar información de un archivo multimedia y mostrarla tanto en formato legible por máquina, como por humanos.

3.4.3 ffmpeg-parse-qp

Librería complementaria a ffmpeg/ffprobe cuya función es calcular los parámetros de cuantificación por cada frame, o macrobloque, dentro de una secuencia.
En nuestro caso es necesario extraer dichos valores de un video y poder utilizar el modo 3 de funcionamiento. Cabe destacar que la librería está en modo experimental aún y puede no funcionar en todos los videos de entrada, por lo tanto, no puede utilizarse para validar otras implementaciones.

3.5 Librería Matplotlib

Ilustración 13 - Logotipo de Matplotlib

Es una librería orientada a la visualización de datos estáticos, animados, e interactivos en Python.
En este proyecto es utilizado para mostrar los valores de salida de *p_1203* (O.23, O.35, O.46 y O.34) como una gráfica de barras. (17)

3.5.1 Funciones

- **figure**

Argumentos:
figsize {(float, float)} -- Dupla de valores float que indican la anchura y la altura de la figura total en pulgadas, por defecto: (18,8).

Crea un contenedor de dimensiones *figsize*, para mostrar las distintas gráficas en su interior.

- **suptitle**

Argumentos:
t {str} -- El título del texto.

Añade un título "*t"* centrado a la figura. En nuestro caso el título incluye el modo utilizado, la fecha y el identificador del stream.

- **subplot**

Argumentos:
nrows {int} -- Parámetro que indica el número de filas de gráficas.
ncols {int} -- Parámetro que indica el número de columnas de gráficas.
index {int} -- Índice que indica instintivamente una gráfica dentro de una figura.

Añade una gráfica vacía a la figura actual.

- **title**

Argumentos:
label {str} -- Indica el texto utilizado para el título.

Fija un texto (*label)* como título centrado de la gráfica.

- **bar**

Argumentos:
label {list} -- Indica los textos {str} utilizado para poner como leyenda debajo de cada barra.
x {list} -- Lista con los valores {float} de cada barra de la gráfica

Crea una gráfica de tipo barras con leyenda (*label*) y los valores de *x* dentro de la gráfica vacía creada con *subplot*.

- **text**

Argumentos:

x {float} -- Indica la posición respecto al eje x, donde va la información.
y {float} -- Indica la posición respecto al eje y, donde va la información.
text {str} -- Indica el texto utilizado para añadir información adicional.
ha {str} -- Indica el alineamiento horizontal del texto, por defecto: 'left'.
va {str} -- Indica el alineamiento vertical del texto, por defecto: 'baseline'.
rotation {float} -- Indica los grados de rotación del texto, por defecto: 0.

Fija un texto (*text)* como información adicional a la gráfica, en nuestro caso muestra el valor de cada barra encima de su respectiva representación, siguiendo las indicaciones de los argumentos.

- **show()**

Sin argumentos.

Ejecuta el proceso actual de *Matplotlib* con las distintas indicaciones especificadas previamente mediante otras funciones, devolviendo las gráficas finales.

3.6 Librería JSON

Librería que habilita el uso del formato JSON (JavaScript Object Notation) en Python. Este formato de intercambio de información es ampliamente usado en muchos lenguajes de programación por su alto nivel de funcionalidad. Utiliza texto legible por humanos para almacenar valores asociados a atributos y cualquier tipo de dato que pueda ser serializado. En nuestro caso sirve para almacenar los datos cuando son extraídos de un video y poder transmitirlos de forma ordenada a la librería *p_1203* para que pueda calcular los resultados finales. Además, el resultado de *p_1203* también se devuelve en formato JSON para que *Matplotlib* pueda interpretarlo y mostrar los datos de forma gráfica. (18)

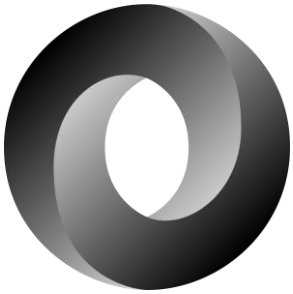

Ilustración 14 - Logotipo de JSON

mnt > LHDD > TFG > itu-p1203 > examples > {} mode1.json > ...


```
{
    "I11": {
        "segments": [
            {
                "bitrate": 331.46,
                "codec": "aaclc",
                "duration": 5.48,
                "start": 0
            },
            {
                "bitrate": 333.02,
                "codec": "aaclc",
                "duration": 5.0,
                "start": 5.48
            },
            {
                "bitrate": 333.2,
                "codec": "aaclc",
                "duration": 5.0,
                "start": 10.48
            },
            {
                "bitrate": 333.57,
                "codec": "aaclc",
                "duration": 4.598,
                "start": 15.48
            }
        ],
        "streamId": 42
    },
    "I13": {
        "segments": [
            {
                "bitrate": 691.72,
                "codec": "h264",
                "duration": 5.48,
                "fps": 25.0,
                "frames": [
                    {
                        "frameSize": "11206",
                        "frameType": "I"
                    },
                    {
                        "frameSize": "8557",
                        "frameType": "Non-I"
                    },
                    {
                        "frameSize": "822",
                        "frameType": "Non-I"
                    },
                    {
                        "frameSize": "7676",
                        "frameType": "Non-I"
```


Ilustración 15 - Captura de JSON de entrada de datos

```
                    {
                        "frameSize": "7675",
                        "frameType": "Non-I"
                    },
                    {
                        "frameSize": "6918",
                        "frameType": "Non-I"
                    },
                    {
                        "frameSize": "24184",
                        "frameType": "Non-I"
                    },
                    {
                        "frameSize": "13474",
                        "frameType": "Non-I"
                    },
                    {
                        "frameSize": "6359",
                        "frameType": "Non-I"
                    },
                    {
                        "frameSize": "8305",
                        "frameType": "Non-I"
                    },
                    {
                        "frameSize": "13974",
                        "frameType": "Non-I"
                    },
                    {
                        "frameSize": "17446",
                        "frameType": "Non-I"
                    },
                    {
                        "frameSize": "7933",
                        "frameType": "Non-I"
                    },
                    {
                        "frameSize": "5646",
                        "frameType": "Non-I"
                    }
                ],
                "resolution": "1920x1080",
                "start": 15.48
            }
        ],
        "streamId": 42
    },
    "I23": {
        "stalling": [],
        "streamId": 42
    },
    "IGen": {
        "device": "pc",
        "displaySize": "1920x1080",
        "viewingDistance": "150cm"
    }
}
```


Ilustración 16 - Captura de JSON de entrada de datos

3.6.1 Funciones

- **dumps**

Argumentos:

obj {dict} -- Indica el objeto JSON al que hace referencia la función.
indent {int} -- Indica el tamaño de sangría para mostrar el contenido del objeto.

Método que muestra el contenido de un objeto JSON como una línea de texto en Python. Con este método mostramos el contenido en el terminal, para posteriormente analizarlo.

- **load**

Argumentos:

fp {a.read()} -- Dirección del archivo a leer.

Lee un archivo de texto o binario, que contiene un documento JSON, y lo convierte en un objeto de Python.

3.7 Librería TkInter

La librería TkInter (Tk Interface) es el estándar de herramientas para el uso de Tk GUI en Python. Tk por sí solo no pertenece a Python, es mantenido por ActiveState, pero TkInter ya es parte de las librerías estándar de Python. Su función es la de administración de ventanas para Interfaces Gráficas de Usuario (GUI). En nuestro caso es una parte fundamental para la selección del modo de funcionamiento, además de la introducción de datos, en caso de no contar con una interfaz gráfica habría que modificar el script proyecto.py para cada evaluación distinta. (19)

Ilustración 17 - Logotipo de TkInter

3.7.1 Funciones

- **Tk()**

Sin argumentos.

Crea una instancia Tk, e inicializa al interpretador creando una ventana root sobre la que trabajar.

- **title**

Argumentos:

text {str} -- Texto a representar.

Añade un texto (*text*) como título de la ventana root.

- **geometry**

Argumentos:

text {str} -- Indica las dimensiones en píxeles de la ventana root, por ejemplo '200x150'.

Dimensiona la ventana root con las medidas de *text*.

- **mainloop()**

Sin argumentos.

Función que bloquea la continuidad del script y lo mantiene fijo en la instancia de TkInter hasta que se cierre la ventana root.

- **destroy()**

Sin argumentos.
Función que cierra la ventana root para dar continuidad al script.

- **Label**

Argumentos:

master {Tk} -- Indica la ventana de TkInter a la que hace referencia el elemento.
text {str} -- Indica el texto que se muestra en el botón de tipo radio.

Elemento de información que aparece en la ventana *master* con un texto (*text*).

- **Button**

Argumentos:

master {Tk} -- Indica la ventana de TkInter a la que hace referencia el elemento.
text {str} -- Indica el texto que se muestra en el botón de tipo radio.
command -- Indica la función de Python a ejecutar cuando se pulsa el botón.

Elemento de acción que aparece en la ventana *master* con un texto (*text*) sirve para accionar una función (*command*) al ser pulsado.

- **Radiobutton**

Argumentos:

master {Tk} -- Indica la ventana de TkInter a la que hace referencia el elemento.
text {str} -- Indica el texto que se muestra en el botón de tipo radio.
variable {var} -- Variable asociada al valor del botón de tipo radio.
value {int} -- Valor asociado a la variable del botón de tipo radio.
command -- Indica la función de Python a ejecutar cuando se pulsa el botón.

Elemento de selección que aparece en la ventana *master* con un texto (*text*) asociado a una *variable* cuyo valor (*value*) depende del botón de tipo radio seleccionado entre los botones con la misma variable. También puede accionar una función (*command*) al ser pulsado.

- **grid**

Argumentos:

row {int} -- Indica la fila en la que se posiciona el elemento.
column {int} -- Indica la columna en la que se posiciona el elemento.
pady {int} -- Indica el margen respecto al eje y, en su respectiva fila y columna.

Esta función es llamada por los distintos elementos dentro de una ventana para posicionarse respectivamente entre ellos como si fuera una tabla con distintas filas (*row*) y columnas (*column*), si no utilizan esta función no serán mostrados en la ventana root.

- **filedialog.askopenfilename**

Argumentos:

title {str} -- Indica el título de la ventana de selección de archivo.

Función que crea una ventana nueva encima de la ventana root con un selector de archivo, y al seleccionar un archivo devuelve la dirección completa del mismo.

3.8 Implementación del código

En este apartado se detalla el funcionamiento del código *Proyecto.py* que hace uso de las librerías *pyshark*, *ffmpeg*, *json*, *itu_p1203*, *matplotlib* y *TkInter*.

El script de Python está estructurado con una función main() que va llamando a las diferentes funciones necesarias para cada parte, empezando por la función box() que selecciona el tipo de funcionamiento y la ruta a seguir, cada ruta llama a las funciones necesarias para llegar al resultado final, el cual es el resultado mostrado de forma gráfica.

3.8.1 Funciones

- **box()**

Sin argumentos.

Función encargada de crear una interfaz gráfica para poder seleccionar el tipo de entrada de datos: captura de tráfico, o documento JSON. Se selecciona el modo de funcionamiento para la captura de datos, y también se seleccionan los archivos utilizados en los distintos tipos.

Para realizar este proceso de forma ágil se hace uso de la librería TkInter, con elementos tales como botones de tipo radio (*RaddioButton*) para seleccionar características o ventanas de selección de archivos (*filedialog.askopenfilename*).

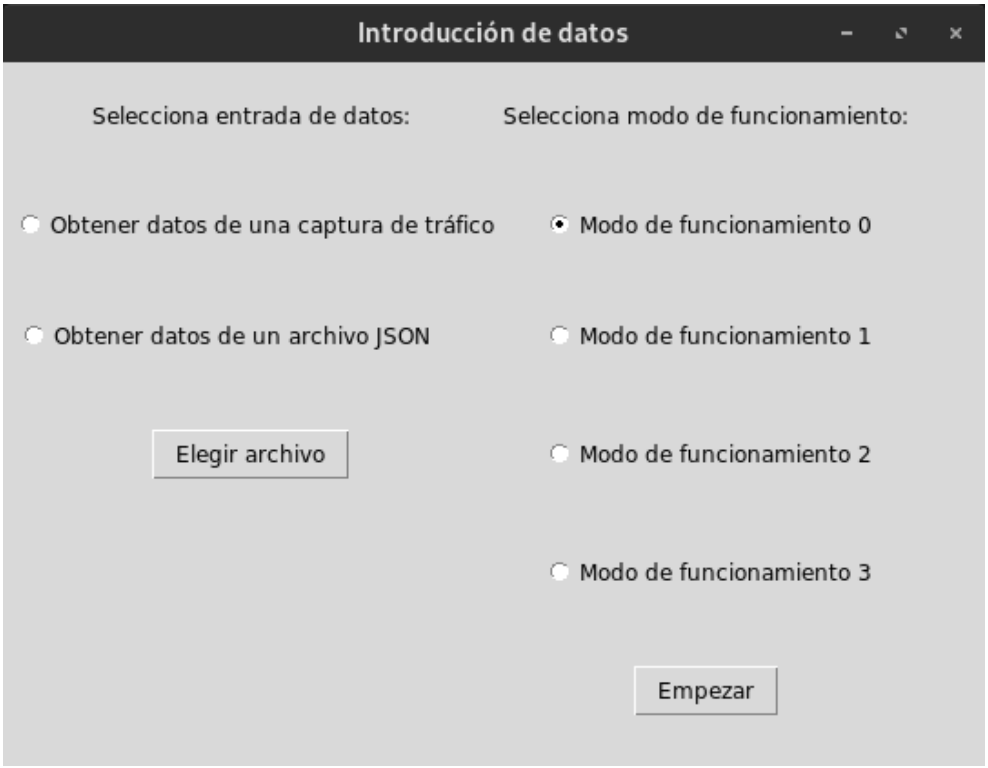

Ilustración 18 - Captura de TkInter de introducción de datos

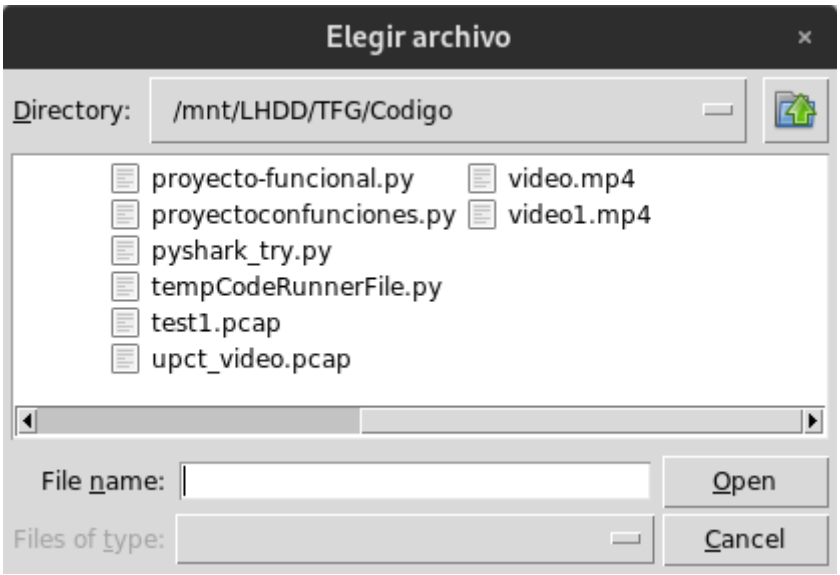

Ilustración 19 - Captura de TkInter de selección de archivo

- **captura_video**

Argumentos:

input {str} -- Indica la dirección de la captura de tráfico de la que obtener el archivo de video.

filter {str} -- Indica el filtro utilizado en Wireshark para filtrar los paquetes a solo los de video, por defecto: 'http.content_type contains "video"'.

length_max {int} -- Indica cuántos bytes se comprueban dentro de la captura de vídeo, por defecto: 9999.

Esta función empieza recolectando los paquetes de video dentro de una captura de tráfico, para ello hace uso de la librería Pyshark y el método *FileCapture*, continúa uniendo las distintas partes sin comprimir, con un bucle que recorre hasta *length_max*. Cuando ya se han unido todas las partes se pasan los datos de hexadecimal a binario y se crea un archivo "video.mp4" con la información obtenida.

- **captura_audio**

Argumentos:

input {str} -- Indica la dirección de la captura de tráfico de la que obtener el archivo de audio.

filter {str} -- Indica el filtro utilizado en Wireshark para filtrar los paquetes a solo los de video, por defecto: 'http.content_type contains "audio"'.

length_max {int} -- Indica cuántos bytes se comprueban dentro de la captura de vídeo, por defecto: 9999.

Esta función empieza recolectando los paquetes de audio dentro de una captura de tráfico, para ello hace uso de la librería Pyshark y el método *FileCapture*, continúa uniendo las distintas partes sin comprimir, con un bucle que recorre hasta *length_max*. Cuando ya se han unido todas las partes se pasan los datos de hexadecimal a binario y se crea un archivo "audio.mp3" con la información obtenida.

- **mix_video_audio**

Argumentos:

video_file {str} -- Indica la dirección del archivo de video de entrada, por defecto: 'video.mp4'.

audio_file {str} -- Indica la dirección del archivo de audio de entrada, por defecto: 'audio.mp3'.

output_file {int} -- Indica la dirección del archivo de salida, por defecto: 'output.mp4'.

Esta función une las partes de audio (*audio_file*) y vídeo (*video_file*) previamente creadas y crea un archivo de salida (*output_file*), para ello hace uso de la librería *ffmpeg*, empieza añadiendo las partes al proceso y luego las une con las especificaciones seleccionadas, en nuestro caso *-shortest* se queda con la parte más corta como tiempo máximo, *vcode='copy'* copia la codificación del video para no hacer recodificación, y la opción *-y* indica que se sobrescribe la salida, por si se van a hacer varias pruebas, no crear distintos archivos.

- **extract_info**

Argumentos:

modo {int} -- Indica el modo de extracción de datos del video.

segment {str} -- Indica la dirección del archivo de entrada, por defecto: output.mp4'.

Función que utiliza el script extractor.py de la librería *p_1203* y devuelve un objeto JSON con la información de un video (*segment*) según el *modo* seleccionado, para posteriormente poder calcular su MOS.

- **read_json**

Argumentos:

path {str} -- Indica la dirección del archivo JSON de entrada, previamente seleccionado con la interfaz gráfica.

Función simple que obtiene la dirección de archivo JSON (*path*) y devuelve el objeto JSON de Python correspondiente usando la función *json.load(f)*.

- **calculate_data**

Argumentos:

json_data {dict} -- Indica el objeto JSON de entrada con la información del video.

Función que realiza los cálculos a través de la librería *p_1203* usando la función:
P1203Standalone(json_data).calculate_complete(print_intermediate=True).
Y guarda el resultado en un objeto JSON, que muestra por consola.
Devuelve el objeto JSON con los resultados.

- **print_plot**

Argumentos:

data {dict} -- Indica el objeto JSON de entrada con los resultados obtenidos por *calculate_data*.

Función que obtiene los datos (*data*) de la función *calculate_data* y los separa en diferentes valores los cuales irá representando en dos gráficas de barras, en la primera gráfica tenemos las puntuaciones de: Calidad de parada, Puntuación audiovisual general y la Puntuación de calidad general, en la segunda gráfica tenemos las Puntuaciones audiovisuales generales por segundo.

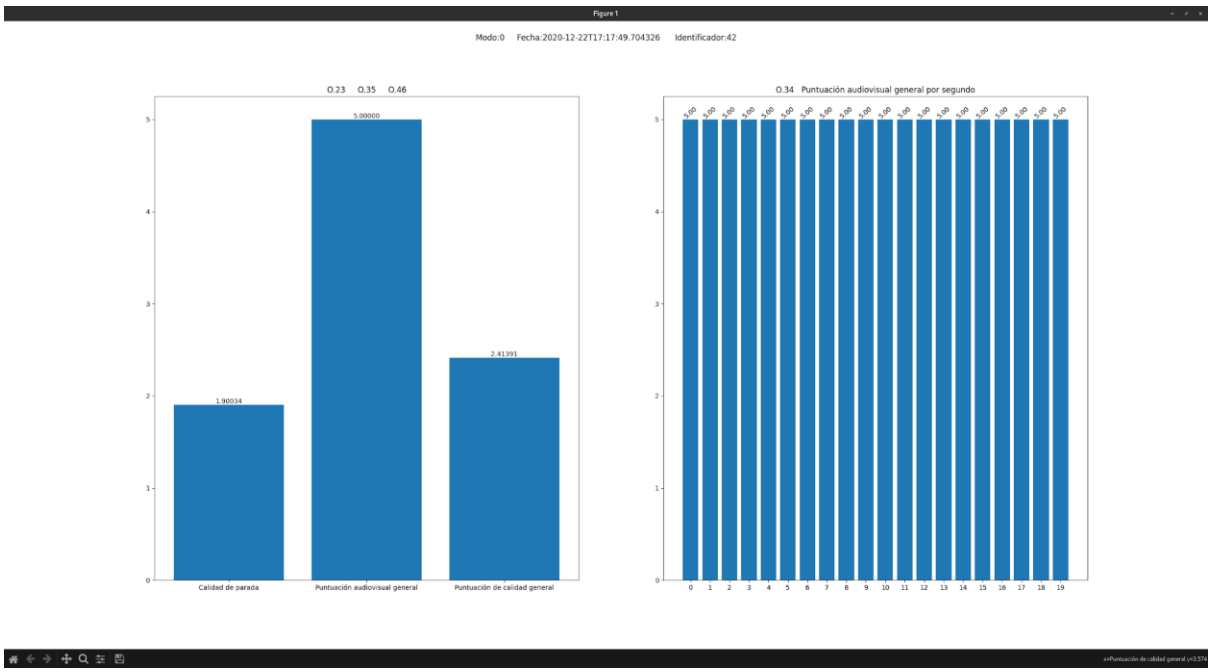

Ilustración 20 - Captura de salida de datos modo 0

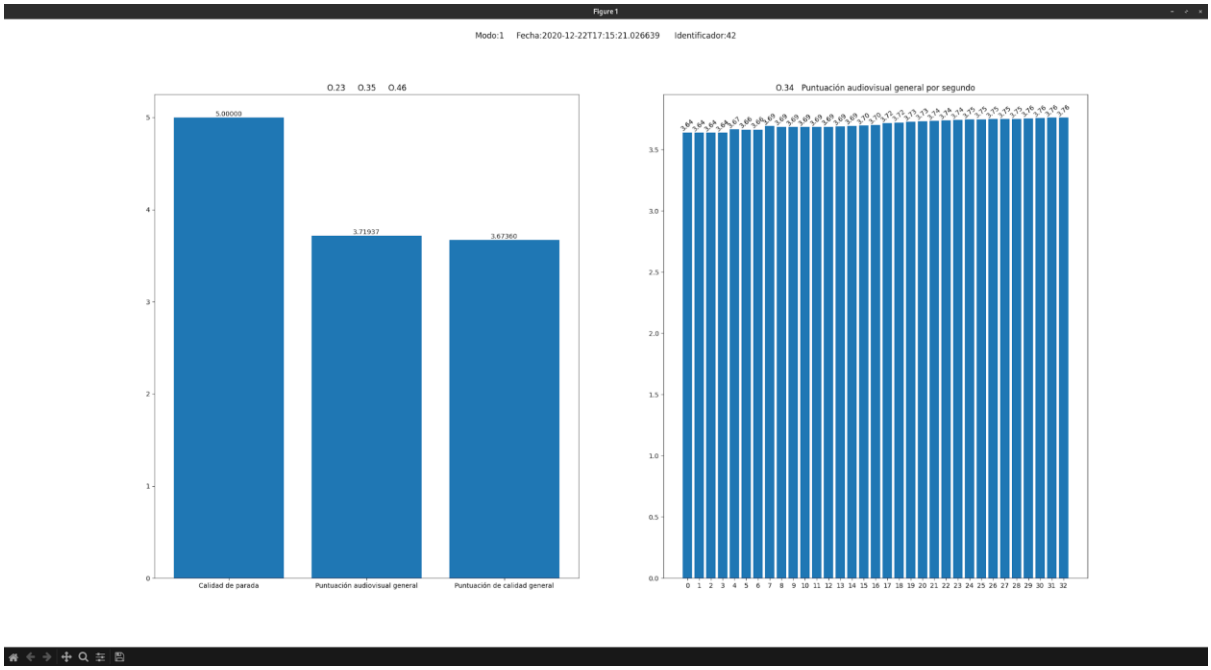

Ilustración 21 - Captura de salida de datos modo 1

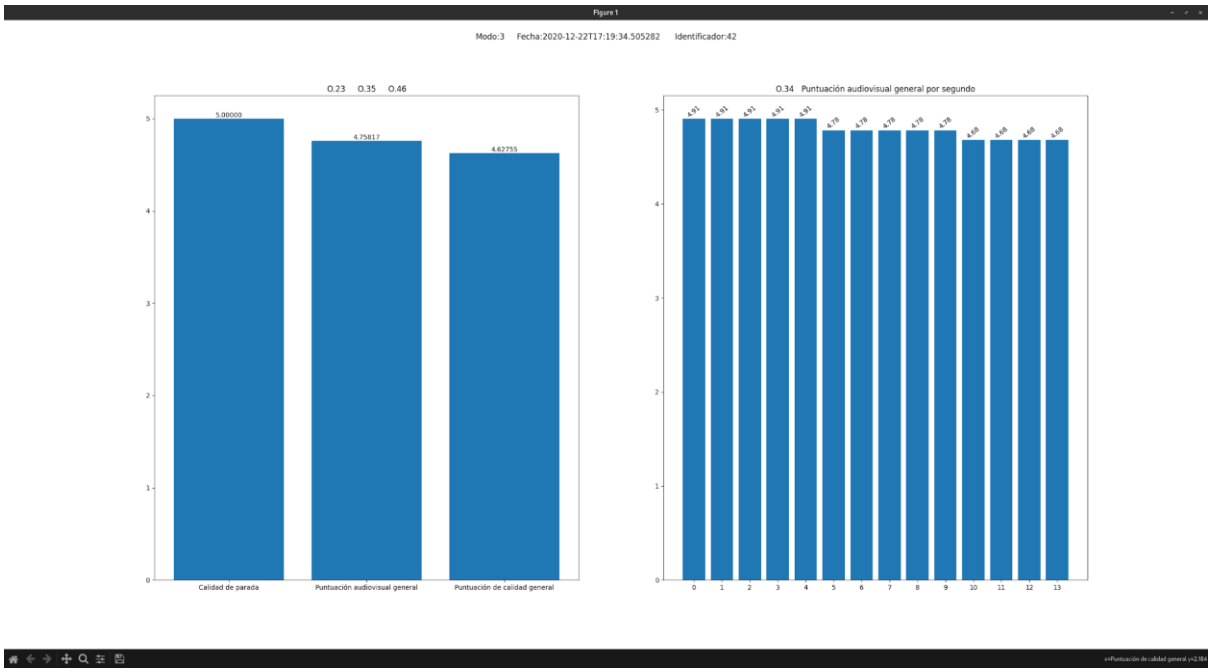

Ilustración 22 - Captura de salida de datos modo 3

- **main()**

Sin argumentos.

Esta función permite ejecutar el script *proyecto.py* desde la línea de comandos, funciona simplemente ejecutándolo con Python.

Anexo - Captura del tráfico

Aquí voy a explicar el procedimiento utilizado para capturar el tráfico, en nuestro caso vamos a utilizar el programa *Wireshark* que funcionará como *sniffer* (Programa para analizar datos de red), el navegador Firefox, una maquina con Linux Arch como Sistema Operativo y vamos a obtener un video de YouTube.

El primer problema al que nos enfrentamos es la encriptación TLS a la que están sometidos los datos del navegador, para poder desencriptar los datos necesitamos una clave, esta clave se almacena en la variable de entorno dinámica *SSLKEYLOGFILE* y lo primero que hay que hacer es guardar esta variable en un archivo de registro.

export SSLKEYLOGFILE=/mnt/LHDD/TFG/tcpdump/sslkeylog.log

A continuación, tenemos que indicar a *Wireshark* que tenemos la clave para desencriptar TLS (Transport Layer Security), vamos a Editar>>Preferencias.

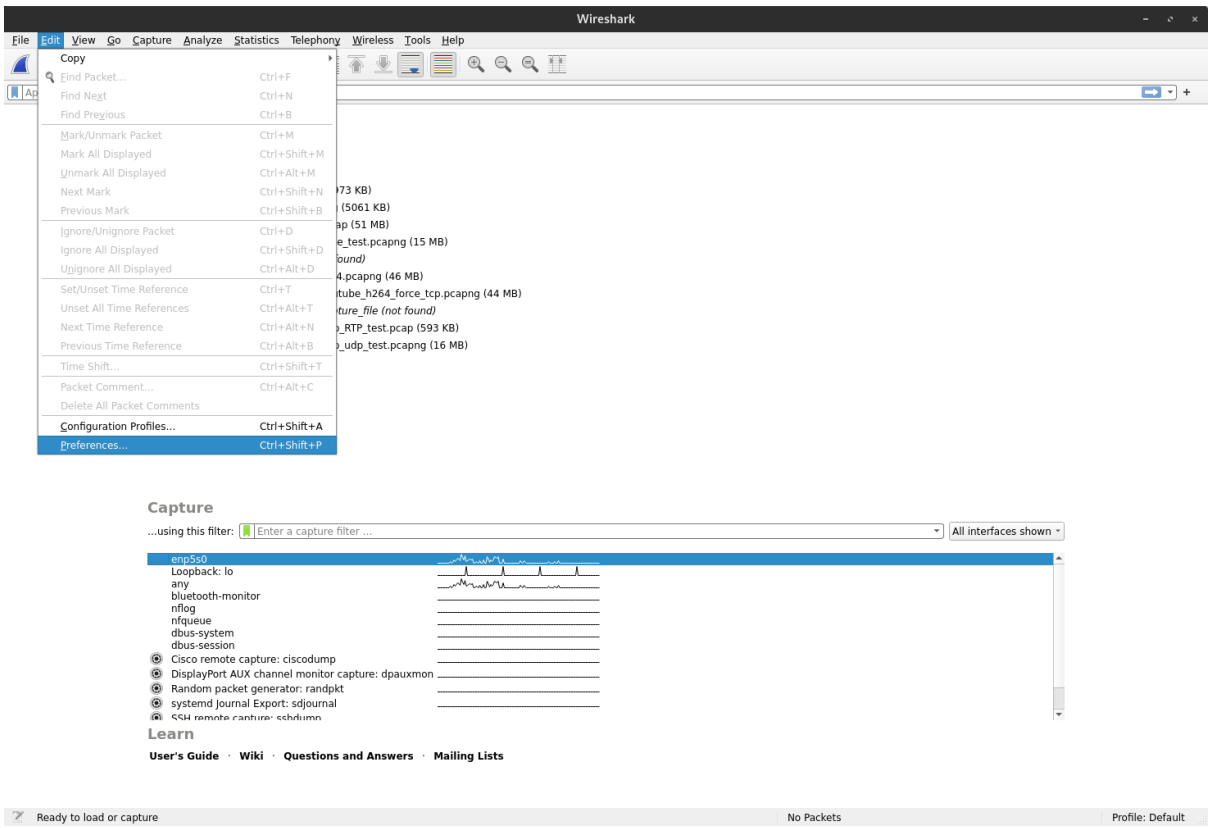

Ilustración 23 - Captura de Wireshark (Preferencias)

Entramos en las preferencias y nos vamos al apartado Protocolos>>TLS (En versiones anteriores es Protocolos>>SSL) y seleccionamos el archivo con la clave que previamente hemos guardado.

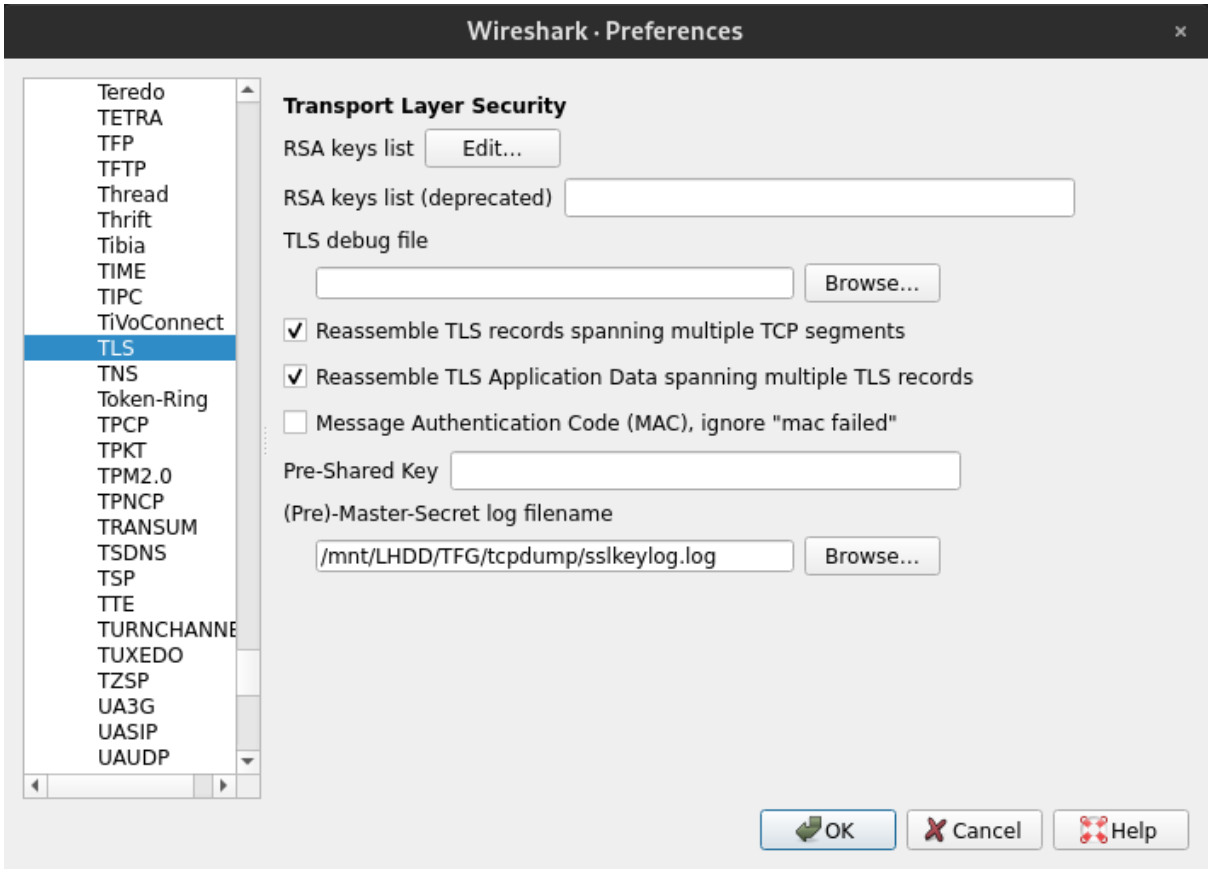

Ilustración 24 - Captura de Wireshark (TLS)

Ahora ya podemos capturar el tráfico y guardarlo en un archivo transportable.

Para ello seleccionamos la interfaz que queremos analizar y pulsamos el botón de captura. Mientras *Wireshark* captura el tráfico abrimos el navegador y visualizamos un video de *YouTube*. Cuando queramos parar simplemente nos vamos a Wireshark y paramos la captura de tráfico, no hace falta que termine el video, por último, guardamos el contenido de la captura de tráfico en un archivo con formato *".pcap"*.

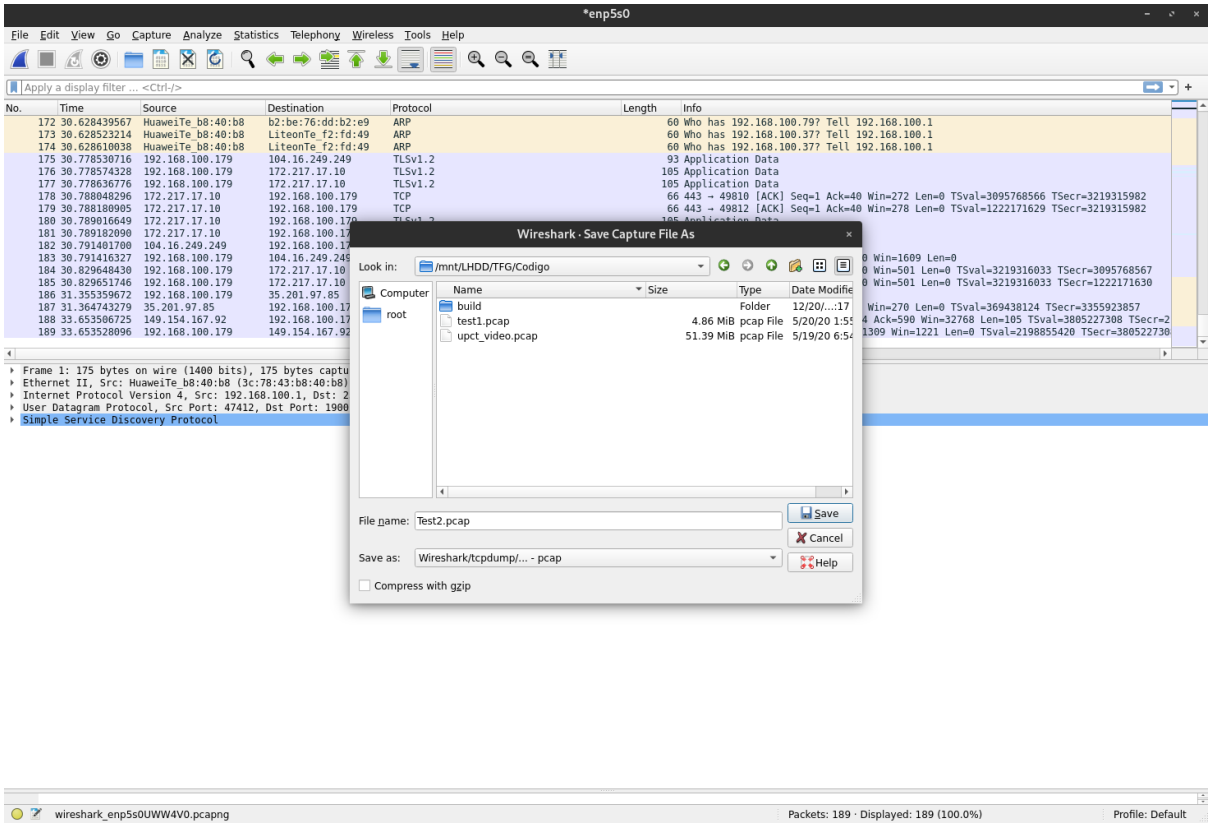

Ilustración 25 - Captura de Wireshark (Guardado)

Capítulo 4 - Conclusiones y líneas futuras

4.1 Conclusiones

Tras trabajar en este proyecto me he dado cuenta de la importancia que tiene realizar pruebas de calidad y en concreto pruebas de calidad de experiencia, las cuales al principio no tenía una idea definida de su objetivo y menos aún de su funcionamiento, y he podido comprobar de primera mano como es un tema de suma importancia.

La medición de los niveles de calidad, en concreto de los servicios multimedia sobre la red, se desarrollan en paralelo junto a las nuevas tecnologías de visualización, transporte y compresión de estos servicios, que indudablemente siguen en continua evolución. Los usuarios son cada vez más exigentes y existe mucha competencia, solo mediante mediciones podemos comparar las distintas tecnologías en diferentes situaciones y comprender donde puede mejorar cada una para poder seguir avanzando hacia un mejor servicio. La ventaja que nos brinda la calidad de experiencia es que se valora el resultado, en cuanto como influye en el usuario, y no como datos que no siempre tienen correlación con la experiencia final ofrecida, y por lo tanto es mucho más precisa en su objetivo.

En este proyecto me he centrado en adaptar la recomendación de la International Telecommunication Union (ITU) sobre el cálculo de la experiencia de visionado de los usuarios, en un programa funcional basado en Python. Para ello tuve que empezar con un análisis completo de la recomendación, para posteriormente poder implementar el código que devuelve la puntuación de la experiencia de visionado. El código hace uso de distintos frameworks para poder lograr su objetivo, desde trabajar con capturas de Wireshark, la fácil introducción de datos o mostrar un resultado visual de las puntuaciones. Hay que hacer especial mención a la librería itu_p1203 que es la encargada de adaptar los cálculos propuestos en la recomendación de la ITU a Python, y por lo tanto también tiene un análisis exhaustivo, para ver como realiza esta implementación.

4.2 Líneas futuras

Para finalizar voy a exponer las posibles líneas de desarrollo que puede seguir este proyecto, según lo aprendido durante la creación de este, haciendo hincapié en lo fácil que sería utilizarlo como base para otros proyectos, debido a la extensa documentación y la modularidad del código.

Primero de todo creo que sería interesante explorar la posibilidad de que el programa capture tráfico en tiempo real pues, aunque no resultaría una ventaja en la obtención de resultados, si que unificaría todo el proceso en un único programa.

Para ello lo más factible a primera vista sería utilizar las funciones de LiveCapture de Tshark, teniendo que ajustar los distintos parámetros para ajustarlos al modo de funcionamiento deseado.

Otra posibilidad para ampliar el rango de funcionamiento sería la obtención y lectura de documento DASH de forma automática, pues estos documentos contienen la información necesaria para la utilización de los modos 0 y 1 de forma inmediata.

El problema reside en la obtención de estos documentos, pues no son de fácil acceso al estar encriptados, y además no son proporcionados por los servicios de streaming, a menos que sean específicamente pedidos.

Finalmente, se podrían ajustar las funciones iniciales para trabajar con un mayor rango de servicios de Streaming, pues como pude comprobar al principio de este proyecto cada uno tiene unas particularidades que tienen que ser atendidas de forma individual, especialmente en el proceso de captura de los datos. Una vez obtenidos los datos, el programa funciona de igual manera para los distintos servicios de Streaming.

Bibliografía

Todos los recursos en línea han sido consultados en la fecha 06/02/2020.

1. **Prange, Stephanie.** mediaplaynews. [Online] Enero 23, 2019. https://www.parksassociates.com/report/ott-video-services-globalization.
2. **Spangler, Todd.** variety. [Online] Abril 21, 2020. https://variety.com/2020/digital/news/netflix-record-16-million-subscribers-q1-2020-coronavirus-1234586125/.
3. **Sweney, Mark.** Netflix to slow Europe transmissions to avoid broadband overload. *The Guardian.* [Online] Marzo 19, 2020. https://www.theguardian.com/media/2020/mar/19/netflix-to-slow-europe-transmissions-to-avoid-broadband-overload.
4. **Sandvine.** Sandvine. [Online] https://www.sandvine.com/hubfs/downloads/archive/whitepaper-video-quality-of-experience.pdf.
5. **Wikipedia**. *ITU-T.* [Online] https://en.wikipedia.org/wiki/ITU-T.
6. **ITU-T**. *Parametric bitstream-based quality assessment of progressive download and adaptive audiovisual streaming services over reliable transport.* 2017. P.1203 .
7. **ITU-T**. *Parametric bitstream-based quality assessment of progressive download and adaptive audiovisual streaming services over reliable transport – Video quality estimation module.* 2017. P.1203.1.
8. **ITU-T**. *Parametric bitstream-based quality assessment of progressive download and adaptive audiovisual streaming services over reliable transport – Audio quality estimation module.* 2017. P.1203.2.
9. **ITU-T**. *Parametric bitstream-based quality assessment of progressive download and adaptive audiovisual streaming services over reliable transport – Quality integration module.* 2019. P.1203.3.
10. **Wikipedia**. *Python (programming language).* [Online] https://en.wikipedia.org/wiki/Python_(programming_language).
11. **Python**. [Online] https://www.python.org/.
12. **Github**. *itu-p1203.* [Online] https://github.com/itu-p1203/itu-p1203.
13. *A bitstream-based, scalable video-quality model for HTTP adaptive streaming: ITU-T P.1203.1.* Raake, Alexander , et al. Erfurt : IEEE, Mayo 2017, Ninth International Conference on Quality of Multimedia Experience (QoMEX). 978-1-5386-4024-1.
14. *HTTP Adaptive Streaming QoE Estimation with ITU-T Rec. P.1203 – Open Databases and Software.* Robitza, Werner , et al. Amsterdam : s.n., 2018, 9th ACM Multimedia Systems Conference. 9781450351928.
15. **KimiNewt**. Github. *Pyshark.* [Online] https://github.com/KimiNewt/pyshark.
16. **FFmpeg**. FFmpeg. [Online] https://ffmpeg.org/about.html.
17. **Matplotlib**. Matplotlib. [Online] https://matplotlib.org/.
18. **JSON**. JSON. [Online] https://www.json.org/json-en.html.
19. **Pyhton**. *Tkinter.* [Online] https://docs.python.org/3/library/tkinter.html.