UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería de Telecomunicación

Modelos de tráfico streaming multimedia e impacto en consumo energético en terminales móviles.

Trabajo Fin de Grado

Grado en Tecnología Digital y Multimedia

AUTOR/A: Fernández Picazo, Julio

Tutor/a: Giménez Guzmán, José Manuel

CURSO ACADÉMICO: 2022/2023

Resumen

En la actualidad, el tráfico streaming multimedia ya representa la mayoría de tráfico que se cursa en las redes de acceso inalámbricas. Por ese motivo, su estudio es clave para el desarrollo de las redes de comunicaciones. En este trabajo, se va estudiar el perfil de tráfico de diferentes flujos de video en streaming muy populares para su caracterización y su comprensión. Para ello, se realizarán capturas de tráfico real fruto de dichas aplicaciones y posteriormente se automatizarán las tareas necesarias para extraer la información temporal de dicho tráfico, tanto uplink como downlink.

Por otro lado, uno de los pilares en los que debe basarse nuestro desarrollo es el de la sostenibilidad. En ese sentido, el consumo energético se está convirtiendo en un parámetro de mérito clave para el desarrollo de las futuras redes de comunicaciones, en cuanto a que cada vez es mayor la energía que se requiere en las redes y terminales celulares para cursar todo el tráfico que se les demanda. En este trabajo, se emplearán los modelos de tráfico desarrollados para el tráfico streaming multimedia para evaluar el consumo energético que requieren por parte de los dispositivos.

Palabras clave: modelizado de tráfico; streaming; consumo energético; redes inalámbricas.

Resum

Actualment, el trànsit streaming multimèdia ja representa la majoria de trànsit que es cursa a les xarxes d'accés sense fil. Per això, el seu estudi és clau per al desenvolupament de les xarxes de comunicacions. En aquest treball, s'estudiarà el perfil de trànsit de diferents fluxos de vídeo en streaming molt populars per a la seva caracterització i comprensió. Per fer-ho, es realitzaran captures de trànsit real fruit d'aquestes aplicacions i posteriorment s'automatitzaran les tasques necessàries per extreure la informació temporal del trànsit, tant uplink com downlink.

D'altra banda, un dels pilars en què s'ha de basar el nostre desenvolupament és el de la sostenibilitat. En aquest sentit, el consum energètic s'està convertint en un paràmetre de mèrit clau per al desenvolupament de les futures xarxes de comunicacions, pel fet que cada vegada és més gran l'energia que es requereix a les xarxes i terminals cel·lulars per cursar tot el trànsit que se'ls demana. En aquest treball, es faran servir els models de trànsit desenvolupats per al trànsit streaming multimèdia per avaluar el consum energètic que requereixen per part dels dispositius.

Paraules clau: modelitzat de trànsit; streaming; consum energètic; xarxes sense fil.

Abstract

Nowadays, multimedia streaming traffic already represents the majority of traffic carried on wireless access networks. For this reason, its study is crucial for the development of communication networks. In this work, we will study the traffic profile of different very popular streaming video flows in order to characterize and understand them. For this purpose, we will capture real traffic from these applications and then we will automate the necessary tasks to extract the temporal information of this traffic, both uplink and downlink.

On the other hand, one of the pillars on which our development must be based is sustainability. In this regard, energy consumption is becoming a key parameter of merit for the development of future communications networks, since the energy required by cellular networks and terminals to handle all the traffic demanded of them is increasing. In this work, traffic models developed for multimedia streaming traffic will be used to evaluate the energy consumption required by terminals.

Key Words: traffic modeling; streaming; energy consumption; wireless networks

Agradecimientos

Este trabajo se ha desarrollado durante meses con el apoyo y dedicación constante de mi tutor, José Manuel Giménez Guzmán. Sin su visión, conocimientos e inconmensurable ayuda, la realización de este hubiera sido imposible. Por ello quisiera darle las gracias en primer lugar.

A todos aquellos amigos y compañeros, de la universidad y de mi etapa de prácticas, que han compartido ideas, reflexiones y toda la experiencia de crecimiento y madurez que ha supuesto en mi vida estos 4 años, fundamentalmente aquellos que me han animado y soportado en el final de los estudios y en el desarrollo del trabajo. A todos ellos, mil gracias.

Finalmente, agradecer a mi familia. Su amor ha servido de motivación y ánimo para logar un hito en mi carrera como estudiante como lo es la consecución de este Trabajo de Fin de Grado.

Índice

Capítulo 1. Introducción

En la última década, el mundo ha sido testigo de una revolución tecnológica sin precedentes en la forma en que consumimos contenido multimedia. Las redes inalámbricas y los servicios de streaming de contenido multimedia han experimentado un crecimiento exponencial, llevando a una mayor demanda de recursos y una mayor dependencia de la conectividad digital. Sin embargo, este auge se ha visto aún más acelerado de manera inesperada por la pandemia de COVID-19, que ha cambiado drásticamente nuestros hábitos de consumo de medios, haciendo que servicios como Netflix, YouTube, Prime Video y Twitch, entre otros, sean protagonistas.

En este contexto tecnológico en constante evolución, surge la necesidad de comprender en profundidad el tráfico generado por estos servicios de streaming y su impacto energético en los dispositivos conectados a las redes que los albergan. En este Trabajo de Fin de Grado, se explorará cómo abordar esta necesidad mediante la combinación de herramientas como Wireshark y Python, que permiten tanto la creación de modelos del tráfico capturado personalizados como la caracterización de su impacto energético. Al igual que la tecnología detrás de las distintas plataformas de streaming de contenidos multimedia sigue en constante crecimiento, el consumo energético como parte del desarrollo sostenible de las redes inalámbricas del futuro se está convirtiendo en un parámetro trascendental. Por ello se pretende analizar también la repercusión del consumo de recursos por parte de los servicios de streaming en lo que se conoce como equipos de usuario (UEs), para valorar si estos realizan un uso eficiente de estos o no.

Además del exhaustivo resumen del contexto tecnológico, en este documento se presentarán, por un lado, las distintas metodologías utilizadas durante el desarrollo, por otro, las distintas soluciones Python que permiten automatizar la generación de los modelos de tráfico para distintos servicios de streaming multimedia, y la caracterización de su impacto energético, explorando distintos patrones de tráfico junto a una breve investigación sobre el ahorro energético llevado a cabo.

Por último, se expondrán todos los resultados y las conclusiones derivadas de esta investigación. A lo largo de este memoria, se desentrañarán las complejidades del tráfico de streaming multimedia y se revelarán sus implicaciones en la era digital actual.

1.1 Objetivos

A continuación, considero necesario dejar constancia de los objetivos a alcanzar en este proyecto:

- Realizar capturas de tráfico de los distintos servicios de streaming mediante el software de Wireshark.
- Automatizar, mediante el lenguaje de programación *Python*, las tareas necesarias para extraer la información relevante de estas capturas de tráfico.
- Mediante Python también, crear modelos de representación de este tráfico capturado para lograr su compresión.
- Finalmente, lograr una caracterización del impacto energético del tráfico capturado en dispositivos conectados a redes inalámbricas, logrando extraer resultados y conclusiones relativas a la eficiencia energética de los servicios de streaming analizados.

Capítulo 2. Contexto tecnológico

Como primer paso dentro del desarrollo del trabajo, se ha realizado un breve estudio con el objetivo de entender mejor el contexto tecnológico en torno al tráfico multimedia en las redes actuales. Concretamente, se pretende descubrir la repercusión del COVID en los hábitos de consumo de la sociedad como punto de inflexión en el avance y desarrollo de las redes que soportan el tráfico multimedia, encontrar sus principales características, las tecnologías utilizadas en su codificación, además del consumo energético que se genera en consecuencia en los distintos terminales móviles en los que generalmente ocurre.

2.1 - Repercusión de la pandemia mundial en el tráfico de la red.

Desde la llegada del COVID-19 a nuestras vidas en el pasado 2020, la forma de vivir, comunicarse y relacionarse de millones de personas ha cambiado radicalmente, dependiendo en gran parte de las distintas herramientas y servicios de Internet. El streaming de vídeo y audio en plataformas de creación de contenido de entretenimiento, los videojuegos, redes sociales, videollamadas, mensajería y chat, entre otras categorías de tráfico, han sufrido un crecimiento exponencial debido a la adopción de nuevos hábitos que implican su uso.

Como parte de la cotidianidad, se ha asumido la modalidad online de compras de productos (comida, ropa, electrónica, medicamentos), docencia, trabajo desde casa, ver entretenimiento en directo, la gestión de trámites administrativos a través de soluciones digitales, llamadas de vídeo con familiares y personal sanitario, etc. En definitiva, parte de la rutina que se nos planteó desde que surgió el COVID se ha continuado también en este año, contribuyendo a un aumento del volumen del tráfico del 23 % respecto al año anterior, tal y como se indica en el "*Sandvine´s Global internet phenomena report*" [1], a diferencia de lo que podría parecer lógico con la vuelta al contacto humano y al fin de restricciones.

En este caso, la vuelta al trabajo y a la escuela no implica una reducción del consumo en comparación a los años de pandemia puesto que ahora el consumo puede estar sucediendo en diferentes momentos del día, en diferentes lugares o diferentes redes (en casa, en la escuela o en el trabajo). En la figura de a continuación se muestra el volumen de tráfico generado por distintos servicios tras la llegada del COVID:

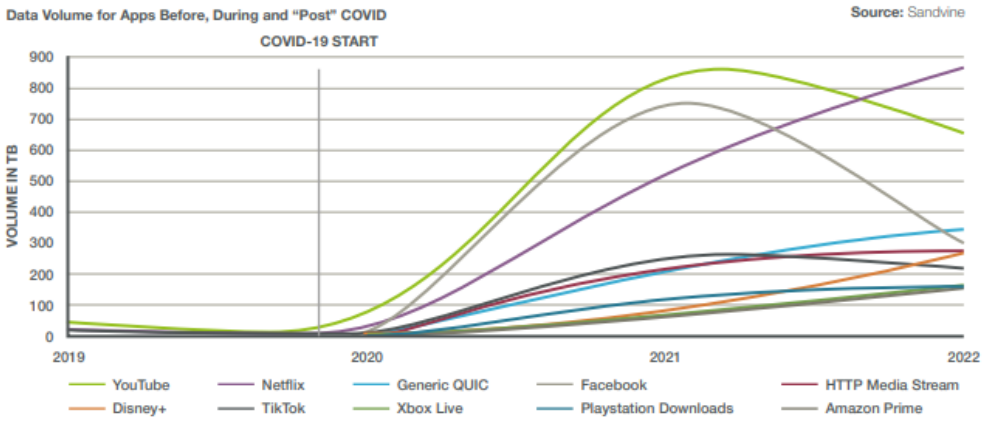


Figura 1. Volumen de datos generado por distintas Apps multimedia antes y después de la pandemia COVID-19. Fuente: [1]

2.2 – Netflix + MAMAA – El "Big 6" que domina el tráfico multimedia actual

A finales del año 2022 se observa que solo un grupo de empresas son responsables de generar casi la mitad del volumen del tráfico total de internet. Este grupo está conformado por un total de 6 empresas, denominadas por su relevancia como "*The Big 6*", siendo participantes de él Netflix y MAMAA, grupo de 5 empresas que ya dominaba el tráfico de internet en años anteriores. MAMAA es hoy en día un acrónimo en expansión, utilizado para representar de forma conjunta a los 5 gigantes tecnológicos más significativos: Microsoft, Alphabet (Google), Meta (anteriormente Facebook), Amazon y Apple.

El ya mencionado incremento en el volumen general del tráfico de internet en el último año se debe en gran parte al peso de estas grandes empresas, especialmente al de Netflix, cuyo aumento en tráfico generado en 2022 fue notablemente superior al resto. Sin embargo, es necesario resaltar que este imperio tecnológico ha sufrido un decrecimiento del 9% en el cómputo total del volumen del tráfico de internet generado respecto al año anterior debido entre otros factores a la diversificación del sector en un mayor número de aplicaciones y servicios.

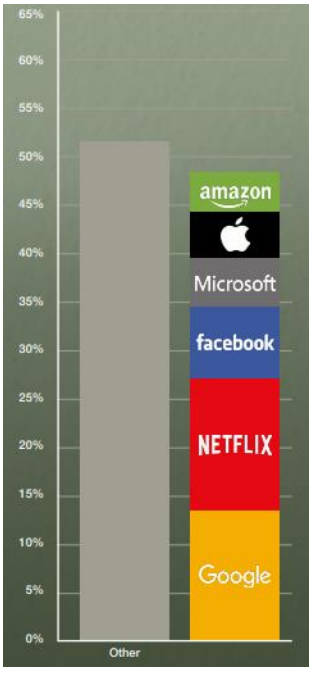


Figura 2. Big 6 vs Other. 2022. Fuente: [1]

Otras empresas como TikTok y Disney+, están aportando también un volumen considerable de tráfico, y nuevas alternativas a TCP como protocolo de transporte de internet por excelencia están cada día más presentes. En este caso, QUIC (*Quick UDP Internet Connections*) ha empezado a ser utilizado en aplicaciones de Google como YouTube, y ya goza de una trascendencia solo superada por el que de momento es el rey del tráfico de vídeo, Netflix, tal y como podemos observar en las figuras de a continuación:

TOP APPS 2022

UPSTREAM TRAFFIC

	Application	Total Volume
1	Netflix	8.78%
2	HTTP Media Stream	6.89%
3	YouTube	5.90%
4	Generic Web Browsing	3.85%
5	HTTP download	3.70%

TOP APPS 2022

DOWNSTREAM TRAFFIC

	Application	Total Volume
1	Netflix	14.93%
2	YouTube	11.62%
3	Generic QUIC	5.88%
4	Disney+	4.49%
5	TikTok	3.93%

Figura 3. Volumen de tráfico de subida y bajada generado por aplicaciones en 2022. Fuente: [1]

2.3 - Importancia y peso del vídeo y el multimedia en el tráfico de Internet.

La demanda de contenidos de vídeo se está disparando, y la mayoría de las plataformas están incrustando y difundiendo vídeos dentro de las demás aplicaciones para aumentar las visualizaciones y la participación, lo que se conoce como "*engagement*". El ya mencionado informe sobre los fenómenos de internet en el mundo [1], lanzado por la empresa *Sandvine* en enero de 2023, muestra que solo el tráfico de vídeo supone alrededor de un 66% del volumen del tráfico global. Por tanto, es imprescindible considerar la relevancia del vídeo y las plataformas que difunden tráfico de vídeo para el entendimiento del tráfico multimedia actual. En este caso "vídeo" incluye TV, vídeo bajo demanda (VoD – Video on Demand), y streaming de vídeo (en directo). A continuación, se mencionarán algunas de las tendencias de consumo de vídeo actuales:

APP CATEGORY TOTAL VOLUME

	2022 Categories	Total Volume
1	Video	65.93%
2	Marketplace	5.83%
3	Gaming	5.58%
4	Social Networking	5.26%
5	Cloud	4.98%
6	Web Browsing	4.63%
7	File Sharing	3.39%
8	Messaging	2.30%
9	VPN	1.13%
10	Audio	0.95%

2.3.1 - Carrera de Netflix contra Disney+ y otros servicios de vídeo:

Durante el pasado año las redes han estado sometidas a una gran presión por parte de los 223 millones de usuarios de pago de Netflix, quienes a fecha de hoy, consumen en torno a mil millones de horas de vídeo cada semana. En sus planes HD, cada vez más populares, los miembros pueden utilizar hasta 3 GB de datos cada hora, y hasta 7 GB para resolución 4K únicamente presente en los planes "premium".

Si indagamos más en las resoluciones del contenido consumido, encontramos por ejemplo que, según el Informe Anual de Internet de Cisco [2], un televisor HD con conexión a Internet en streaming, durante dos o tres horas puede recibir tanto tráfico de Internet como todo un hogar en un día. Las tasas de bits (*bitrates*) de la UHD (4K) serían el doble que las de la HD y 9 veces superiores a las de la SD.

Cisco [2] calcula que el 66% de los televisores de pantalla plana serán UHD en 2023, y que la velocidad media mundial de banda ancha fija pasará de 46 Mbps en 2022 a 110 Mbps en 2023, por lo que se espera incluso un crecimiento mucho mayor en el volumen del tráfico y en la demanda de ancho de banda para este año.

Respecto a las plataformas que distribuyen la mayoría del contenido multimedia, Disney+ por su cuenta, ha apostado a lo grande por un contenido para toda la familia. Gastándose alrededor de 33.000 millones de euros durante el trascurso del pasado 2022, ha generado un 4,20% del volumen del tráfico, un incremento del 199% respecto al año anterior. Con las franquicias de Star Wars, Marvel, Pixar, National Geographic y las historias de Walt Disney, Disney+ ha crecido rápidamente hasta alcanzar los 164,2 millones de suscriptores, o 235 millones si también se incluyen Hulu y ESPN+, que también forman parte de las franquicias integrantes de Disney+.

Por otro lado, Netflix, que está sintiendo el calor de Disney+, ha declarado que está gastando 13.600 millones de dólares en creación de contenido, cifra lejana al gasto de su mayor competidor pero que le permite mantenerse como líder en el cómputo de tráfico multimedia generado durante el año pasado. Las cifras, tanto en volumen de tráfico, como económicas, definitivamente denotan la fuerza que han adquirido estas dos empresas y consecuentemente la industria multimedia y audiovisual en el último par de años.

Finalmente, también encontramos como competidores directos a otros servicios de streaming de vídeo como Amazon Prime, que aún figura entre los 10 primeros en términos de volumen total (2,67%); y HBO Max, que no figura en esta lista. [1]

2.3.2 – La explosión del formato de vídeo "short" de TikTok.

Es cierto que en 2022 TikTok ha caído un puesto en la clasificación de volumen total de tráfico respeto al año anterior [1], pero su descenso se debe precisamente al aumento desmesurado que consiguió durante 2021, algo que no se esperaba que se mantuviera una vez que su base de suscriptores (principalmente gente joven) volviera a la escuela. Con un 3,55% del volumen total del tráfico, TikTok también está lidiando con la invasión de competidores que se dieron cuenta de su meteórico ascenso el año pasado. *YouTube Shorts* e *Instagram´s Reels* se disputan la posición en este mercado bastante nuevo.

Hasta ahora, sus diversas funcionalidades y su alcance la están convirtiendo en la aplicación más descargada del mundo, con más de mil millones de usuarios activos mensuales. Considerando que el tiempo medio de visualización de esta cantidad abrumadora de usuarios va de los 50 a los 90 minutos diarios en el caso de los más activos, podemos hacernos una idea de la inmensidad de contenido multimedia que circula en las redes procedente del consumo masivo de "*shorts*" o vídeos cortos, con una capacidad para viralizarse nunca vista hasta el momento.

2.3.3 – Las apps móviles que han revolucionado el tráfico multimedia.

Verdaderamente esta revolución parte del incremento del uso de terminales móviles y teléfonos inteligentes ya abordado en la sección "*2.1 - Repercusión de la pandemia mundial en el tráfico de la red"*, hasta un punto casi excesivo. Se observa que el tiempo dedicado a las apps móviles ha aumentado de las tres horas diarias antes de la pandemia, a un promedio global de aproximadamente 5 horas al día en la actualidad.

Según Ericsson [3], multinacional dedicada a ofrecer soluciones en telecomunicaciones, en 2028, todo el crecimiento en el tráfico de internet provendrá de la tecnología 5G. Este crecimiento del tráfico previsto incluye la hipótesis de que la adopción inicial de servicios de tipo XR (*Extended Reality*), como Realidad Aumentada, Realidad Virtual y Realidad Mixta, se producirá de forma más próxima a la fecha estimada, es decir 2028, sin embargo, si la adopción es más fuerte de lo esperado, el tráfico de datos podría aumentar significativamente más de lo previsto, en particular en la estimación de tráfico de subida. En la actualidad, se calcula que el tráfico de vídeo representa el 67% de todo el tráfico de datos móviles, y se prevé que este porcentaje aumente hasta el 80% en 2028 [3].

Como se puede observar en las figuras siguientes, YouTube (Google) es el principal responsable del tráfico de vídeo en terminales móviles, seguido por Meta (Facebook), debido en gran parte a la trascendencia de Instagram y Facebook Video; y las ya mencionadas TikTok y Netflix. Es importante resaltar también el peso del nuevo protocolo de transporte QUIC, ya que, incluso excluyéndolo de YouTube, las aplicaciones que hacen uso de él se mantienen dentro del top 5 aplicaciones con más volumen de tráfico. Esto se verá más adelante cuando se analicen las distintas tecnologías presentes en estos servicios de vídeo, incluyendo códecs y protocolos de transporte.

Además, las gráficas representan un cambio con respecto al año pasado, cuando el tráfico de Netflix no figuraba entre las 15 aplicaciones principales en términos de tráfico móvil. Esto indica que más personas están viendo contenido de Netflix en sus dispositivos móviles, en lugar de hacerlo únicamente en sus hogares en pantallas más grandes.

TOTAL MOBILE VOLUME			TOTAL MOBILE VOLUME		
	Category	Total Volume		Application	Total Volume
1	Video	67.60%	1	YouTube	16.24%
2	Social Networking	12.16%	2	Facebook Video	14.37%
3	Messaging	5.89%	3	TikTok	13.76%
4	Web Browsing	4.51%	4	Generic QUIC	9.49%
5	Marketplace	2.77%	5	Facebook	5.94%
6	Gaming	2.41%	6	Instagram	4.82%
7	File Sharing	1.97%	7	HTTP Media Stream	4.09%
8	Cloud	1.79%	8	WhatsApp	2.68%
9	VPN	0.79%	9	Netflix	2.41%
10	Audio	0.11%	10	Snapchat	1.60%

Figura 4.. Volumen total de tráfico móvil generado por categorías y aplicaciones. Fuente: [1]

Cabe destacar también el impacto de las redes sociales (12,16% del tráfico móvil total) y apps de mensajería (5,89%), como WhatsApp, Wattpad, Slack, Google Chat, Telegram, Discord. A medida que estas aplicaciones incorporan más capacidades de audio y vídeo, su contribución al volumen de tráfico aumentará.

2.4 - Características tecnológicas de los principales servicios de streaming multimedia:

Los servicios de streaming de contenido multimedia, como los ya mencionados Netflix, YouTube y Prime Video, hacen uso de una serie de tecnologías para ofrecer contenido de alta calidad de manera eficiente a sus usuarios. Algunas de las principales tecnologías son los códecs de compresión de vídeo, protocolos de transmisión adaptativa, redes de distribución de contenido (CDN - *Content Delivery Networks*), además de distintas resoluciones y formatos de vídeo, tal y como veremos a continuación:

2.4.1 – Resoluciones y formatos de vídeo:

Los servicios de streaming de contenido multimedia ofrecen una variedad de resoluciones y formatos de vídeo para adaptarse a diferentes dispositivos y velocidades de conexión a Internet. Las resoluciones que emplean estos servicios son los siguientes:

1. **Resolución estándar (SD)**: La resolución estándar generalmente se refiere a unas resoluciones de vídeo de 480p (720x480 píxeles) o 576p (720x576 píxeles). Es la opción más básica en términos de calidad de imagen y es adecuada para conexiones a Internet más lentas o dispositivos con pantallas más pequeñas.
2. **Alta definición (HD)**: La alta definición es una opción más nítida y de mayor calidad que ofrece una resolución de 720p (1280x720 píxeles. Proporciona una experiencia de visualización más envolvente y es ampliamente compatible con la mayoría de los dispositivos modernos y conexiones de banda ancha.
3. **Full HD (FHD)**: El Full HD es una versión mejorada de la alta definición y se refiere a una resolución de 1080p (1920x1080 píxeles). Ofrece una calidad de imagen más nítida y detalles más precisos, lo que lo convierte en una opción popular para televisores de pantalla grande y dispositivos de alta gama.
4. **Ultra HD (4K)**: El Ultra HD, también conocido como 4K, es la resolución más alta disponible en la mayoría de los servicios de streaming. Es una resolución de 3840x2160 píxeles y proporciona una calidad de imagen altamente superior con detalles extremadamente nítidos y colores vibrantes. Es ideal para televisores de pantalla grande y dispositivos que admiten esta resolución.

Por otra parte, un formato de vídeo es un archivo diseñado para contener audio y vídeo en una estructura definida por el desarrollador del mismo. Es por eso que esta estructura específica también se conoce con el nombre de contenedor de vídeo. El audio y vídeo puede ir o no comprimido, dependerá de su uso.

Cuando se va a utilizar el archivo para edición, lo mejor es mantener tanto el audio como el vídeo sin compresión. Pero cuando se utilizará para ser visto en internet, se deberá comprimir para que el archivo no sea demasiado grande.

Los formatos pueden ser de código abierto, es decir que cualquier sistema operativo o fabricante los puede utilizar, o bien cerrados que sólo podrán ser vistos en ciertos sistemas operativos o dispositivos. Aunque existe una gran variedad de formatos en el mercado, no todos son utilizados muy a menudo, dependerá del dispositivo en donde se visualice. En el caso de internet, lo que se busca es que sea visto en la mayor cantidad de teléfonos inteligentes, tabletas y navegadores. Para ello lo más recomendable es utilizar un formato estándar que sea soportado por la mayor cantidad de dispositivos [4]:

1. **MP4**: Es uno de los formatos más utilizado en internet, se utiliza para vídeo en definición estándar (SD) y alta definición (HD). Dentro de este formato se utiliza la compresión de vídeo H.264 y de audio AAC. Esto se verá más adelante cuando se analicen los distintos códecs de compresión.
2. **MKV**: Es un formato muy completo que puede servir, para vídeo/audio, subtítulos o sólo audio. Además de ser de código abierto puede ser integrado por cualquier fabricante de productos. Un archivo MKV puede contener compresión de vídeo VP8/VP9 y audio Vorbis.
3. **WebM**: Este formato en particular surge como alternativa a MP4 y es una versión recortada de MKV. También es un formato de código abierto que soporta compresión de vídeo VP8/VP9 y de audio Vorbis. Es compatible en la mayoría de los exploradores, aunque en Safari se requiere de un *plug-in* extra. Si no se necesita tanta versatilidad como la que brinda MKV, este formato de vídeo puede ser una buena opción.
4. **MPEG-TS**: Es uno de los formatos que más se utilizan para streaming de vídeo, tanto para vídeo en directo como bajo demanda. Es utilizado con HLS (Http Live Streaming), uno de los protocolos de transmisión de streaming de vídeo que más aceptación ha tenido, y que se desarrollará más adelante. En resumen, el vídeo y audio se divide en segmentos que el protocolo HLS distribuye hacia los dispositivos.
5. **AVI**: Es el contenedor propietario de Microsoft, dentro de este formato de vídeo se comprime el vídeo en AC3 y el audio en MP3. No se utiliza para streaming de vídeo, pero sí para distribuir archivos de vídeo.
6. **MOV**: MOV es utilizado por la aplicación Quicktime y fue desarrollado por Apple. Puede ser visto tanto en Windows, como en los dispositivos de Apple. Dentro del contenedor puede ir video comprimido en H.264 y audio en AAC. Al igual que AVI tampoco se puede utilizar para streaming de vídeo en internet, pero sí para compartir archivos.

2.4.2 – Códecs de compresión de vídeo:

Los códecs generalmente especifican cómo se debe codificar y decodificar algún tipo de información. En este caso, son algoritmos utilizados para comprimir y descomprimir archivos de vídeo. Los servicios de streaming utilizan códecs avanzados, como H.264, H.265 y VP8 o VP9 para reducir considerablemente el tamaño de los archivos de vídeo sin sacrificar demasiada calidad visual. Dentro de los códecs generalmente utilizados por los servicios de streaming de contenido multimedia encontramos los siguientes [5]:

1. **H.264/AVC**: También conocido como *Advanced Video Coding (AVC)*, H.264 es uno de los códecs más utilizados en la industria de streaming y en la distribución de contenido en línea. Fue desarrollado por el *ITU-T Video Coding Experts Group (VCEG)* y el *ISO/IEC Moving Picture Experts Group (MPEG)*. H.264 ofrece una buena relación de compresión y calidad de vídeo, lo que lo convierte en una opción popular para transmisiones de alta definición y contenidos en línea.
2. **H.265/HEVC**: El *High Efficiency Video Coding (HEVC)* o H.265 es una evolución del códec H.264 que ofrece una mayor eficiencia de compresión en comparación con su predecesor, lo que significa que puede lograr una calidad de vídeo similar a H.264 con un menor tamaño de archivo. Esta eficiencia lo hace especialmente útil para transmisiones de alta calidad, como vídeos en 4K o contenido en HDR (High Dynamic Range).
3. **VP9**: VP9 es un códec de vídeo de código abierto desarrollado por Google. Fue diseñado para competir con H.265 y ofrece una calidad de vídeo comparable con una menor carga computacional en el decodificador. Google utiliza VP9 en YouTube para transmitir contenido de alta definición y 4K a sus usuarios. VP9 es compatible con una amplia gama de dispositivos y navegadores web, lo que lo convierte en una opción popular para la distribución de vídeo en línea.
4. **AV1**: AV1 es otro códec de vídeo de código abierto desarrollado por la *Alliance for Open Media (AOMedia)*, que incluye empresas como Google, Apple, Amazon, Microsoft y otras. AV1 se diseñó específicamente para superar a H.265 y VP9 en términos de eficiencia de compresión. Es necesario resaltarlo puesto que promete una mejor calidad de vídeo y una mayor tasa de compresión, lo que podría reducir aún más el uso de ancho de banda y mejorar la experiencia de visualización para los usuarios. Aunque AV1 se está implementando cada vez más, su adopción en la industria todavía se encuentra en crecimiento.[6]

2.4.3 – Protocolos de transmisión adaptativa:

Para ofrecer una experiencia de reproducción fluida y sin interrupciones, los servicios de streaming utilizan protocolos de transmisión adaptativa. Estos protocolos permiten que el contenido se adapte en tiempo real a las condiciones de la red y a las capacidades del dispositivo del usuario, cambiando la calidad de la transmisión según sea necesario para mantener una reproducción continua. La arquitectura y diferenciación de los protocolos que

se pretenden describir a continuación resulta de una amplitud y complejidad notable, por lo que no es razonable desarrollarla en su totalidad. Sin embargo, sí es necesario resaltar su funcionamiento. Este se basa en la segmentación del contenido para que la fase de distribución en la que el servicio streaming envía los ficheros al reproductor del cliente, ocurra a demanda de este. Para ello, es fundamental que toda la información relevante respecto a las distintas calidades (resoluciones) en la que se almacena el contenido, lenguaje de los subtítulos si los hay, tamaño de los segmentos, ancho de banda, la codificación llevada a cabo, etc; debe quedar registrada en un archivo de texto almacenado por la parte del servicio: [7]

1. **HTTP Live Streaming (HLS)**: HLS es un protocolo desarrollado por Apple y ampliamente adoptado para la transmisión de vídeo en línea. Su funcionamiento es el siguiente:
    a. Un vídeo se divide en segmentos (archivos TS o fMP4), y la duración de cada segmento suele ser de unos 6 o 10 segundos.
    b. La ubicación y secuencia de entrega de estos segmentos se describen en un conjunto de archivos XML denominados *playlists* (listas de reproducción) o archivos m3u8.
    c. Las *playlists* y los segmentos de vídeo se almacenan en un servidor de streaming.
    d. El vídeo puede reproducirse utilizando un reproductor de vídeo compatible con HLS (reproductores HTML5, AVPlayer en el ecosistema Apple, etc.) para reproducir los vídeos. Originalmente, HLS solo admitía archivos TS, pero Apple introdujo la compatibilidad con archivos fmp4 (mp4 fragmentados) más adelante en la WWDC de 2016. [8]

    En la figura siguiente se observa la arquitectura de HLS en un esquema proporcionado por Apple:

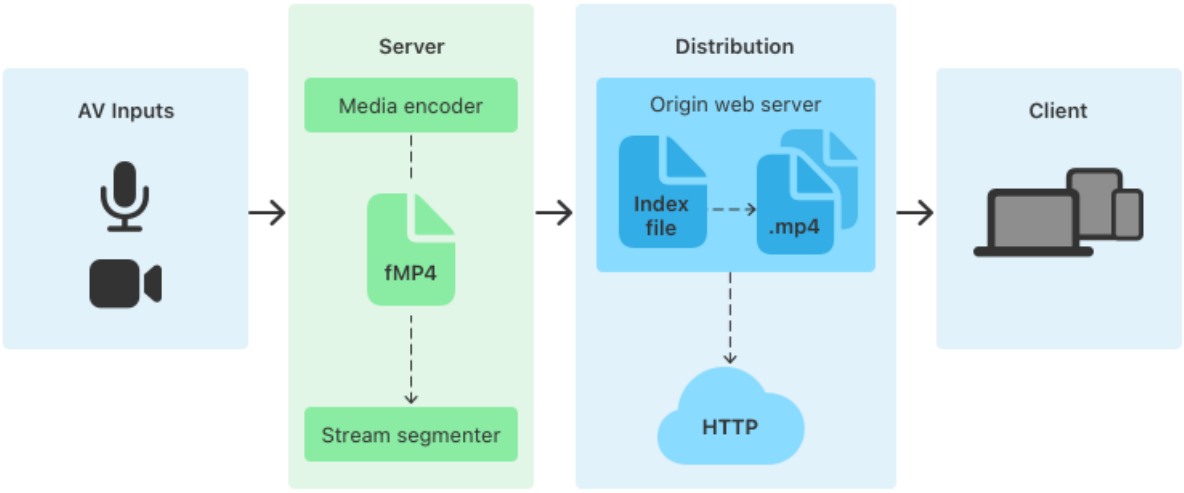


Figura 5. Arquitectura HLS. Fuente: [9]

2. **Dynamic Adaptive Streaming over HTTP (DASH)**: DASH es un estándar de transmisión adaptativa desarrollado por el Moving Picture Experts Group (MPEG). Al igual que HLS, DASH segmenta el contenido multimedia en pequeños fragmentos y permite que el reproductor seleccione y descargue segmentos según la velocidad de conexión y la capacidad del dispositivo. Suponiendo que cada vídeo se codifica en diferentes variantes de representación o perfiles, que son combinaciones de distintos bitrate y resoluciones, el funcionamiento de DASH sería el siguiente [7]:
    a. Los perfiles o variantes de representación se envían a un servicio de empaquetado MPEG-DASH que divide cada variante de representación en pequeñas piezas o trozos de una duración específica (por ejemplo, 2/4/6 segundos de duración).
    b. Un *packager* (empaquetador) también registra cómo se han dividido los vídeos, el orden en que deben entregarse y otros metadatos importantes en un archivo de texto llamado MPD o manifiesto.
    c. Los vídeos empaquetados y las MPD se almacenan en un servidor y se sirven a través de una red de distribución de contenido (*CDN – Content Delivery Network).*
    d. Los usuarios pueden reproducir el vídeo con un reproductor de vídeo compatible con MPEG-DASH.

Se puede apreciar en las figuras siguientes no solo la arquitectura de este protocolo, sino también un esquema del comportamiento de un reproductor multimedia cuando se usa DASH:

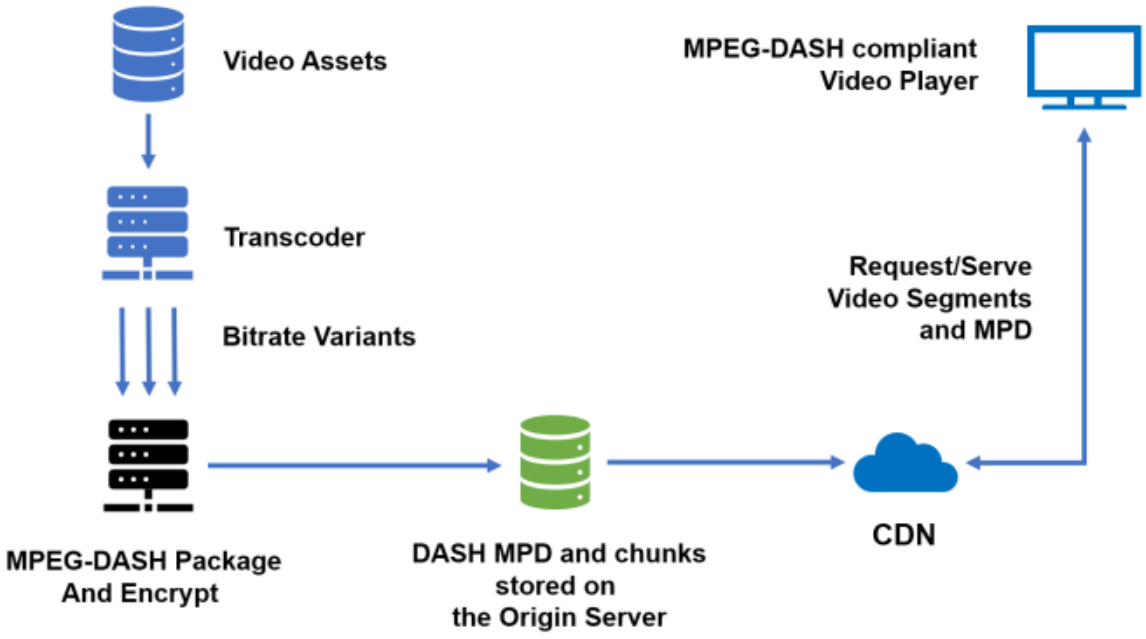


Figura 6. Arquitectura MPEG-DASH. Fuente: [7]

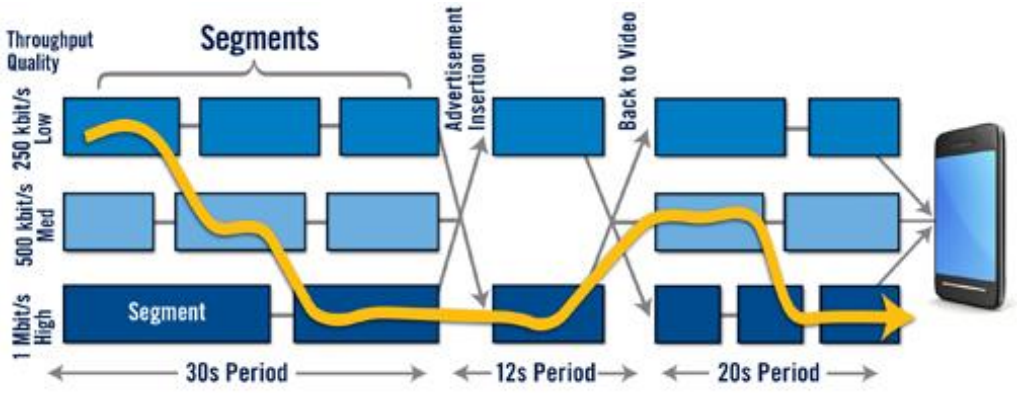


Figura 7. Esquema de un reproductor multimedia utilizando MPEG-DASH. Fuente: https://tdngan.wordpress.com/2016/11/17/how-to-encode-multi-bitrate-videos-in-mpeg-dash-for-mse-based-media-players/

Este esquema del comportamiento del reproductor pretende explicar la capacidad del reproductor de cambiar entre las distintas calidades disponibles con DASH, gracias a la segmentación del vídeo en periodos de tiempo.

2.4.4 – Protocolo de transporte QUIC

Los protocolos de transmisión adaptativa mencionados anteriormente, son esenciales para ofrecer una experiencia de streaming óptima, pero también dependen del protocolo de transporte de Internet que se utilice para transmitir los datos. Dos de los protocolos de transporte más destacados en la actualidad, utilizados en servicios de streaming de contenido son TCP (*Transmission Control Protocol*) y QUIC (*Quick UDP Internet Connections*).

TCP ha sido uno de los protocolos de transporte más utilizados en Internet durante décadas, y de hecho sigue siéndolo. Proporciona una comunicación segura y ordenada entre dispositivos, garantizando que los datos se entreguen correctamente y en el orden correcto. Sin embargo, TCP tiene algunas características que pueden afectar el rendimiento de la transmisión de contenido multimedia en tiempo real. El mecanismo de control de congestión de TCP puede introducir cierta latencia en la transmisión de datos, lo que puede afectar la reproducción fluida del contenido multimedia. Además, TCP tiene un comportamiento reactivo ante pérdidas de paquetes, lo que puede resultar en una reducción significativa en la velocidad de transmisión en caso de congestión en la red.

Es por esto por lo que resulta interesante QUIC. Tal y como se ha visto en secciones anteriores, ha tenido un crecimiento significativo en los últimos años. En este caso QUIC es un protocolo de transporte diseñado por Google para mejorar la entrega de contenido en tiempo real en situaciones donde TCP pueda tener limitaciones. Está basado en el protocolo de transporte UDP (*User Datagram Protocol*) en lugar de TCP. Esto le permite ofrecer un rendimiento más rápido y eficiente en comparación con TCP. Verdaderamente, el telón de fondo de los trabajos en QUIC fue el deseo de desarrollar una alternativa a las soluciones de seguridad establecidas a partir de TCP, HTTP/2 y TLS/SSL que ofrecen la misma protección, pero también una demora reducida en la conexión y en el transporte, y otras ventajas como la posibilidad de realizar conexiones multiplexadas. [10]

Desde 2016 existe un grupo de trabajo oficial del IETF (*Internet Engineering Task Force*) que se encarga de la optimización del protocolo QUIC. Alrededor de 50 desarrolladores de Google, Mozilla, Microsoft y otras empresas, están implicados en el desarrollo y la expansión de la especificación. En los servidores de Google el protocolo ya se utiliza desde hace una década (2013). Asimismo, QUIC también se ha implementado en Chrome, por lo que una parte del tráfico de Internet (p. ej., YouTube) se gestiona actualmente a través de este protocolo de transporte. [10]

La razón principal del rendimiento superior de QUIC frente a TCP es su rapidez en establecer conexión. QUIC inicia una conexión con un único paquete (o dos paquetes si se trata de la primera vez que se establece la conexión) y transmite en ellos todos los parámetros TLS o HTTPS necesarios. En la mayoría de los casos, un cliente puede enviar datos directamente a un servidor sin que este tenga que enviar una respuesta, mientras que TCP debe obtener y procesar la confirmación del servidor. [10]

Mientras que la autenticación y el cifrado se ocupan, sin duda, de un transporte de datos seguro, estos dos factores también son responsables de una desventaja decisiva de QUIC: debido a que los encabezados del paquete contienen menos información con texto claro que en las conexiones TCP, tareas como la solución de problemas, la regulación del tráfico o la gestión de redes en conexiones QUIC se complican notablemente. Otro problema del protocolo QUIC es que el control automático de las sobrecargas en las conexiones con un amplio ancho de banda puede provocar en algunos casos una peor tasa de transferencia. [10]

Sin embargo, también se encuentran otras ventajas, que en el caso de transmisión de contenido en directo resultan interesantes. QUIC implementa su propio mecanismo de control de congestión y recuperación rápida ante pérdida de paquetes, lo que puede reducir significativamente la latencia y mejorar la reproducción de contenido multimedia en tiempo real. Además, QUIC también permite a los servidores enviar múltiples flujos de datos a través de una única conexión, lo que disminuye la latencia de establecimiento de nuevas conexiones y mejora aún más la velocidad de transmisión. Es por eso que la implementación de QUIC en YouTube ha demostrado una mejora significativa en el rendimiento, ofreciendo una reproducción más rápida y fluida para los usuarios, especialmente en redes con alta latencia y pérdida de paquetes.

2.4.5 – Content Delivery Networks (CDN)

Las Redes de Distribución de Contenidos (CDN) son infraestructuras diseñadas para mejorar la eficiencia y la calidad en la entrega de contenido en línea, incluyendo vídeos, imágenes, archivos y otros tipos de datos.

Estas redes distribuyen el contenido a través de servidores ubicados en diferentes puntos geográficos, lo que permite que el contenido se entregue desde un servidor cercano al usuario final, reduciendo la latencia y mejorando el rendimiento. En el contexto de los servicios de streaming multimedia, las CDN juegan un papel esencial para garantizar una experiencia de

visualización óptima para los usuarios. A continuación, se explican las principales características y ventajas de las CDN en el contexto de los servicios de streaming:

- **Latencia reducida:** Al tener servidores distribuidos en diferentes ubicaciones geográficas, las CDN permiten que el contenido se entregue desde un servidor cercano al usuario. Esto reduce la latencia, es decir, el tiempo de retraso entre la solicitud de un vídeo y su reproducción, lo que resulta en una experiencia de streaming más fluida y sin interrupciones.
- **Mayor capacidad de carga:** Los servicios de streaming populares, como Netflix o YouTube, experimentan una gran cantidad de solicitudes de reproducción en todo el mundo. Las CDN tienen una infraestructura escalable y robusta que puede manejar grandes volúmenes de tráfico simultáneo, asegurando que el contenido se entregue de manera rápida y confiable, incluso en momentos de alta demanda.
- **Almacenamiento en caché:** Las CDN utilizan técnicas de almacenamiento en caché para mantener copias del contenido popular en sus servidores distribuidos. Cuando un usuario solicita un vídeo, la CDN puede entregar el contenido desde su caché local, evitando así la necesidad de transmitir el contenido desde el servidor principal, lo que reduce la carga y mejora la eficiencia de la entrega.
- **Adaptabilidad a la calidad de red:** Como parte de la transmisión adaptativa mencionada anteriormente, las CDN trabajan en conjunto con los protocolos de transmisión adaptativa (HLS, DASH, entre otros) para ajustar la calidad del vídeo según las condiciones de la red y la capacidad del dispositivo del usuario. Esto garantiza que los usuarios puedan disfrutar del contenido en la mejor calidad posible, incluso en conexiones de Internet más lentas.
- **Respaldo contra fallos:** La distribución del contenido a través de múltiples servidores y ubicaciones geográficas también ofrece una mayor resistencia ante posibles fallos en la red o en los servidores. Si un servidor falla, la CDN puede redirigir automáticamente el tráfico a otros servidores disponibles, asegurando una disponibilidad continua del contenido.

2.5 - Ahorro energético en redes inalámbricas.

El ahorro de energía en las redes móviles inalámbricas ha sido crucial en los últimos años debido a la limitación de la capacidad de las baterías entre otras cosas. Sin embargo, tal y como se ha podido observar en secciones anteriores, el consumo de contenido multimedia, y consecuentemente, el consumo de energía de los dispositivos móviles crece a medida que los servicios prestados por la red de comunicaciones se hacen más y más complicados.

Aunque las baterías siguen evolucionando, es difícil satisfacer el creciente consumo de energía de los dispositivos móviles. Por tanto, desarrollar técnicas para reducir el consumo de energía es otra forma de alargar la vida de la batería de los dispositivos móviles. A continuación, la figura 8 resume las principales estrategias o mecanismos para reducir el consumo de energía de los equipos de usuario (a partir de ahora mencionados como UE) y prolongar la vida útil de la batería.

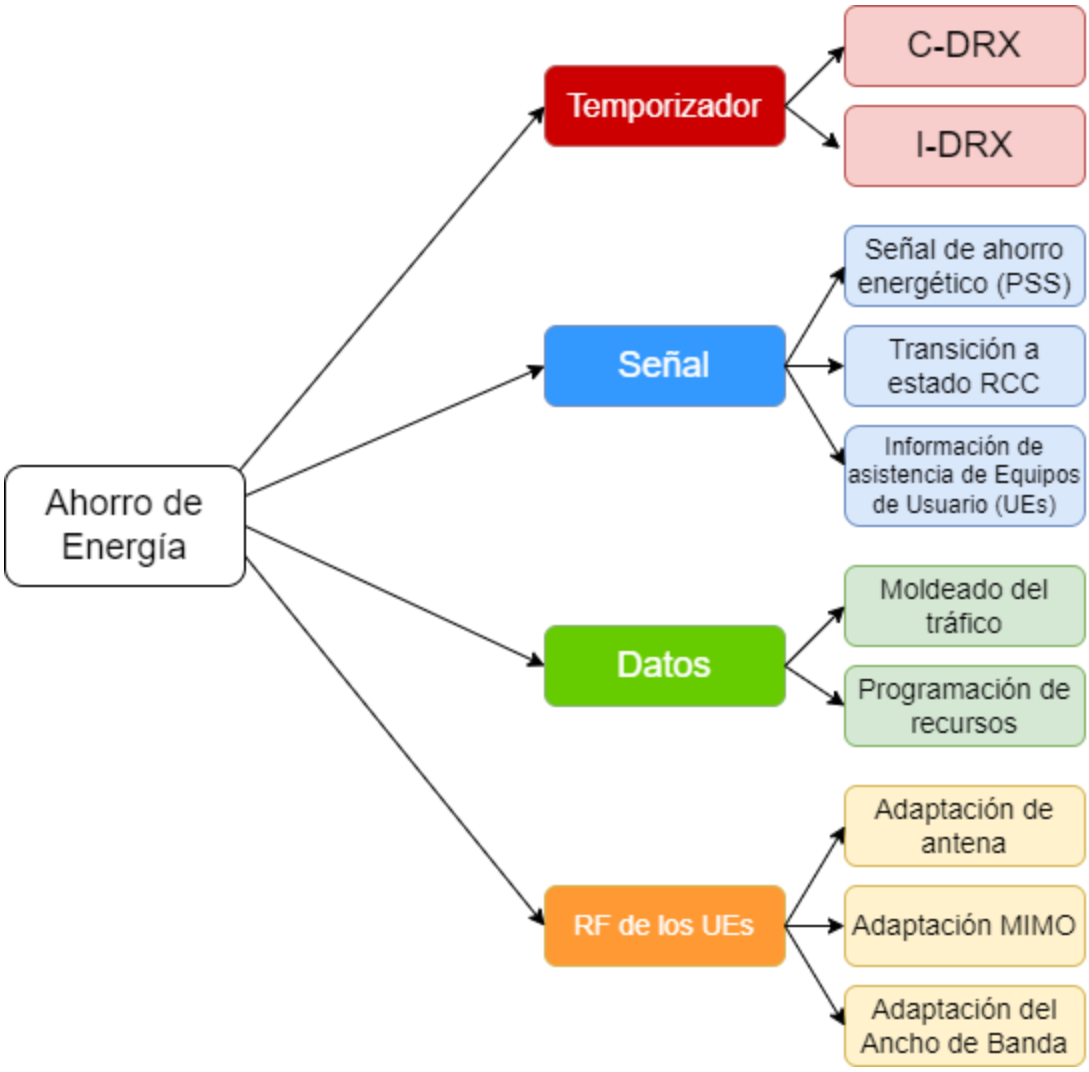


Figura 8. Principales mecanismos de ahorro de energía para equipos en redes móviles inalámbricas.

Los mecanismos de ahorro de energía se pueden clasificar en cuatro grupos: basados en temporizadores, basados en señales, de dominio de datos y de dominio de radiofrecuencia (RF) del equipo de usuario. Los dos primeros grupos son los mecanismos que estudian los procedimientos relacionados con la recepción discontinua (*Discontinuous Reception - DRX*) para el equipo de usuario y la estación base con o sin intercambio de mensajes de control.

En cuanto a los mecanismos de dominio de datos, la estación base ayuda al UE a ahorrar energía programando y dando forma al flujo de tráfico de datos. Por último, las soluciones del dominio Radio Frecuencia del Equipo de Usuario, mejoran las configuraciones y los ajustes de los módulos relacionados con la transmisión, como el módulo de antena, el módulo de entrada múltiple-salida múltiple (*Multiple-Input Multiple-Output - MIMO*) y ajustes de la parte de ancho de banda (*Bandwidth Part - BWP*). [11]

Muchos investigadores se han dedicado a desarrollar estos mecanismos de ahorro de energía. Entre estas técnicas, DRX es el mecanismo más famoso, ya que permite a los dispositivos móviles ahorrar energía sin mucha sobrecarga de señalización entre los UE y las estaciones base. Por ello, es también el principal mecanismo de ahorro de energía en la capa de control de acceso al medio (*Media Access Control - MAC*) de los sistemas del *3rd Generation Partnership Project (3GPP)*, además del principal mecanismo a estudiar en el desarrollo de este proyecto. Tal y como se explicará más adelante, DRX, propuesto en la red LTE (4G) puede llegar a disminuir significativamente el consumo de energía de los UE.

2.5.1 – Funcionamiento del DRX básico en modo "Conectado" en el estándar 3GPP

En el estándar 3GPP [12], el mecanismo DRX puede aplicarse a los equipos de usuario en los estados "Conectado", "Inactivo" y "Reposo". Sin embargo, esta explicación únicamente servirá de introducción al funcionamiento de la Recepción Discontinua en estado "Conectado" (C-DRX).

El mecanismo DRX [11] [12] ya se había adoptado en las redes LTE tradicionales para el ahorro de energía de los equipos de usuario (UEs). Permite al equipo apagar la mayoría de los circuitos de radiofrecuencia cuando no hay tráfico de datos. El equipo entra en reposo durante el periodo de tiempo en el que no recibe paquetes. Sin embargo, el equipo permanece conectado para poder recuperarse de la inactividad más rápidamente. Si el equipo permanece en reposo durante más tiempo, dejará de estar conectado y pasará a estar inactivo para así liberar más recursos de red para otros equipos.

La monitorización del *Physical Downlink Control Channel (PDCCH)* y la recepción de datos son dos procesos que suceden cuando se produce tráfico de bajada en los que UE consumen mucha energía. Cuando un equipo está activo o conectado, debe supervisar este canal de control, el PDCCH, y comprobar si se ha enviado un nuevo paquete *downlink*. El proceso de monitorización del PDCCH implica un procedimiento de decodificación, que consume mucha energía. Si se envía un nuevo paquete, el UE decodifica los recursos inalámbricos correspondientes indicados en el PDCCH; de lo contrario, el equipo de usuario sigue monitorizando el PDCCH. Así pues, el concepto central del mecanismo DRX es omitir la monitorización del PDCCH mientras no existan indicaciones de llegada de paquetes *downlink*.

La recepción discontinua en modo conectado (C-DRX) se caracteriza principalmente por tres parámetros: *DRX cycle (short / long), on duration timer* e *inactivity timer*. [11]

- *DRX cycle*: Ciclo temporal fijo que cada vez que sucede requiere que el UE se despierte o se active. El *long cycle* suele ser el utilizado por defecto, pero opcionalmente se aplica el *short cycle* justo en el momento en el que un UE entra en estado de letargo o inactividad, para volver al *long cycle* si tras aplicarse varias veces no se recibe tráfico de paquetes.
- *On duration timer*: Temporizador que establece el mínimo tiempo de escucha de un UE que sucede cada vez que este despierta.

- *Inactivity timer*: Temporizador que establece el tiempo mínimo de espera que sucede justo después de que se reciba con éxito un paquete. En caso de que este temporizador expire el UE puede pasar a estado de letargo.

Cuando un equipo de usuario se configura con DRX sucede lo siguiente: El UE se conecta periódicamente e inicia su *on duration timer* de acuerdo con el *DRX cycle*. A lo largo del tiempo de duración de este temporizador, el equipo debe supervisar el PDCCH para comprobar si existe tráfico de bajada. Una vez expirado el *on duration timer*, el UE tiene la oportunidad de utilizar DRX mientras no finalice el *DRX cycle*, pudiendo omitir la decodificación del PDCCH y apagar su módulo RF para ahorrar energía.

Por otro lado, en el caso en el que el UE sí que reciba un paquete *downlink* antes de que expire el *on duration timer*, el UE inicia el *inactivity timer* y prolonga el tiempo conectado para recibir más paquetes consecutivos. Tal y como se muestra en la figura siguiente, cada recepción con éxito reinicia el *inactivity timer*, por lo que el UE no tiene la oportunidad de utilizar DRX hasta que expire el *inactivity timer*.

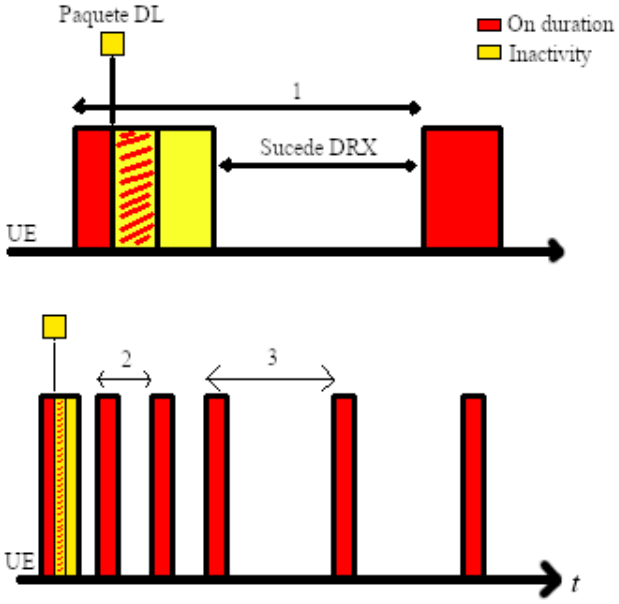


Figura 9. Funcionamiento del on duration timer y del inactivity timer en el mecanismo DRX. También se muestra en la parte inferior un ejemplo de cómo se aplica el long cycle tras varias iteraciones del short cycle, siendo: 1) DRX cycle, 2) Short cycle, 3) Long cycle.

Diseñando y ajustando un periodo adecuado para el *DRX cycle*, el mecanismo DRX posibilita un gran ahorro energético en redes cuyo tráfico es periódico. Si además se configuran adecuadamente el *inactivity timer* y el *short cycle*, el mecanismo DRX también brinda un óptimo ahorro energético en redes cuyo tráfico sucede en ráfagas. Como contraparte, los requisitos de los distintos servicios de las nuevas redes 5G son cada vez mayores y más diversos, por lo que el estudio y la evolución de DRX siguen siendo inevitables [11].

Es interesante conocer el funcionamiento y los parámetros descritos en el estándar del 3GPP, explicados en [11] y [12], puesto que en el desarrollo del proyecto se configurará un simulador de este mecanismo de ahorro energético en redes que nos permitirá obtener resultados a partir de la ejecución del simulador con datos de tráfico de paquetes reales. La explicación del simulador se llevará a cabo más adelante en la sección "*4.2 – Caracterización de modelos de consumo energético de los servicios de streaming multimedia"*.

Otro de los parámetros fundamentales de este mecanismo, es la Señal de Ahorro Energético (*PSS – Power Saving Signal*). Esta es una señal clave, ya que indica a los dispositivos cuándo entrar en un modo de bajo consumo de energía, lo que ahorra batería al reducir la frecuencia de escucha de datos. Cuando se recibe el PSS, el dispositivo sabe que puede dormir para ahorrar energía y despertarse solo cuando se esperan datos, logrando eficiencia energética.

Finalmente, tal y como se explica en [11] el mecanismo DRX es propuesto para el ahorro de energía del UE durante la recepción del tráfico (*downlink*). Sin embargo, es importante destacar que también influye en el comportamiento de las transmisiones *uplink.* Mientras que en el UE exista tráfico de bajada, siempre podrá enviar los paquetes *uplink* que correspondan durante el periodo de tiempo que esté activado el *inactivity timer*, pero es necesario tener en cuenta que cada transmisión *uplink* reiniciará el *inactivity timer* y por tanto impedirá que el equipo de usuario permanezca en reposo, de acuerdo con el estándar 3GPP, aumentando consecuentemente el consumo de energía del equipo.

Para evitar precisamente esto, algo que algunos fabricantes de UEs consideran un derroche de energía, es posible que prefieran aplazar las transmisiones *uplink* hasta el siguiente ciclo DRX. Otra alternativa que podría ahorrar más energía se daría alineando las transmisiones de enlace ascendente con el ciclo DRX. En el ámbito de este proyecto se analizará en su correspondiente sección, el impacto de evitar o no evitar el tráfico de paquetes de subida en el consumo energético de los UEs con un simulador de eficiencia energética que implementa DRX.

2.6 – Puntos clave

Es posible que la relevancia de lo escrito en este capítulo que sirve a modo de contexto, no se entienda sin la presencia de esta sección que lo cierra.

En definitiva, se ha descubierto un conjunto de fenómenos sociales, como la llegada del virus COVID-19, cuya trascendencia ha supuesto un punto de inflexión en nuestros hábitos de interacción y vida en sociedad, con un consecuente desarrollo de las tecnologías de las redes de comunicación, más concretamente, aquellas que soportan la reproducción, gestión y distribución de contenido multimedia.

Este incipiente desarrollo viene caracterizado en parte por diversos aspectos tecnológicos presentes en plataformas web que sirven el contenido multimedia, junto con la implementación de mecanismos de ahorro energético en las redes que lo albergan, para así asegurar una gestión óptima y eficiente de la distribución y reproducción del contenido.

Capítulo 3. Metodología.

Este capítulo juega un papel fundamental en este trabajo, ya que proporciona una descripción detallada de los enfoques y herramientas utilizados para llevar a cabo el proyecto y alcanzar los objetivos planteados.

En primer lugar, se abordarán los principios del *mindset Agile* ya que se ha buscado trabajar con la intención de garantizar eficiencia y adaptabilidad durante todo el proyecto. A continuación, se presentará una visión general de las herramientas de software utilizadas. Para la captura del tráfico, se ha utilizado WireShark, para el desarrollo de scripts, Microsoft Visual Studio como IDE (*Integrated Development Environment*), junto con Python como lenguaje de programación. Cabe destacar también la implementación de la tecnología de Git en el proyecto. La descripción de estas herramientas además de la justificación de su uso se verá en sus correspondientes secciones.

3.1 - Agile Mindset

El framework *Agile*, también conocido como enfoque ágil, es una filosofía de trabajo y conjunto de principios utilizados en el desarrollo de proyectos para mejorar la flexibilidad, eficiencia y adaptabilidad. Se basa en la idea de que los requisitos y objetivos de un proyecto pueden cambiar a lo largo del tiempo, y que es fundamental poder responder rápidamente a esos cambios.

En lugar de seguir un enfoque tradicional y lineal, en el que se establecen requisitos detallados y se planifica todo el proyecto de antemano, el enfoque ágil se centra en iteraciones cortas y colaborativas, donde se construyen, prueban y mejoran incrementos del producto en cada ciclo, siendo un ciclo una unidad de tiempo determinada por el equipo. El objetivo es proporcionar valor al cliente de forma temprana y continua, a la vez que se fomenta la adaptabilidad y la retroalimentación constante. En este caso, el proyecto no se desarrolla con ningún cliente ni con ningún equipo involucrado, ya que no hay producto que vender. Únicamente existe la relación de tutor-alumno, en la que el tutor es consciente de la amplitud del proyecto, sus características y los requisitos imprescindibles.

Agile se rige por una serie de valores fundamentales, descritos todos en el "*Agile Manifesto*" [13]:

- **Individuos e interacciones sobre procesos y herramientas**: Se prioriza la comunicación y colaboración efectiva entre las personas involucradas en el proyecto, reconociendo que son la clave para el éxito.
- **Software funcional sobre documentación exhaustiva**: Se valora más la entrega de un producto que funcione correctamente que la generación excesiva de documentación detallada.
- **Colaboración con el cliente sobre negociación contractual**: Se promueve la colaboración activa y continua con el cliente para comprender y satisfacer sus necesidades en constante evolución.

- **Responder a los cambios sobre seguir un plan**: Se acepta que los requisitos pueden cambiar y se adopta una mentalidad de adaptación y respuesta rápida a medida que se obtiene más información a lo largo del proyecto.

En *Agile*, se pueden utilizar diversas metodologías, como *Scrum*, *Kanban* o *Extreme Programming (XP)*, que ofrecen estructuras y prácticas específicas para implementar los principios ágiles. Estas metodologías se caracterizan por la autogestión del equipo, las iteraciones cortas, las reuniones diarias de seguimiento, la priorización del trabajo y la retroalimentación constante.

Dicho lo cual, es imprescindible tener en cuenta que en este trabajo no se ha implementado ninguna de las metodologías anteriormente descritas que trabajan con Agile. *Scrum* por ejemplo implica que dentro de la autogestión que se lleve a cabo, es necesario trabajar con tareas que surgen de la visión del proyecto del equipo (*user stories*), que se añaden a una lista en la que se controla el progreso y el avance (*sprint/product backlog*), existiendo también reuniones diarias de seguimiento (*daily meetings*), y ese no ha sido el caso. Simplemente se ha utilizado la filosofía de trabajo descrita en los principios de *Agile* para llevar a cabo de la mejor forma posible la autogestión y el progreso dentro del proyecto. A continuación, se describirá como se ha aprovechado esta filosofía de trabajo.

3.1.1 – Aplicación de Agile en el proyecto

Durante el desarrollo del proyecto, se considera que se ha aplicado de manera efectiva este *mindset Agile* por diversas razones. En primer lugar, la relación alumno-tutor ha sido fundamental en este enfoque, ya que el tutor ha tenido una visión clara de la amplitud y los requisitos del proyecto, guiando al alumno en cada etapa.

Se han adoptado ciclos de trabajo cortos, estableciendo reuniones periódicas de seguimiento que abarcaban períodos de una o dos semanas. Estas reuniones han sido oportunidades para revisar y analizar el trabajo realizado hasta el momento, así como para abordar cualquier problema o desafío que surgiera. La comunicación ha sido constante y fluida, manteniendo el contacto tanto mediante correo electrónico como mediante Microsoft Teams, lo que ha facilitado la colaboración y la toma de decisiones informadas.

Desde el principio, se ha buscado entregar software funcional en cada iteración. En la parte de análisis de paquetes con WireShark, por ejemplo, se ha logrado obtener resultados tangibles y realizar mejoras incrementales con cada ciclo. Si bien han surgido problemas y errores a lo largo del camino, se ha abordado cada desafío de manera ágil y resolutiva, aprendiendo de ellos y realizando las correcciones necesarias en cada iteración.

En resumen, la aplicación del *mindset Agile* ha sido clave en este proyecto. Los ciclos de trabajo cortos, las reuniones periódicas, la comunicación constante y la entrega de software funcional en cada iteración han permitido una gestión eficiente y adaptable. Esta metodología ha brindado la flexibilidad necesaria para enfrentar los cambios del proyecto, logrando así resultados satisfactorios y una mejora continua en cada etapa del desarrollo.

3.2 – Software utilizado

3.2.1 – WireShark

Se ha utilizado la herramienta WireShark para la captura de tráfico de paquetes de diferentes servicios de streaming de vídeo. WireShark es una herramienta de análisis de red ampliamente reconocida y utilizada en el campo de las comunicaciones de datos.

La elección de WireShark se ha basado en mi experiencia previa con esta herramienta, habiendo trabajado con ella en asignaturas como "*Plataformas de Streaming*". Esta experiencia previa me ha permitido aprovechar eficientemente las capacidades de WireShark y aplicarlas al análisis del tráfico de paquetes de los servicios de streaming seleccionados.

Con WireShark, se han podido realizar capturas de paquetes de servicios populares de streaming, como *Netflix, Prime Video, Twitch* y *YouTube*. Estas capturas han proporcionado datos fundamentales para el estudio del tráfico multimedia. Además, WireShark ofrece una interfaz intuitiva y funcionalidades avanzadas que han facilitado la inspección detallada de los paquetes capturados, pudiendo examinar los campos de los protocolos utilizados, analizar los flujos de datos y extraer información relevante para el estudio de los servicios de streaming.

En el momento de realizar las distintas capturas de tráfico de cada plataforma, se ha detectado una complejidad inesperada del mismo. Se observa que no todo el contenido de paquetes procede de una misma dirección IP, o de un mismo puerto, ni tampoco llegan al mismo puerto. Por tanto, no resulta razonable aplicar en WireShark filtros de captura que aíslen únicamente el contenido de paquetes intercambiados con el servicio de streaming que se pretenda capturar, y esto ha sido determinante en la metodología usada para hacer las capturas, que resulta ser:

I. Comprobación manual de que los únicos servicios que están generando tráfico en el momento de la captura son los servicios de streaming de vídeo deseados. Se cierran previamente otras ventanas del navegador, además de otras aplicaciones utilizadas para la comunicación, como lo son Microsoft Teams o Discord.
II. Filtrado automático de paquetes en un script de Python. Este filtrado se explicará más adelante en el capítulo "*4.1.1 – Parser. Analizador de archivos .pcap*" , pero implica una selección del tráfico que resultará interesante de cara al análisis conjunto del tráfico multimedia generado por cada servicio streaming.

Las consecuencias de esta metodología en la captura de paquetes también se verán desarrolladas en el posterior análisis de resultados.

3.2.2 – Python

Se ha escogido Python como lenguaje de programación único para el desarrollo, ya que todas las funcionalidades y scripts necesarios para alcanzar los objetivos del proyecto han sido codificados en Python. Esta elección se ha basado en varios factores fundamentales.

En primer lugar, Python ha sido seleccionado debido a mi familiaridad y experiencia previa con este lenguaje a lo largo de la carrera. Durante mis estudios, he tenido la oportunidad de adquirir un sólido conocimiento de Python, lo que me ha permitido aprovechar eficientemente sus capacidades y características durante el desarrollo del proyecto.

Además, Python es ampliamente reconocido por su facilidad de uso y legibilidad de código. Su sintaxis clara y concisa, combinada con su enfoque en la legibilidad, facilita la comprensión y el mantenimiento del código a lo largo del tiempo. Esta característica resulta especialmente valiosa en un proyecto de larga duración, como es el caso, donde la claridad y el orden del código son esenciales para garantizar la escalabilidad y la facilidad de colaboración.

Otro aspecto destacado de Python es su amplia gama de librerías y paquetes disponibles. La comunidad de Python ha desarrollado una gran cantidad de librerías especializadas que abarcan desde el análisis de datos hasta la creación de interfaces gráficas de usuario. Esto ha permitido aprovechar estas librerías para simplificar el desarrollo y aumentar la eficiencia del proyecto. En particular, se han utilizado librerías populares como *numpy*, *pandas* y *matplotlib* para el análisis y visualización de datos. Su uso concreto será explicado más adelante en el capítulo "*4.1 – Caracterización de modelos de tráfico multimedia de plataformas de streaming* "

3.2.3 – Microsoft Visual Studio Code

Para llevar a cabo la implementación y codificación de los archivos Python que conforman este proyecto, se ha utilizado Microsoft Visual Studio Code como el entorno de desarrollo integrado (IDE) principal. Esta herramienta ha proporcionado un conjunto de características y funcionalidades que han resultado fundamentales para el desarrollo eficiente y efectivo del proyecto.

Microsoft Visual Studio Code ha sido configurado con varias extensiones específicas que han optimizado el flujo de trabajo y mejorado la productividad. Estas extensiones incluyen:

- *Python*: Esta extensión permite el soporte completo del lenguaje Python dentro del IDE. Proporciona herramientas avanzadas de edición, resaltado de sintaxis, sugerencias automáticas y depuración, lo que facilita la codificación y mejora la eficiencia del desarrollo.
- *Pylance*: Pylance es una extensión de Microsoft que ofrece una experiencia de desarrollo Python aún más enriquecida. Proporciona capacidades avanzadas de análisis estático, autocompletado inteligente, navegación rápida entre archivos y una mejor integración con otras herramientas de terceros.

- *Prettier*: Esta extensión es útil para mantener un formato de código consistente y legible. Automatiza el formateo del código Python, lo que garantiza una apariencia uniforme en todo el proyecto y facilita la colaboración con otros desarrolladores.
- *Kite AutoComplete AI Code*: Kite es una extensión impulsada por inteligencia artificial que mejora la experiencia de autocompletado de código. Proporciona sugerencias contextuales y precisas mientras se escribe código Python, lo que acelera el proceso de desarrollo y ayuda a evitar errores comunes.
- *Isort*: Isort es una extensión que se encarga de organizar y ordenar automáticamente las importaciones en los archivos Python. Esta función es especialmente útil cuando se trabaja con múltiples módulos y dependencias, garantizando una estructura limpia y organizada del código.

Además de estas extensiones, se han aplicado distintos temas de apariencia de tipo 'light' dentro de Microsoft Visual Studio Code, como lo son *Atom One* o *Bluloco*. Estos temas han sido seleccionados con el propósito de realizar capturas de código más limpias y legibles, lo que ha facilitado la documentación y presentación visual del proyecto.

3.2.4 – Git

Git es un sistema de control de versiones distribuido ampliamente utilizado en el desarrollo de software. Se utiliza para rastrear y administrar los cambios realizados en archivos y proyectos a lo largo del tiempo, permitiendo a los desarrolladores colaborar de manera efectiva y mantener un historial completo de todas las modificaciones realizadas.

A lo largo de la carrera ya se había utilizado Git dentro de la creación de proyectos en asignaturas como "*Aplicaciones y Usabilidad*", pero verdaderamente ha sido escogido como tecnología por razones con más peso. Entre ellas, caben destacar el enfoque distribuido, que permite tener, en este caso, tanto al tutor como al alumno una copia completa del repositorio; un historial completo de cambios, que quedan grabados gracias a la utilización de "*commits*" en los distintos archivos que son añadidos, eliminados o modificados, pudiendo acceder a cualquier versión anterior de estos archivos; y por último, por su integración con varias plataformas de alojamiento en la nube, como GitHub y GitLab, que facilitan la colaboración e intercambio de código entre los participantes.

Precisamente esta última razón es considerada de vital importancia, puesto que la totalidad del proyecto es llevada a cabo en un único ordenador. Si todo el proyecto queda almacenado en local y por algún casual o circunstancia desfavorable este ordenador se extravía o sufre una avería, la totalidad del proyecto se pierde con él. Este es un escenario que, a pesar de ser totalmente indeseado por cualquier alumno durante el desarrollo de su Trabajo de Fin de Grado, es más común de lo que parece, por lo que la utilización de Git respecto a este proyecto resulta imprescindible.

La interacción con Git se ha llevado a cabo desde la terminal que ofrece Visual Studio Code como IDE. La instalación del propio software de Git incorpora otra terminal específica "*Git Bash*", sin embargo, se ha utilizado la anterior por mayor comodidad.

El proceso ha incluido en primer lugar la creación de un repositorio remoto vacío en la nube de GitHub. Posteriormente, se ha clonado en local, con la idea de añadir ahí los sucesivos cambios que se requieran durante la etapa de desarrollo. Cada vez que un cambio era añadido, se guardaba en un *commit* en cada archivo con un breve texto descriptivo para luego subir el cambio al repositorio remoto y así guardar el progreso de forma segura.

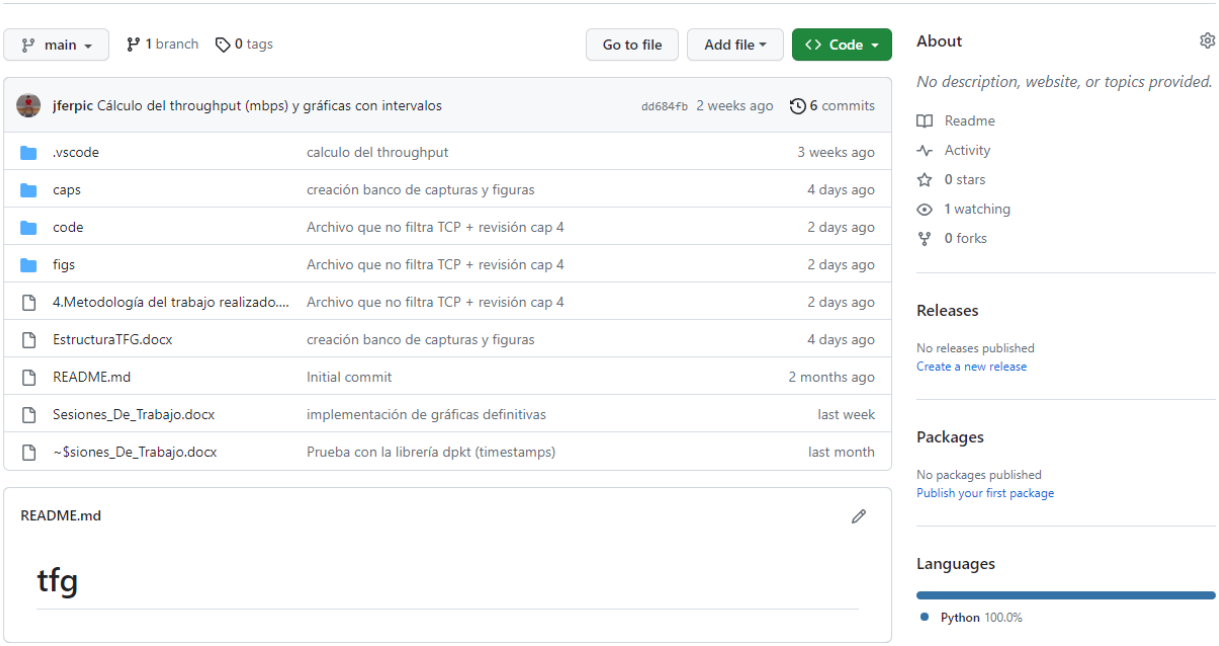


Figura 10. Captura del repositorio remoto del proyecto en GitHub.

Todo el proceso de interacción con el repositorio remoto se llevaba a cabo desde la terminal ya mencionada con los siguientes comandos:

- **"git clone":** Comando utilizado para crear una copia local de un repositorio remoto. Esto permite trabajar en el proyecto de forma local y realizar cambios en los archivos.
- **"git add ."** Utilizado para agregar los cambios realizados en los archivos al área de preparación (*staging area*). Cuando se hacen modificaciones en los archivos del repositorio local, necesitas agregar esos cambios al área de preparación antes de realizar un *commit*. El punto (".") indica que se deben agregar todos los archivos modificados.
- **"git commit -m**": Se utiliza para confirmar los cambios realizados en los archivos y crear una nueva revisión en el historial del repositorio. Una vez añadidos los cambios al área de preparación, se hace *commit* para guardarse permanentemente en el repositorio. El flag "-m" se utiliza para agregar un mensaje descriptivo que explique los cambios realizados en el *commit*.

- "**git push**": El comando git push se utiliza para enviar los commits locales al repositorio remoto.

Cabe destacar también el uso de "gitk". Se trata de una herramienta que proporciona una interfaz visual al ser ejecutado. A través de la interfaz de *gitk*, se puede navegar por los *commits*, ver detalles de cada *commit* (como el autor, fecha y mensaje), y obtener una visión general del historial de cambios en forma gráfica.

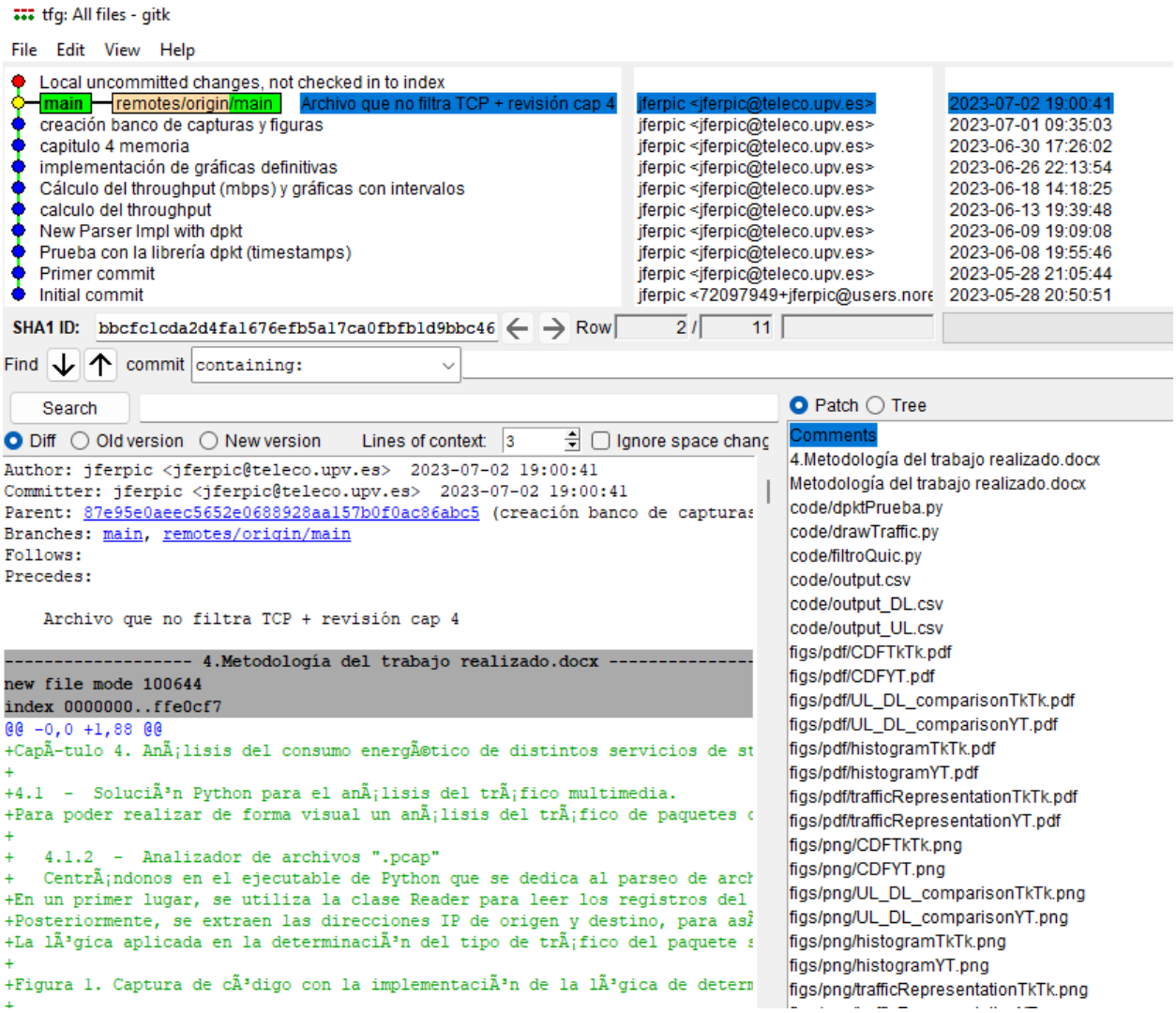


Figura 11. Captura de la interfaz gráfica de gitk.

Capítulo 4. Desarrollo de modelos de tráfico de streaming multimedia y de consumo energético usando Python:

En este capítulo se pretende explicar cómo mediante distintas soluciones programadas en Python se ha logrado realizar un estudio sobre distintos modelos de tráfico generado por algunos de los principales servicios de streaming multimedia, concretamente, los ya mencionados Netflix, Prime Video, YouTube y Twitch. En este caso con el término modelo se pretende designar a la representación de las características del tráfico red correspondiente a la reproducción de contenido en streaming usando estos servicios. Mediante datos reales extraídos de capturas del flujo de paquetes realizadas con Wireshark, ha sido posible analizar cualidades y particularidades del tráfico generado en un dispositivo cuando se consume contenido multimedia, además de obtener resultados del impacto energético en el dispositivo a raíz de utilizar un simulador de eficiencia energética en equipos de usuario.

4.1 – Caracterización de modelos de tráfico multimedia de plataformas de streaming.

Para poder realizar de forma visual un análisis del tráfico de paquetes capturado con WireShark, se ha planteado una solución diseñada en Python que consta de dos archivos; el primero, que correspondería a un analizador de paquetes, permite "parsear" el archivo con extensión ".pcap" que genera el software de Wireshark al guardar en nuestra máquina local una captura de tráfico, para así almacenar en archivos externos (.csv) valores de tamaño y tiempo de cada paquete, en función de si corresponden a tráfico de subida o de bajada; y el segundo, correspondiendo a un visualizador de tráfico, permite plasmar de forma gráfica el resultado de dicha captura, además de un breve estudio estadístico que consta de un histograma de paquetes y su correspondiente función de distribución acumulada. El desarrollo de la caracterización de tráfico multimedia de diferentes servicios streaming se resumiría en la figura a continuación:

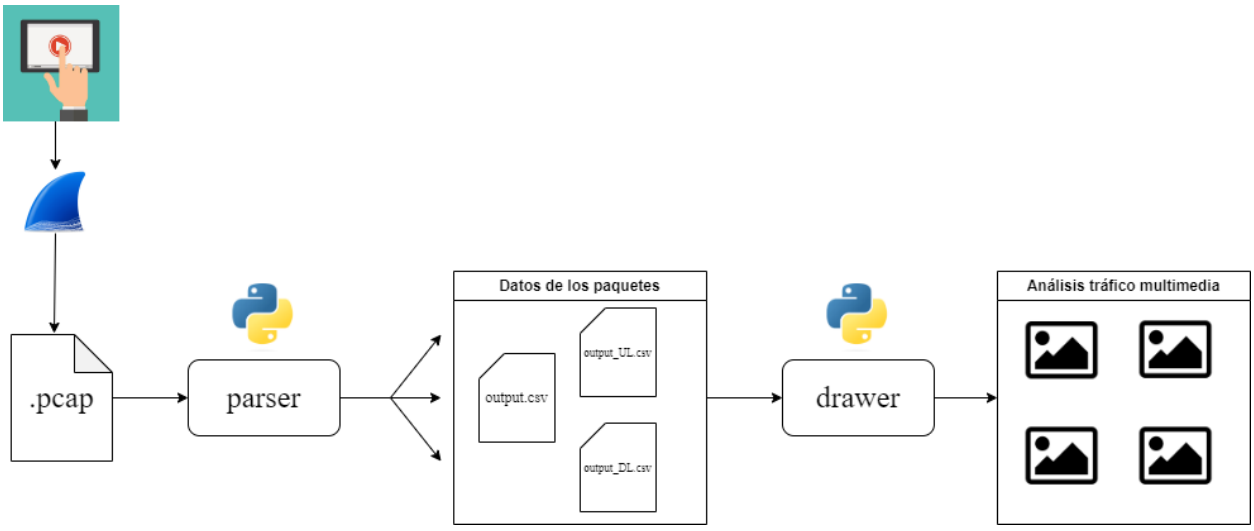


Figura 12. Diagrama representativo del proceso de caracterización del tráfico multimedia de diferentes servicios streaming.

4.1.1 – Parser. Analizador de archivos ".pcap"

Este ejecutable de Python que se dedica al parseo de archivos ".pcap", trabaja sobre una de las múltiples librerías de Python que se han investigado en el trascurso de este trabajo, dedicándose a transformar el paquete completo de datos de red almacenado en los archivos ".pcap" en variables que guarden dichos datos para poder trabajar con ellos de algún modo. Se trata de la librería *dpkt* [14].

En un primer lugar, se leen los registros del archivo de captura de Wireshark que se quiera indicar. Consecuentemente, se decide recorrer cada registro que aparece en el archivo con la finalidad de filtrar y almacenar solo aquellos paquetes que pertenezcan al registro *ethernet*, y a su vez dentro de este, los que únicamente sean IP y TCP. En los archivos de las capturas de paquetes inevitablemente aparecen múltiples tipos de paquete, sin embargo para realizar el análisis sólo nos interesan aquellos que contienen la información propia de la comunicación con el servicio de streaming en cuestión. Como se ha visto en secciones anteriores, la mayoría de servicios de streaming de contenido multimedia utilizan TCP como protocolo de transporte, a excepción de aquellos servicios de Google que utilizan el protocolo QUIC desarrollado por la propia empresa, como es el caso de YouTube. Es por esto que es necesario realizar un filtrado selectivo en función del servicio que se pretenda analizar. En el caso de YouTube, únicamente nos interesarán los paquetes cuyo protocolo de transporte sea QUIC, y en el resto, los paquetes TCP.

Es entonces cuando se extraen las direcciones IP de origen y destino, para así poder determinar si un paquete pertenece a tráfico de subida o a bajada, quedando finalmente registrados en variables el número total de paquetes procesados, el número total de paquetes que son interesantes, es decir, que pasan el filtrado, y el *throughput* medio de la captura, además del tipo de tráfico (*uplink/downlink*), tamaño (en bytes), y el *timestamp* de cada paquete.

La lógica aplicada en la determinación del tipo de tráfico del paquete se debe a la capacidad de la librería de extraer de cada uno las direcciones IP de origen y destino. De este modo, se comparan dichas IP con la dirección IP de nuestra máquina local, de forma que si nuestra IP coincide con la IP de origen del registro de paquete, se le clasificará como tráfico de subida, independientemente de cuál sea el destino, mientras que si coincide con la IP de destino, será clasificado como tráfico de bajada, tal y como se muestra a continuación:

```
if src_ip == '192.168.1.80':
#DICHO PAQUETE CORRESPONDE A TRÁFICO DE SUBIDA
    trafico_list.append('UL')
    trafico_list_UL.append('UL')
    #buf contains the raw packet data -> len(buf) will return the packet size in bytes
    pkt_size_list_UL = np.append(pkt_size_list_UL, len(buf))
    pkt_size_list = np.append(pkt_size_list, len(buf))
    timestamp_list_UL = np.append(timestamp_list_UL, relative_timestamp)

elif dst_ip =='192.168.1.80':
#DICHO PAQUETE CORRESPONDE A TRÁFICO DE BAJADA
    trafico_list.append('DL')
    trafico_list_DL.append('DL')
    pkt_size_list_DL = np.append(pkt_size_list_DL, len(buf))
    pkt_size_list = np.append(pkt_size_list, len(buf))
    timestamp_list_DL = np.append(timestamp_list_DL, relative_timestamp)
```

Figura 13. Captura de código con la implementación de la lógica de determinación del tipo de paquete

Los valores 'UL' (*UpLink* o en castellano "de subida") o 'DL' (*DownLink* o "de bajada"), son añadidos como identificadores a unas listas. De la misma manera, se añade a otra lista el tamaño del paquete, en este caso en bytes. Si el paquete corresponde a tráfico de subida, se añadirá el valor de longitud de ese paquete a la lista que contendrá los distintos tamaños de paquetes de tráfico de subida, además de su *timestamp*, exactamente igual que sucedería en caso de ser un paquete de tráfico de bajada. Esta asignación de tamaños y tiempos a listas distintas se hace con el objetivo de representar más adelante gráficas distintas para tráfico de subida y para tráfico de bajada.

Por otro lado, la palabra reservada *timestamp* en esta librería devuelve el valor de tiempo en el que se capturó cada paquete. Debido a que el formato que presenta dicha variable al ser accedida corresponde al formato universal de fecha, en el que se muestran los segundos transcurridos desde el 1 de enero de 1970, es necesario recalcular dichos timestamps de forma que queden relativos a 0. Esto se consigue guardando el valor del primer timestamp en una variable que se restará cuando se quiera obtener el timestamp de cada paquete, quedando también almacenado en una lista.

Cabe destacar que el motivo principal por el que se escogió la librería *dpkt* antes que otras como *pcapkit* o *scapy*, también muy completas y mucho mejor documentadas, es precisamente por esta palabra reservada que se acaba de comentar. En el caso de dichas librerías, se utilizan métodos que en realidad devuelven el valor del timestamp de ejecución en máquina local del "parseo" de dicho paquete también en formato universal. Aun convirtiendo estos valores de tiempo a valores relativos a 0 se observa que además de no coincidir de ninguna manera con los valores observados desde WireShark, hay una diferencia ínfima entre ellos que confirma la idea de que no se devuelve el tiempo correctamente, sino un valor que cambia con cada ejecución y que por tanto no es válido.

A continuación, el cálculo del *throughput* se hace como medida de rendimiento de la red que se trata de analizar, ya que nos muestra lo que sería una estimación de la velocidad del transporte de datos llevado a cabo. Para ello se implementa la siguiente fórmula en el código, en la que se devuelve además el resultado expresado en *Mbps*. Siendo pkt size $_i$ el tamaño del paquete en cuestión (en bytes); t $_{max}$ el último instante de tiempo de captura de paquetes, y t $_{min}$ el primer instante de tiempo, el throughput se calcularía como:

$$Throughput = \frac{(\sum_{i=1}^{n} \text{pkt size } i) * 8}{(\text{t}\,max - t\ min) * 10\text{^}6}\ [Mbps]$$

Siempre que el script de Python es ejecutado, este valor junto con el computo de paquetes, y de paquetes que pasan el filtrado de la captura, son mostrados por consola al usuario, tal y como podemos observar:

```
Julio@LAPTOP-218G6T4U MINGW64 ~/Desktop/tfg/code (main)
$ C:/Users/Julio/AppData/Local/Programs/Python/Python310/python.exe c:/Users/Julio/Desktop/tfg/code/dpktPrueba.py
../caps/cap2Netflix.pcap contains 11039 packets (10898 interesting)
Throughput medio de la captura: 2.0335 Mbps
```

Figura 14. Captura de consola con el resultado de la ejecución del script que analiza archivos .pcap

Sin embargo, la ejecución de este script en Python no implica únicamente este examen completo del contenido de un archivo de captura de tráfico generado por WireShark, sino también el almacenamiento de nuestras listas de datos de tiempos, tamaños y tipo de tráfico en archivos externos de tipo ".csv" (*comma-separated values*), con la finalidad de ser tratados posteriormente.

Para ello, se hace uso de la librería *csv* [15], que incorpora métodos que permiten crear archivos con esta misma extensión en modo escritura para incorporar tantos datos como se deseen. En este caso se ha decidido guardar las listas de tamaños de paquetes, tiempos y tipo de tráfico, cada una en una columna del csv. Además, puesto que nos interesa hacer representaciones gráficas tanto del tráfico de subida como del de bajada, se crean 3 archivos que almacenen cada uno las listas correspondientes al tipo de tráfico (total, de subida o de bajada) en columnas. En la siguiente figura, podemos observar cómo se ha llevado a cabo esta implementación para crear el archivo csv que almacenará todo el conjunto del tráfico de subida y de bajada capturado. En este caso este tráfico total no se va a representar, a diferencia de los de subida y bajada, pero es interesante almacenarlo también para utilizarlo posteriormente en las simulaciones de consumo energético.

```
# ARCHIVO CON LOS VALORES DEL TRÁFICO TOTAL              # output.csv
with open(output_file, 'w', newline='') as csvfile:
    writer = csv.writer(csvfile)
    writer.writerow(['P_size', 'P_time', 'U / D'])  # Write the column headers
    for i in range(len(trafico_list)):
        writer.writerow([pkt_size_list[i], timestamp_list[i], trafico_list[i]])
```

Figura 15. Captura de código que crea el archivo .csv en los que almacenamos los valores de las listas que nos interesan.

4.1.2 – Drawer. Representación gráfica y estudio estadístico del tráfico streaming almacenado.

Este segundo script se dedica a procesar fuentes externas de datos de tráfico de red seleccionado, que en este caso son datos almacenados en los csv creados anteriormente, representándolos gráficamente. Mediante la librería *pandas*, se hace uso del concepto de *DataFrame* [16], para encapsular los datos que nos interesen como si de una lista se tratase. Por otro lado, mediante la librería *matplotlib.pyplot* [17] y los métodos que incorpora, se generan figuras con finalidades analíticas distintas. En este caso se pretenden mostrar *stem plots* [18] (o diagramas de tallo en castellano) del tráfico capturado tanto de subida como de bajada, histograma y función de distribución acumulada.

El concepto de *DataFrame* de pandas que se ha utilizado es especialmente útil para el análisis y manipulación de datos, ya que proporciona una amplia gama de funciones y métodos para realizar operaciones eficientes en su estructura de datos tabulares. Permite realizar tareas como filtrar, ordenar, agrupar, transformar y combinar datos de manera sencilla. En la situación que el script nos requiere, los DataFrames han resultado imprescindibles para el filtrado de la información de los distintos ficheros en rangos de tiempo distintos. Este filtrado sirve para separar la estructura entrante de datos de tiempos de tráfico del segundo 0 al segundo 30, y del

segundo, tal y cómo se observa en la siguiente figura, de manera que se puedan representar en dos gráficas distintas, algo interesante de cara al análisis.

```
# Read the CSV file into a pandas DataFrame
df_output = pd.read_csv('output.csv')
df_output_UL = pd.read_csv('output_UL.csv')
df_output_DL = pd.read_csv('output_DL.csv')

x_UL = df_output_UL['P_time']
y_UL = df_output_UL['P_size']

x_DL = df_output_DL['P_time']
y_DL = df_output_DL['P_size']


# Filter data for the desired time intervals
interval_1 = (df_output['P_time'] >= 0) & (df_output['P_time'] <= 30)
interval_2 = (df_output['P_time'] > 30) & (df_output['P_time'] <= 60)
```

Figura 16. Filtrado temporal de datos mediante DataFrames.

Una vez realizado este filtrado, se hará uso de la librería *matplotlib.pyplot* para representar y guardar como archivo local un total de cuatro figuras. La primera representa una muestra del primer minuto de tráfico, distinguiendo tráfico de subida o de bajada en distintos intervalos de tiempo. La segunda, una comparativa de los dos tipos de tráfico en el mismo intervalo de tiempo. La tercera, un histograma que muestra la frecuencia normalizada o densidad de probabilidad de los distintos tamaños paquetes, y la cuarta, su distribución acumulada.

4.1.2.1 – Representación de tráfico en intervalos de tiempo, y su comparativa directa.

Como ya se ha mencionado anteriormente, haciendo uso de los *DataFrames* dejamos estructuras de datos de tiempo de paquetes filtrados de 0 a 30 segundos, y de 30 a 60 segundos. Gracias a esto, podemos representar cuatro diagramas dentro de la primera figura. Los dos ubicados en la parte superior corresponden a los dos intervalos de tiempo de tráfico de subida, y los dos restantes de la parte inferior, a los intervalos de tráfico de bajada.

La distribución de los cuatro diagramas se consigue mediante el método *subplots()* [19] de la librería *matplotlib.pyplot*. Este método crea la figura en base a un conjunto de sub-figuras. Su uso es conveniente para crear diseños de figuras con un formato común, en los que la información representada en cada una puede variar de las demás gracias a su accesibilidad por filas y columnas. Esto es, en la llamada a la creación de la figura, se establecen un número de filas y columnas que conformarán el número y disposición de las figuras. Para instanciar a cada parcela, basta con indicar la fila y la columna en la que se ubican.

Puesto que la causa por la que se crea esta figura no es otra que la de entender los distintos comportamientos del tráfico en función de si es de subida o bajada, se ha elegido el método *stem()* [18], también de la misma librería, para que el tráfico de paquetes quede dibujado en forma de líneas verticales perpendiculares a un eje horizontal. En las figuras siguientes se observa un ejemplo de una representación del tráfico proveniente de una captura de prueba (no es utilizada para extraer resultados ni conclusiones).

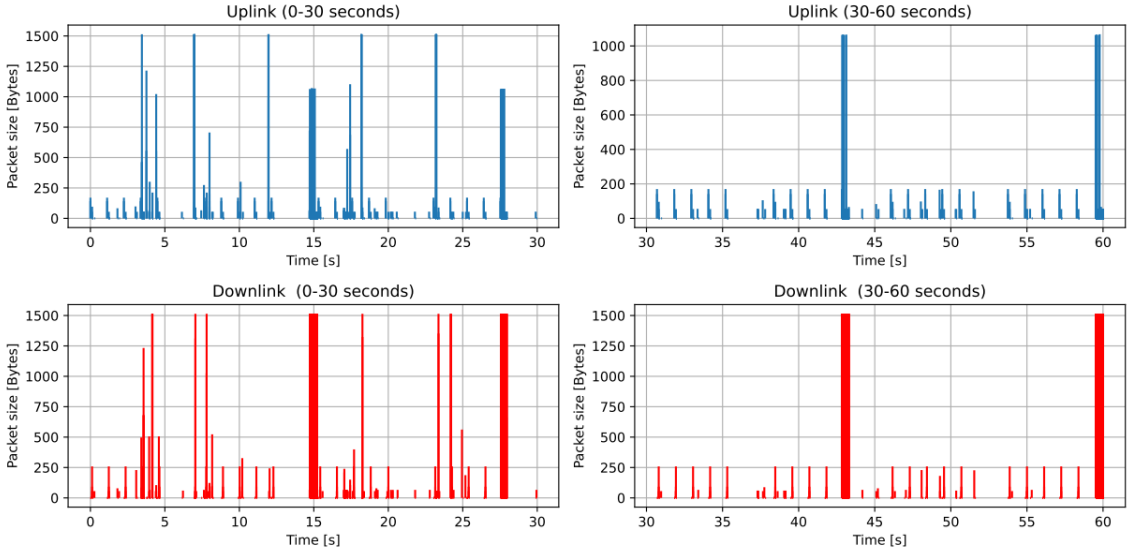


Figura 17. Representación de tráfico de una captura de red usando Netflix en el momento de la captura.

La comparativa directa del tráfico de subida y de bajada es llevada a cabo en otra figura [Fig. 17]. Esta se planteó con la idea de ser complemento a la que muestra los cuatro diagramas ya explicados, debido a que simplemente se centra en un único intervalo de tiempo en el que se aprecian con distintos colores los dos tipos de flujo de paquetes. En este caso es interesante porque ofrece un nivel de detalle superior al anterior, con la salvedad de observar solamente uno de los dos intervalos de tiempo.

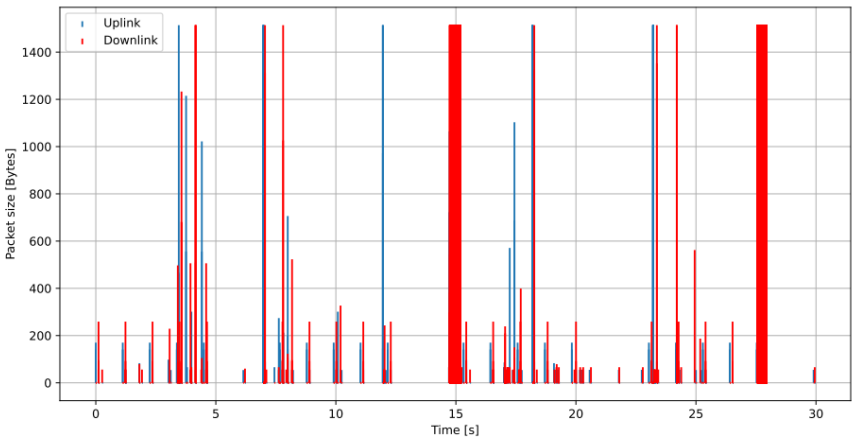


Figura 18. Comparativa directa: Uplink vs Downlink [0 - 30 s]

4.1.2.2 – Representaciones estadísticas. Histograma y Función de Distribución Acumulada.

La visión que se pretende dar con la representación de un histograma es la de entender según si estamos observando tráfico de subida o de bajada, cuál es la frecuencia con la que se suceden los distintos tamaños de paquete existentes. Para ello, se necesita un código Python que no solo genere un histograma, sino que además represente los valores de dicho histograma normalizados entre 0 y 1. De esta manera, se conseguiría observar en cierto modo la densidad de probabilidad de los tamaños de paquetes.

Los métodos planteados para la consecución de un histograma de sendas características son *hist()* [20] y *bar()* [21], ambos pertenecientes a la librería de *matplotlib* que ya se viene utilizando. El primer método es el encargado de computar en una primera instancia el histograma. Entre los distintos parámetros que se dan de entrada, se añade *density = 'True'*, puesto que este es el encargado de devolver la densidad de probabilidad de los valores del histograma. Posteriormente se utiliza el segundo método para llevar a cabo la representación finalmente normalizada, la cual se consigue pasando a este último método como parámetro de entrada los valores del histograma que ya hemos computado divididos entre el sumatorio de dichos valores.

Respecto a la representación de la función de distribución acumulada del tamaño de los paquetes, se ha seguido el siguiente enfoque [22]. Para obtener valores normalizados de probabilidad acumulada, primero se deben ordenar de forma ascendente los valores de tamaño de paquetes en un array. Esto se consigue utilizando el método *sort()* [23] de la librería *numpy*. Posteriormente, se dividen por el sumatorio de elementos de cada array para que queden normalizados a 1 (ídem a la normalización en el histograma). De esta manera, ya que los valores de tamaño de paquetes han sido previamente ordenados, la representación de este nuevo array que contiene estos mismos valores también normalizados corresponderá a una estimación de la función de distribución acumulada.

Es interesante aplicar esta curva para ampliar el entendimiento del tráfico de la red a analizar, de una forma análoga a lo que hace el histograma en este breve estudio estadístico. A continuación, se muestran en la figura las representaciones del histograma y de la función de distribución acumulada sacadas a partir de la misma captura de tráfico de prueba utilizada para obtener las figuras anteriores.

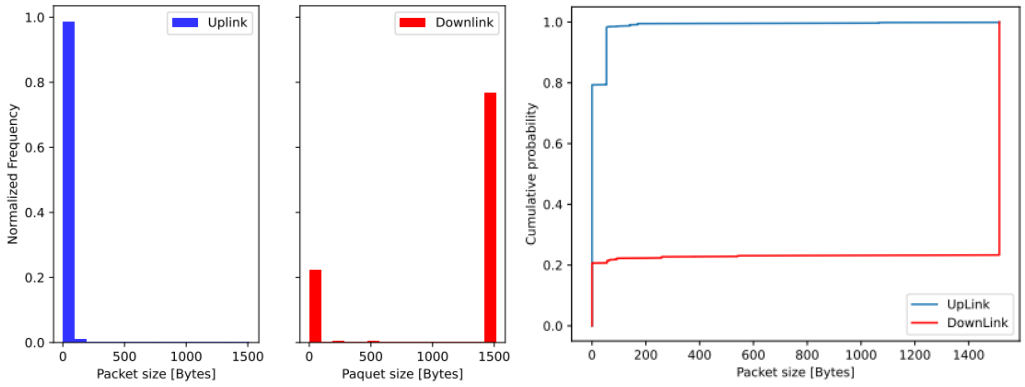


Figura 19: Histogramas y Función de Distribución Acumulada de tráfico Uplink vs Downlink.

4.2 – Caracterización de modelos de consumo energético de los servicios de streaming multimedia.

Al igual que sucede con la caracterización de los distintos modelos de tráfico de streaming, también se ha hecho uso de Python para desarrollar una solución que permita analizar el impacto de los servicios de streaming de contenido multimedia en el consumo energético en distintos equipos de usuario. De nuevo este desarrollo comenzaría con el "parseo" de alguna captura de tráfico realizada con Wireshark, ya explicado en la sección anterior. Quedaría resumido en el diagrama siguiente:

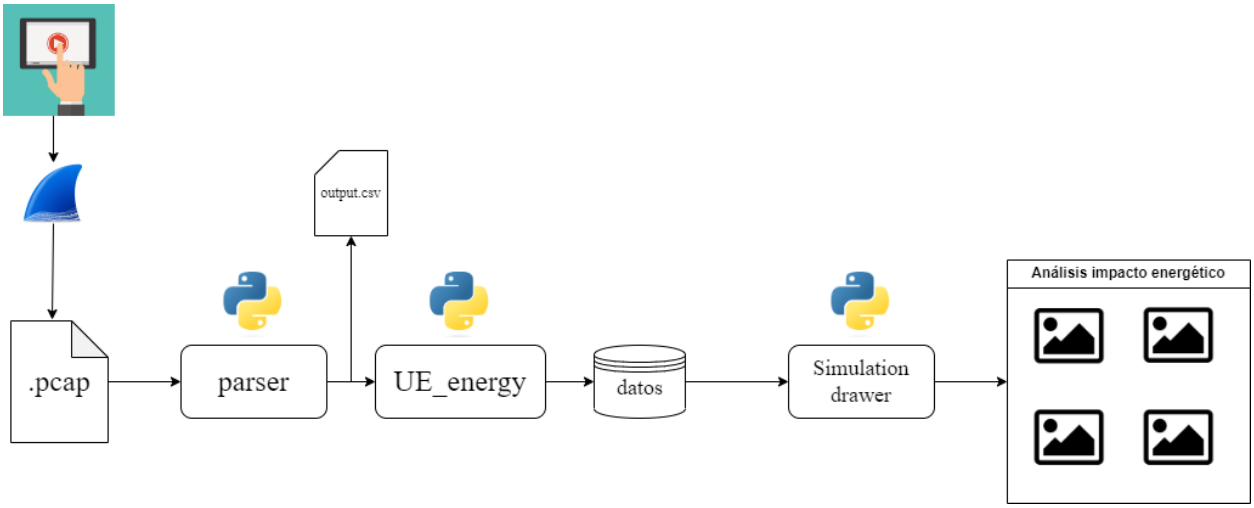


Figura 20. Diagrama representativo del proceso de caracterización del impacto energético de diferentes servicios streaming en UEs.

4.2.1 – UE_energy. Simulador de consumo energético en UEs.

Este ejecutable de Python es un simulador de eventos discretos utilizado para obtener información relativa al impacto en el consumo energético de un UE al simular el tráfico generado por la visualización de contenido multimedia de los servicios de streaming capturados en un dispositivo conectado a una red (hay intercambio de paquetes de subida y bajada). Dicho esto, considero necesario esclarecer que el desarrollo de la función principal en el código que corresponde a este script, *delay_energy()*, forma parte del trabajo de investigación que lleva a cabo mi tutor, José Manuel Giménez Guzmán, quién ha decidido prestarme su uso para el propósito de este proyecto. No obstante, sí que forma parte de este trabajo fin de grado la adaptación de dicho simulador al propósito planteado, dado que la entrada de datos en el caso que nos ocupa se encuentra en ficheros csv con un determinado formato.

A continuación, se explica qué es un simulador de eventos discretos, por qué se ha decidido usar en el ámbito del trabajo y su funcionamiento. Un simulador de eventos discretos es una herramienta computacional utilizada para modelar y analizar sistemas en los que los cambios en

el estado del sistema ocurren en momentos discretos y específicos en el tiempo, en lugar de cambios continuos.

En otras palabras, se centra en eventos específicos que ocurren en puntos discretos en el tiempo y que afectan al estado del sistema. Los simuladores de eventos discretos son ampliamente utilizados para simular sistemas complejos en diversos campos como la ingeniería, la investigación operativa, la logística, la manufactura, la ciencia de la computación, la biología, entre otros. Estos sistemas pueden ser desde procesos industriales y logísticos hasta sistemas de comunicación y redes, como es el caso.

El simulador sigue una lógica donde se define el estado inicial del sistema para luego programar eventos específicos que pueden cambiar el estado del sistema en momentos determinados. Estos eventos pueden ser cualquier cosa, desde llegadas de pedidos en una cola hasta la finalización de una tarea en una cadena de producción. En el script, los eventos programados tienen que ver con la recepción o transmisión de paquetes en un dispositivo (UE) conectado a una red inalámbrica.

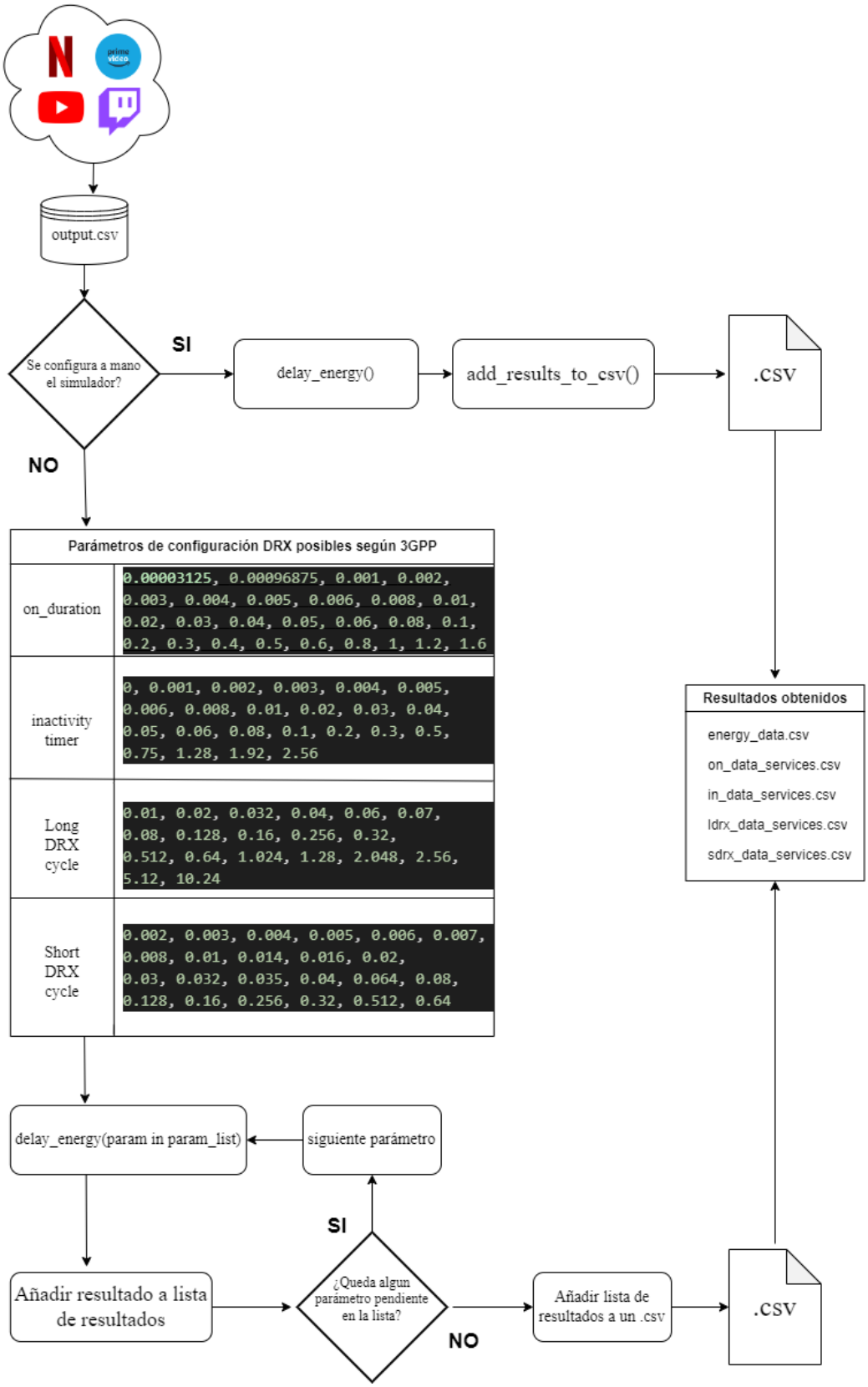
El desencadenante temporal que lanza un evento está determinado por una lista de diferencias de los instantes de tiempo almacenados en los ".csv" externos resultado de la ejecución del "parser", ya mencionado en este capítulo. En cada paso de tiempo, el simulador avanza al próximo evento en orden cronológico y actualiza el estado del sistema en función de lo que ocurrió en ese evento.

Ya que se ha decidido estudiar la Recepción Discontinua (*DRX – Discontinuous Reception*) [11][12] como principal mecanismo de ahorro energético en este proyecto, el simulador programa los eventos que se suceden en función de un ciclo temporal en el que surge la oportunidad de utilizar DRX, especificado en el estándar 3GPP [12]. Estos eventos son: "ON", "OFF", "END_INACTIVITY TIMER", "DL_PACKET" y "UL_PACKET". Los tres primeros resultan obvios una vez entendido el contexto tecnológico descrito en el capítulo 2 de esta memoria, en su sección correspondiente; y los dos restantes, corresponden con la recepción de paquetes de subida (UL - *Uplink*) o de bajada (DL - *Downlink*). Estos se programan para que se ejecuten en un orden cronológico determinado por la lista de diferencia de instantes de tiempo que ya se ha comentado. Todo el proceso de simulación en base a estos eventos se repite hasta que se alcanza un criterio de finalización, que en este caso es terminar de recorrer toda la lista de diferencias de tiempo.

La inclusión por tanto de este simulador dentro del proyecto corresponde con la idea de obtener resultados que permitan realizar un análisis del impacto energético que tiene en los UEs la visualización del contenido multimedia que nos ofrecen distintos servicios de streaming.

En una primera instancia, la función principal del simulador devuelve tres valores que permiten estudiar este impacto energético. Estos son el factor de ahorro energético (*PSF - Power Saving Factor*), el retardo en la recepción y el retardo adicional en la transmisión o recepción de paquetes (*Delay_RX/TX*) resultado de la aplicación del DRX como mecanismo de ahorro de energía.

Figura 21. Diagrama representativo del uso del simulador de eficiencia energética.



Resulta trascendental explicar cómo se ha trabajado con este simulador, y de qué manera ha sido posible obtener resultados. Con la figura anterior [Fig. 21], se pretende explicar de manera visual el procedimiento llevado a cabo en esta utilización del simulador para analizar el impacto energético producido por los diferentes servicios de streaming multimedia en UEs. Al igual que ocurre con el ejecutbale que representa los datos de tráfico capturado, es necesario obtener primero los valores de tiempo y de tamaño de los paquetes de la captura almacenados en los "csv" externos (*output.csv*).

Posteriormente se define la configuración de los parámetros de entrada con los que el simulador va a aplicar el mecanismo DRX. Si se decide configurar manualmente estos parámetos, es decir, establecer su único valor para la ejecución del DRX, entonces se llamará a la función principal de este simulador "*delay_energy()*", que imprimirá por pantalla los valores resultado correspondientes para el PSF (*Power Saving Factor*), el retardo en la transmisión, y el retardo en la recepción, pudiendo de manera opcional utilizar otra función externa "*add_results_to_csv()*" para guardar el resultado de esta ejecución en un archivo .csv, en un registro identificado por el nombre del servicio streaming al que corresponde la captura de tráfico utilizada.

Si de lo contrario se decide configurar mediante listas con todos los valores posibles de entrada para los parámetros definidos para el mecanismo DRX, se elegirá de entre los cuatro parámetros de entrada (*On duration timer, Inactivity timer, Long DRX cycle* y *Short DRX cycle*) cuál se desea configurar mediante una lista con todos los valores que permite el estándar 3GPP en [11]. Los tres parámetros restantes deberán obtener su valor manualmente. Entonces, la función principal del simulador, "*delay_energy()*", se ejecutará tantas veces como elementos haya en la lista de valores posibles para ese parámetro. Al finalizar, almacenará de una sola vez las listas de resultados obtenidas por el simulador en otros archivos .csv, igual que en el caso de configurar manualmente los parámetros, solo que con muchos más resultados simultáneos.

Se ha hecho uso de este tipo de configuración automática precisamente para entender de una forma visual, dentro del estudio de este proyecto de qué manera afecta la variación de un solo parámetro de configuración DRX en el crecimiento o decrecimiento del valor del PSF, es decir, un análisis de sensibilidad, por lo que su implementación era necesaria. Esta relación se estudiará más adelante en el capítulo correspondiente al análisis de resultados.

Además, que la configuración de los parámetros de entrada del mecanismo DRX del simulador quede a elección del usuario, se ha planteado por habilitar consecuentemente la elección en el tipo de resultados que se desean obtener. El simulador es una herramienta muy potente, que permite incluso obtener resultados de simulación de impacto energético a una escala tan grande como se desee imaginar, lo cual siempre es interesante de cara a comparar la eficiencia energética de varios servicios de streaming de contenido multimedia de forma simultánea, sin embargo, es posible que únicamente se pretenda probar alguna configuración de parámetros de DRX concreta, resultando más cómoda la configuración manual diseñada para el simulador en vez de la otra. De ahí que exista esta doble posibilidad a la hora de usar este simulador para caracterizar el tráfico.

Por otro lado, es asumible decir que al utilizar la configuración automática del DRX mediante listas, se obtengan resultados complejos de interpretar, ya que siempre debe existir una relación entre los valores de los parámetros determinada por el tipo del tráfico que se esté simulando. Sin embargo, en el momento de recorrer una lista compuesta a la vez por valores muy pequeños y valores muy grandes para cada parámetro, se observa que esta relación se vuelve compleja o deja de existir, de manera que los resultados que se obtienen no son sencillos de comprender.

Es por ello por lo que, además de las formas de utilizar el simulador ya mencionadas, se decide considerar otra que simplifica mucho más los resultados, conservando ciertos aspectos de la configuración automática, tal y como se muestra a continuación [Fig. 22]. En el capítulo siguiente, "*Análisis de resultados*", se explicarán mediante tablas los parámetros utilizados en cada configuración, que constan de valores pequeños, medianos y grandes con el objetivo de simplificar el análisis de la variación del Factor de Ahorro Energético (*PSF – Power Saving Factor*) y del retardo para cada configuración utilizada.

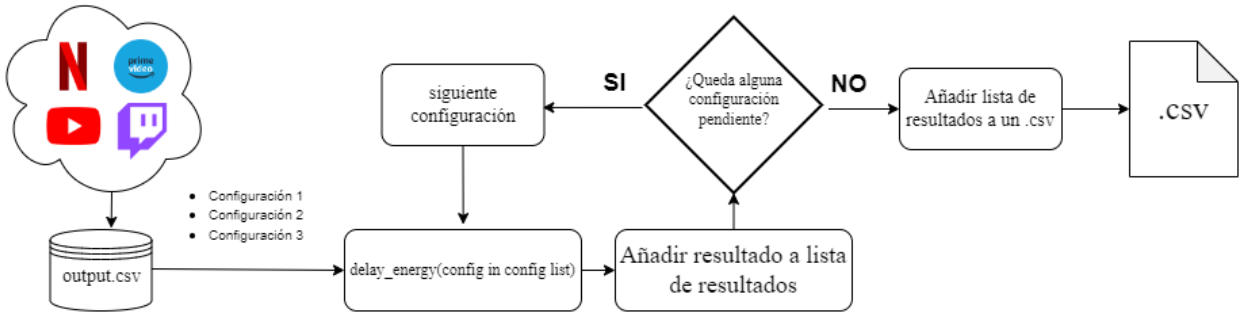


Figura 22. Diagrama representativo de la simplificación de la configuración automática del simulador.

4.2.2 – Simulation Drawer. Representación gráfica de los resultados obtenidos a partir de la utilización del simulador.

En el primer diagrama que indica cómo se ha llevado a cabo la caracterización de la eficiencia energética de los distintos servicios de streaming en UEs [Fig. 20] aparece este archivo como el último bloque dentro de la estructura de archivos. Verdaderamente más que un archivo, son un conjunto de archivos, cada uno encargado de representar distintas gráficas a raíz de los resultados obtenidos en el simulador que también han sido almacenados en archivos ".csv", sin embargo, en el diagrama se ha optado por representar uno único para facilitar su entendimiento.

En primer lugar, se busca representar como afecta la variación de los distintos valores establecidos en el estándar 3GPP [11][12] para los parámetros de configuración DRX, al PSF (*Power Saving Factor*) y al retardo en la recepción. De ahí la importancia de la configuración

automática del DRX explicada en el apartado anterior, ya que ha permitido facilitar la consecución de estas representaciones.

Según lo estudiado en el capítulo segundo de este proyecto "*Contexto Tecnológico*", referente a la Recepción Discontinua y al estudio mencionado en [11] y [12], se entiende que las variaciones en los parámetros de configuración del DRX tendrán un impacto directo en el PSF y en el retardo producido fruto de la aplicación de este mecanismo de ahorro energético en UEs. Posteriormente, en el análisis de resultados se desarrollará de forma exhaustiva el resultado de esta variación, además de las complejas relaciones existentes.

En segundo lugar, una vez preparados para entender cómo afecta cada parámetro de la configuración DRX al impacto energético, es necesario entender también de qué manera afecta al consumo energético la utilización de unos servicios de streaming de contenidos multimedia u otros, de nuevo observando los valores obtenidos para el PSF y para los retardos en la recepción y la transmisión de paquetes.

Como ya se ha mencionado antes, la necesidad de representar varios gráficos distintos de mayor complejidad ha llevado a la utilización de varios scripts de Python distintos, a diferencia del proceso de caracterización del tráfico de paquetes generado por los distintos servicios de streaming, que solo utilizaba uno.

Sin embargo, a nivel de programación, se han utilizado exactamente las mismas librerías, ya que únicamente se ha trabajado con gráficos de barras (*plt.bar*)[21] y con curvas simples, por lo que no se considera necesario resaltar aquí ningún código desarrollado puesto que a pesar de trabajar con valores distintos, utiliza los conceptos ya explicados del *DataFrame*, o de personalización de gráficas de *matplotlib*, resultando de manera indeseada en algo repetitivo.

Capítulo 5: Análisis de resultados

Este capítulo final sirve para mostrar de manera visual el resultado de los procesos de caracterización llevados a cabo, tanto para los modelos de tráfico generado por las distintas plataformas de streaming de contenido multimedia como para el consumo energético generado por dicho tráfico en un dispositivo.

En primer lugar, se introducirán los aspectos tecnológicos más importantes a tener en cuenta de las muestras utilizadas en el proceso de caracterización, es decir, las capturas de tráfico hechas con Wireshark, como pueden ser posibles códecs de compresión, resolución, protocolos de transmisión, etc. Estos afectan, en mayor o menor medida a la tipología del tráfico generado, por lo que resultan fundamentales para la comprensión de los modelos obtenidos. Posteriormente se discutirán los resultados gráficos que conforman los modelos de tráfico de Netflix, YouTube, Prime y Twitch, para después analizar los resultados obtenidos en el simulador de consumo energético.

Consecuentemente, se compararán ciertos resultados para extraer conclusiones definitivas sobre el estudio realizado, tanto para el tráfico de paquetes, como para el impacto energético, fruto de la comunicación de dispositivos con los servicios de streaming, para así poder finalizar el entendimiento de las particularidades y características propias de cada servicio que tienen una repercusión directa en la su eficiencia energética a la hora de la distribución y reproducción del contenido multimedia.

5.1 - Introducción a las muestras extraídas de los distintos servicios de streaming.

El conjunto de servicios seleccionado para extraer las muestras de tráfico se conforma de aquellos más relevantes a día de hoy, como son Netflix, YouTube, Prime Video y Twitch, debido, entre diversos motivos, a sus características tecnológicas que garantizan la entrega del contenido en alta calidad a los usuarios y a su contundente presencia en los hábitos de consumo de la sociedad actual. Para cada uno de estos servicios se ha realizado una muestra de tráfico con el software Wireshark, correspondiente a la reproducción de contenido en streaming de aproximadamente 60 segundos de duración. Todas estas muestras han sido tomadas además teniendo en cuenta las siguientes características, relacionadas con el marco tecnológico desarrollado en el capítulo segundo de esta memoria:

- **Netflix:** Emplea H.264 y H.265 como códecs de compresión de vídeo, además de MPEG-DASH y HLS como protocolos de transmisión adaptativa. Netflix opera con una amplia CDN (*Content Delivery Network*) con servidores distribuidos en todo el mundo para reducir la latencia y mejorar la entrega de contenido. La muestra tomada con Netflix corresponde a la reproducción de un fragmento de *The Witcher S1:E1*, con la idea de obtener tráfico de paquetes correspondiente a la resolución disponible de contenido en Ultra HD (4K).

- **Prime Video**: Utiliza los mismos códecs de compresión y protocolos de transmisión adaptativa que Netflix. Es uno de los principales competidores del anterior servicio, siendo además interesante por su CDN propia (*Amazon Web Services*), cuya escalabilidad y fiabilidad podrían resultar en un perfil de tráfico distinto. La muestra obtenida de Prime Video corresponde con un fragmento de la serie *The Boys S1:E1* también disponible en resolución Ultra HD (4K).

- **YouTube:** Utiliza VP9 y recientemente AV1 como códecs de compresión de vídeo. También utiliza MPEG-DASH como protocolo de transmisión adaptativa, sin embargo, a diferencia del resto de servicios, YouTube utiliza QUIC como protocolo de transporte, lo cual ha sido determinante en el análisis de paquetes de la captura. En este caso, y siguiendo con la línea de reproducción de contenido de resolución Ultra HD (4K), se ha tomado una muestra de un vídeo de paisajes grabados con un equipo que permite luego reproducir esta resolución, también de 60 segundos, como las demás.

- **Twitch**: Los códecs de compresión utilizados y CDN entre otras características, son los mismos que Prime Video, debido a que Twitch pertenece a la empresa Amazon al igual que Prime. Sin embargo, Twitch es un servicio de streaming en directo, por lo que emplea protocolos de transmisión en tiempo real, como RTMP (*Real Time Messaging Protocol*), siendo la resolución del contenido un factor limitante en la transmisión. No es una consecuencia obligatoria, sin embargo, al ser streaming en directo (*Live Streaming*) la mayor resolución a la que se puede reproducir el contenido es Full HD (1080p), por lo que sería asumible pensar que el perfil de tráfico generado por este servicio no resultaría comparable al del resto de servicios. Sin embargo, debido a su trascendencia en el panorama actual no deja de ser interesante comparar las diferencias existentes en el tráfico capturado. La muestra correspondiente a este tráfico ha sido obtenida durante la reproducción de un contenido aleatorio dentro de esta plataforma, por supuesto cerciorándose de la resolución Full HD deseada para realizar esta comparación.

5.2 - Modelos resultado del tráfico de Netflix, YouTube, Prime y Twitch.

A continuación, se tratará de mostrar desde un punto de vista analítico tanto las diferencias y similitudes del tráfico entre los servicios de streaming de contenido como las peculiaridades de cada uno.

En las cuatro figuras siguientes se observa una representación del tráfico de paquetes capturados por todos y cada uno de los servicios mencionados, dividido en dos intervalos de tiempo, de 0 a 30 segundos y de 30 a 60 segundos. También se observa que se ha diferenciado el flujo de paquetes de subida del de bajada. Estas representaciones buscan mostrar el flujo y las dimensiones de los paquetes que se intercambian durante la comunicación del dispositivo con el servicio de streaming en cuestión, para intentar entender comportamientos similares o diferentes entre ellos.

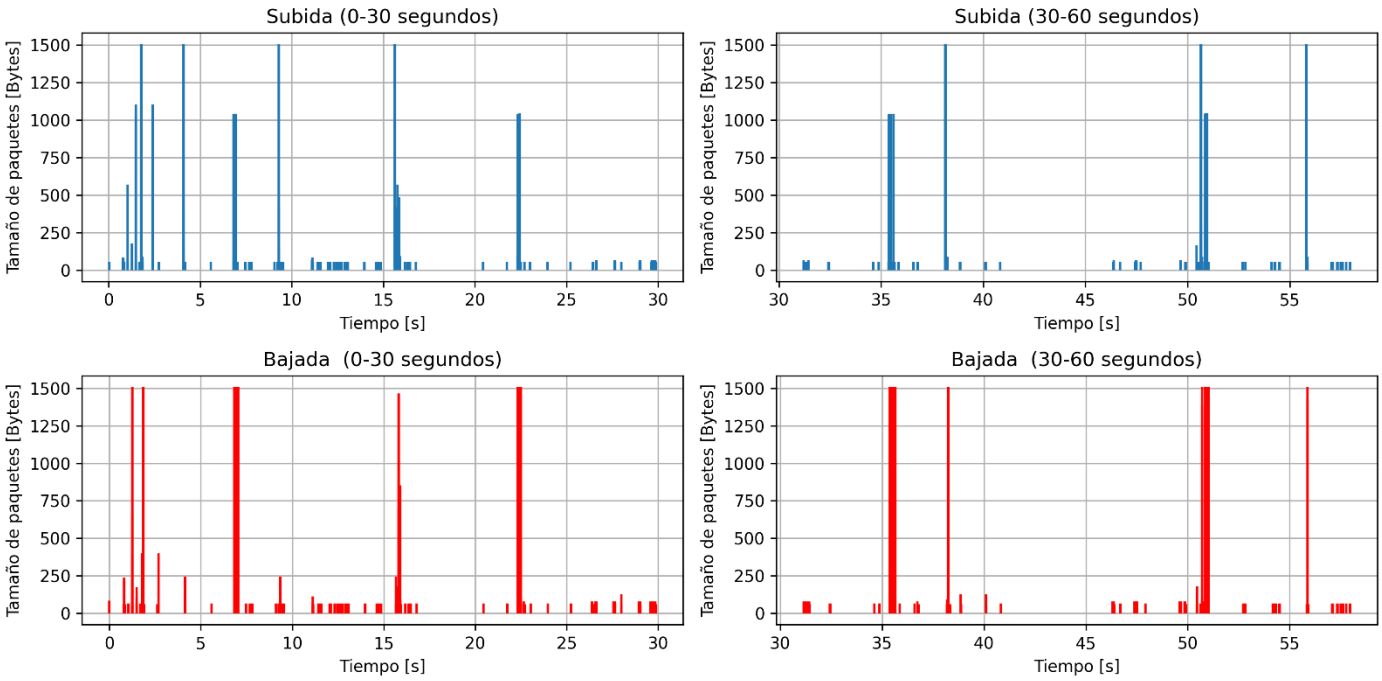


Figura 23. Muestra de tráfico de red obtenida de la reproducción de contenido multimedia en Ultra HD (4K) en Netflix. La muestra contiene 7666 paquetes de los cuales 7491 pasan el filtrado TCP. Throughput medio de la captura: 1.1555 Mbps

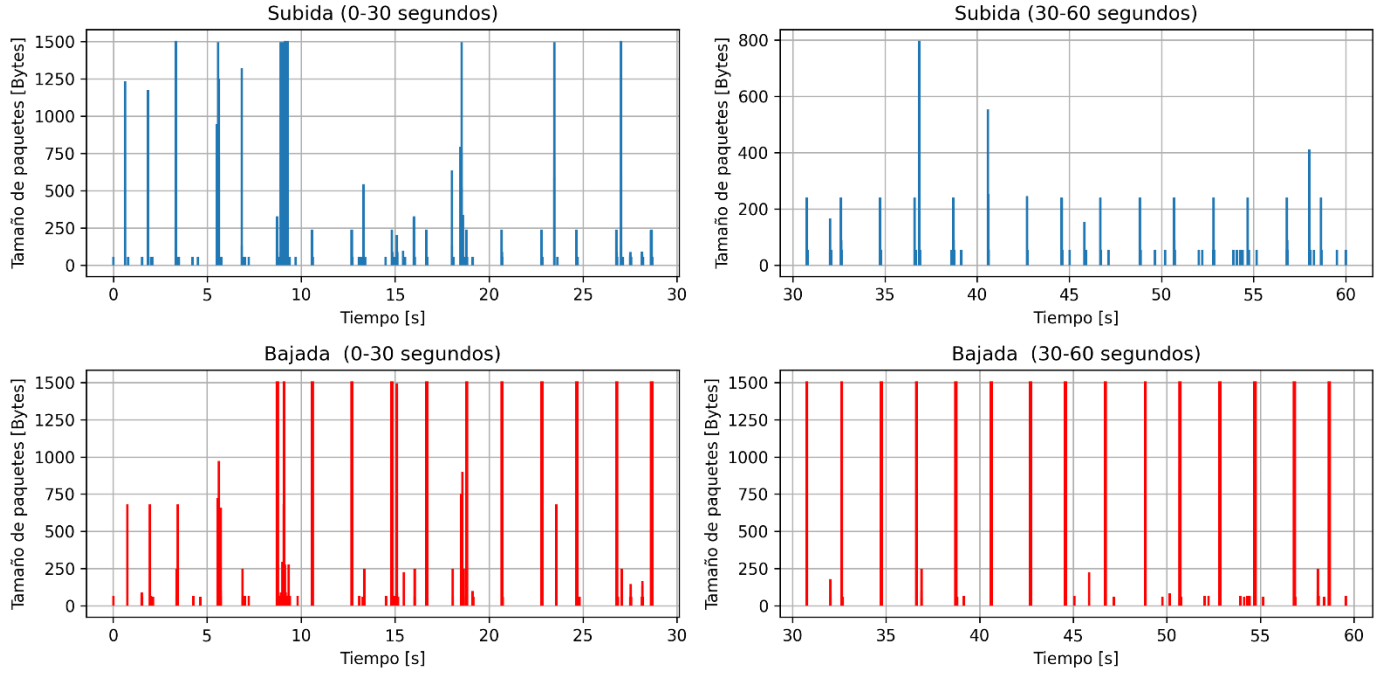


Figura 24. Muestra de tráfico de red obtenida de la reproducción de contenido multimedia en Ultra HD (4K) en Prime Video. La muestra contiene 11022 paquetes de los cuales 10729 pasan el filtrado TCP. Throughput medio de la captura: 1.2742 Mbps

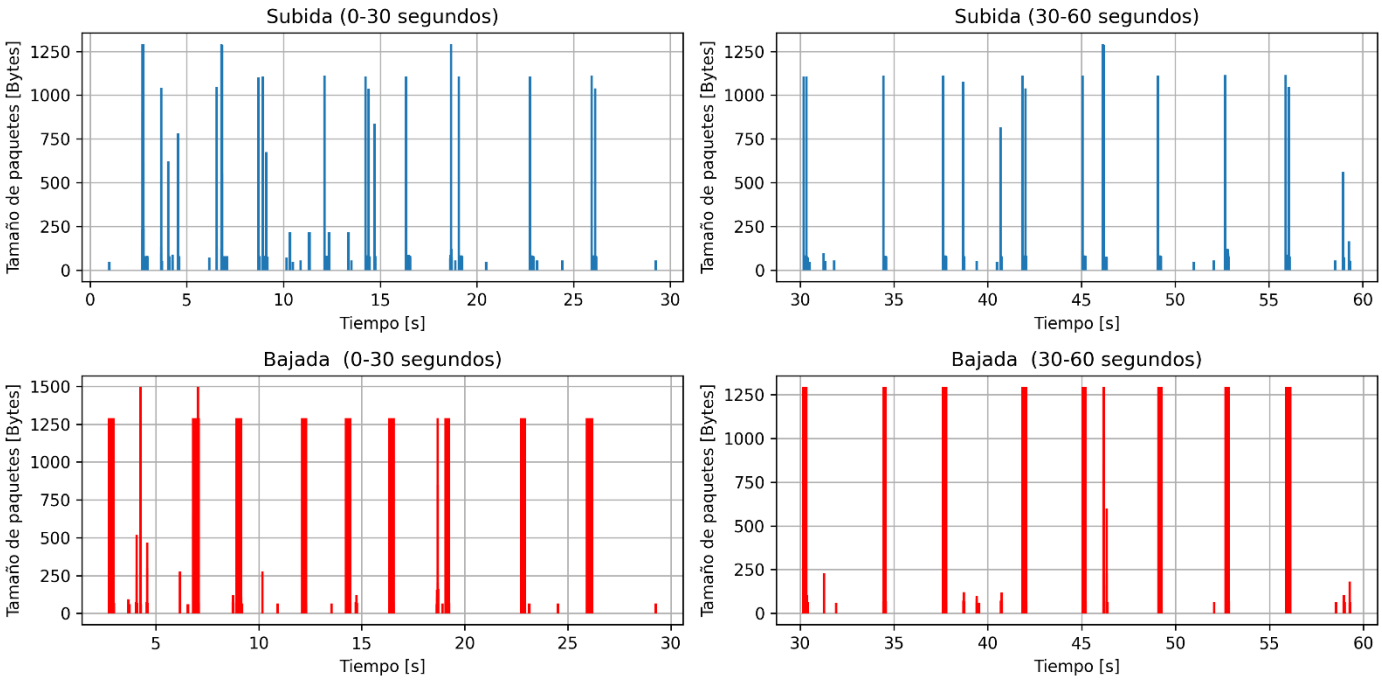


Figura 25. Muestra de tráfico de red obtenida de la reproducción de contenido multimedia en Ultra HD (4K) en YouTube. La muestra contiene 97158 paquetes de los cuales 97099 pasan el filtrado TCP. Throughput medio de la captura: 14.2568 Mbps

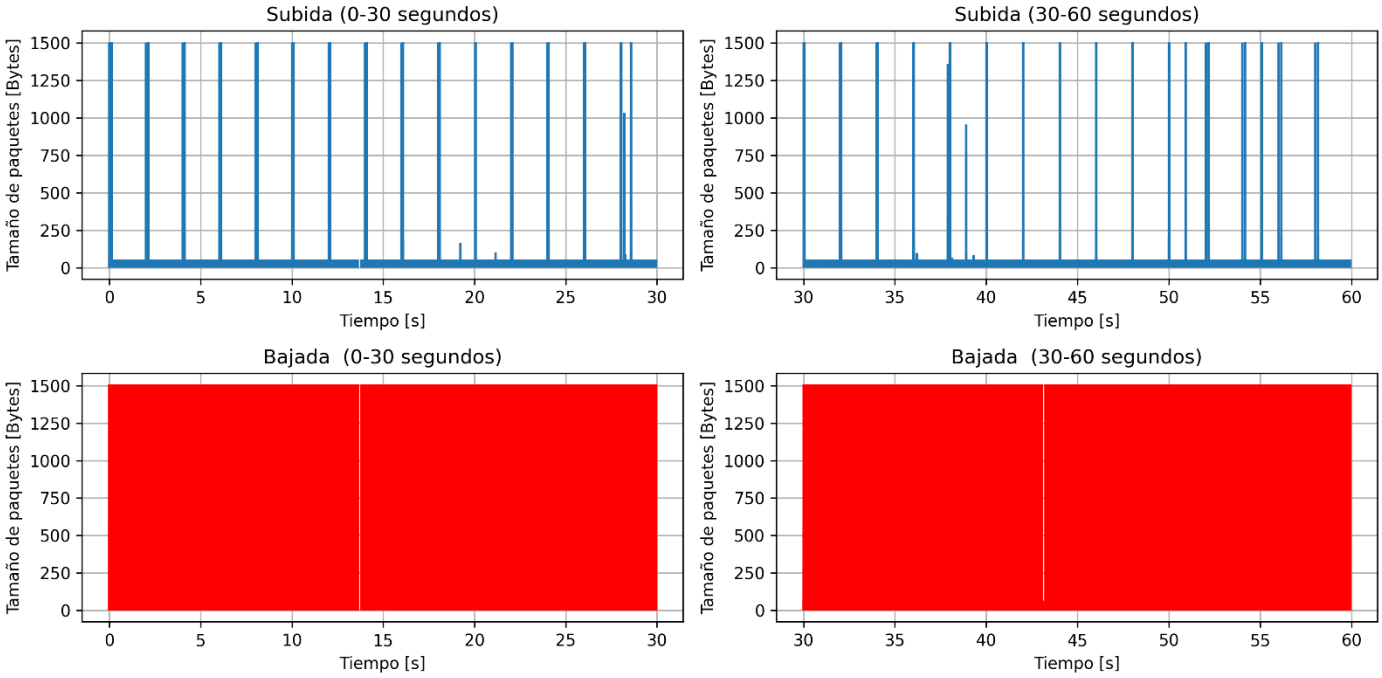


Figura 26. Muestra de tráfico de red obtenida de la reproducción de contenido multimedia en Full HD (1080p) en Twitch. La muestra contiene 45099 paquetes de los cuales 44983 pasan el filtrado TCP. Throughput medio de la captura: 6.8221 Mbps

Como complemento a estas gráficas se han utilizado tanto histogramas como funciones de distribución acumulada para determinar la probabilidad normalizada a 1 con la que aparece un determinado tamaño de paquete.

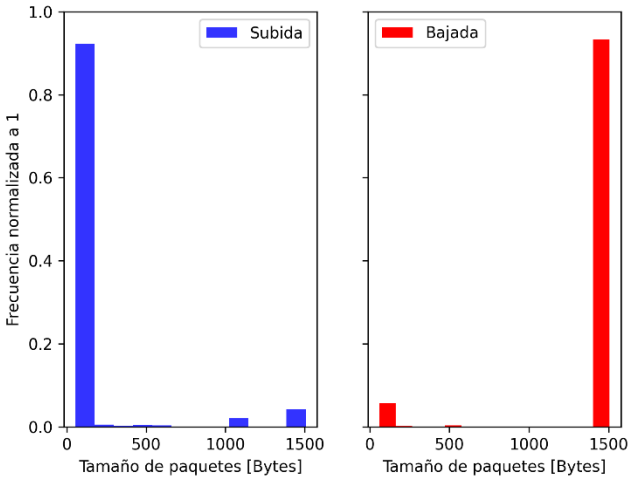


Figura 27. Histograma extraído de muestra de tráfico de Netflix.

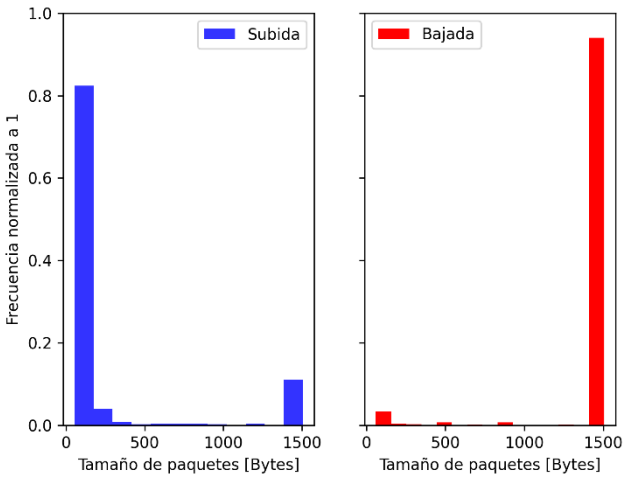


Figura 28. Histograma extraído de muestra de tráfico de Prime Video.

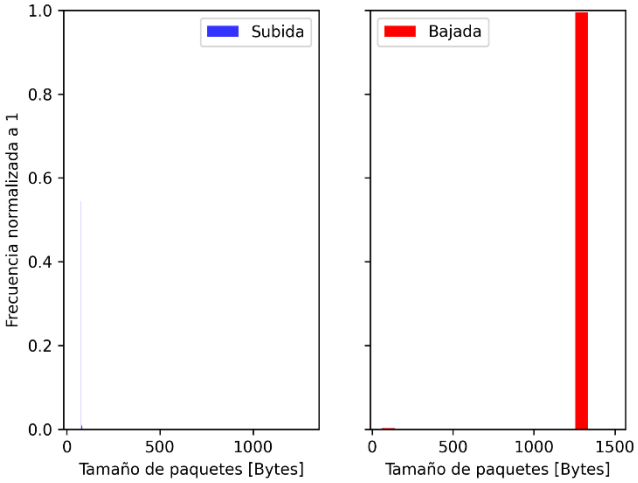


Figura 29. Histograma extraído de muestra de tráfico de YouTube

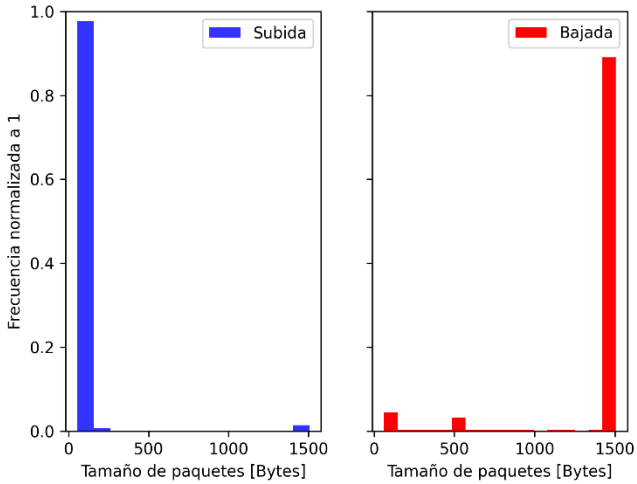


Figura 30. Histograma extraído de muestra de tráfico de Twitch

En [Figs. 27, 28 y 30] se observa que tanto para Netflix, Prime Video y Twitch el perfil de distribución de tamaños de paquetes es realmente similar. Para el tráfico de subida la mayoría de los paquetes se distribuyen en torno a tamaños pequeños (de 0 a 200 Bytes) a excepción de una pequeña porción de grandes tamaños, en cambio, para el tráfico de bajada se distribuyen de manera opuesta; la mayoría en torno al máximo tamaño que permite la red (1452 Bytes).

Esto no ocurre exactamente en el caso de YouTube [Fig. 29]. Se observa también una concentración casi absoluta de paquetes en torno a grandes tamaños en el tráfico de bajada, sin embargo, no existe apenas acumulación de paquetes en torno a un rango de tamaños en tráfico de subida. Aparentemente se trataría de aproximadamente un único tamaño de paquetes de subida, aunque esta escasa acumulación probablemente se deba a la relación de paquetes de subida respecto a paquetes de bajada de la captura. Al calcular el valor del *throughput*, la captura de tráfico de YouTube es la que presentaba un mayor valor, además de un número de paquetes interesantes notablemente superior al del resto, por lo que es asumible pensar que un número pequeño de tamaños de paquetes de subida respecto al cómputo de paquetes en este caso resulte en un histograma de este aspecto. Continuando con las funciones de distribución acumulada, vemos también sus representaciones gráficas para cada servicio, sirviendo de apoyo al análisis estadístico que proporcionan los histogramas:

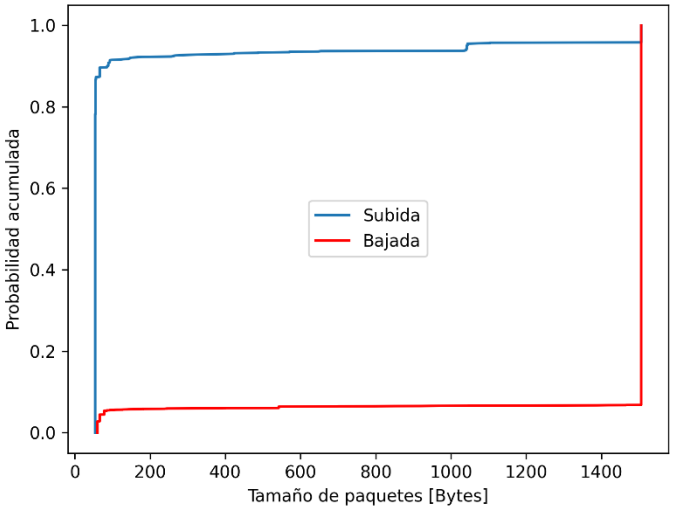


Figura 31. Función de Distribución de la muestra de tráfico extraída de Netflix.

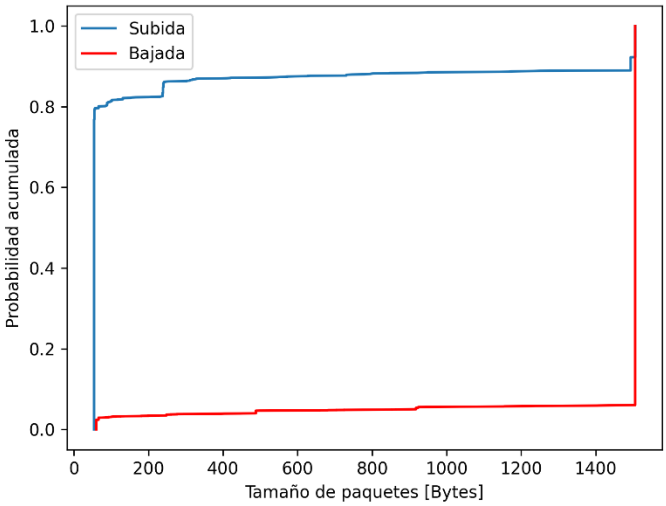


Figura 32. Función de Distribución de la muestra de tráfico extraída de Prime Video.

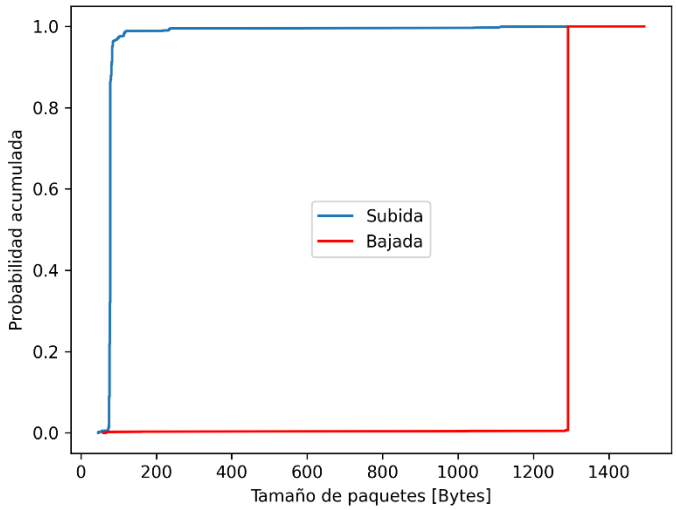


Figura 33. Función de Distribución de la muestra de tráfico extraída de YouTube.

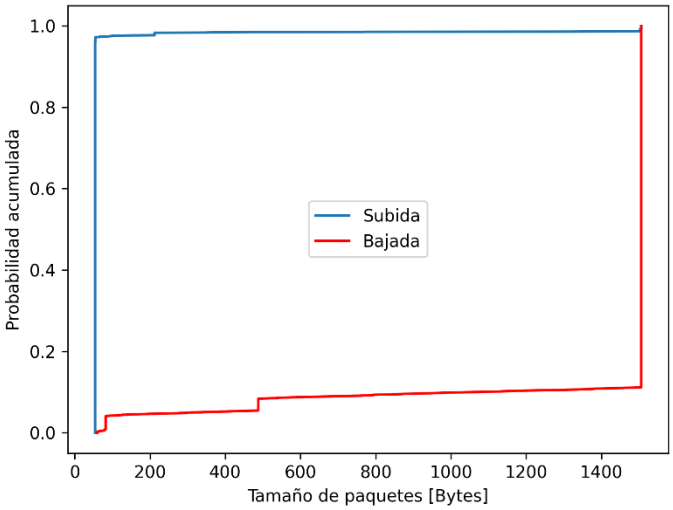


Figura 34. Función de Distribución de la muestra de tráfico extraída de Twitch.

5.2.1 – Conclusiones extraídas.

Las particularidades que han surgido fruto de las representaciones de histogramas y probabilidad acumulada [Figs. 27 a 34] ya han sido comentadas en la sección anterior. Sin embargo, los resultados que representan el flujo del tráfico de paquetes de todos los servicios estudiados [Figs. 23 a 26] no han sido comentadas por el momento. De ahí el propósito de esta sección. Las posibles conclusiones que se pueden extraer a raíz de estos resultados gozan de una trascendencia que se ve incluso enriquecida una vez visto el breve análisis estadístico que ofrecen el resto de representaciones. Por ello, es necesario desarrollarlas de la forma más detallada posible en este apartado.

En primer lugar, se observa que Twitch consta de un perfil de tráfico característico en el que constantemente se reciben paquetes de bajada, debido a que es un servicio de streaming multimedia en directo. Hay mucha diferencia con el tráfico de subida, el cual solo se muestra como muchas ráfagas periódicas. Quizás era sencillo imaginar el aspecto que tendría el tráfico de bajada de este servicio por el hecho de suceder en directo, sin embargo, gracias a la representación obtenida también para el tráfico de subida, se ha obtenido este perfil de ráfagas de paquetes que realmente transcurre.

Precisamente, se observa también un perfil de tráfico en Prime Video, YouTube y Netflix caracterizado por la sucesión de paquetes en forma de ráfagas. Sin embargo, estas se producen con distinta periodicidad. Prime parece que recibe ráfagas de paquetes con un periodo de alrededor de 2 segundos, mientras que YouTube alrededor de 5 segundos y Netflix, 8 segundos.

Entre estos, resulta especialmente interesante el perfil de tráfico de Netflix puesto que aparentemente el volumen de tráfico es mucho menor al del resto de servicios. Si nos fijamos en el tamaño de las ráfagas de paquetes se puede llegar a suponer que en el caso de Netflix tienen un grosor superior al de las ráfagas producidas por otros servicios, que da la impresión de una alta concentración de paquetes en ese instante concreto. Sin embargo, esta impresión, sumada a la escasez de flujo existente en los largos periodos de tiempo entre ráfagas puede llegar a desconcertar. Siendo que además esta muestra es la que obtiene el mínimo valor de *throughput* medio, denota una fuerte diferenciación en cómo se produce la descarga de contenido respecto al resto de servicios.

Por ello, se decide complementar a las figuras resultado-anteriores [Figs. 23 a 26], con una representación del acumulado de *bytes* que se sucede a lo largo de toda la captura, de manera que se pueda cuantificar la cantidad de información intercambiada en cada ráfaga.

Estas representaciones aportan un gran valor analítico, puesto que, a diferencia de las anteriores, demuestran el volumen de bytes que sucede en cada ráfaga de paquetes. Con las gráficas anteriores se podía observar la periodicidad con la que sucede el intercambio de paquetes en la comunicación, pero debido al alto número de paquetes de cada muestra resultaba realmente complejo estimar cuantos datos se intercambiaban en cada momento. Lo más llamativo de estos resultados es sin duda la forma de escalera en la que se recibe volumen de tráfico de paquetes.

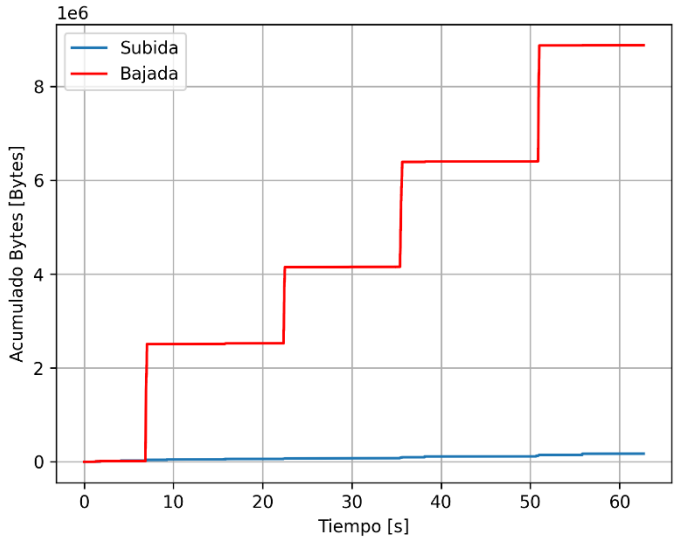


Figura 35. Representación del acumulado de bytes correspondiente a la muestra de tráfico de Netflix.

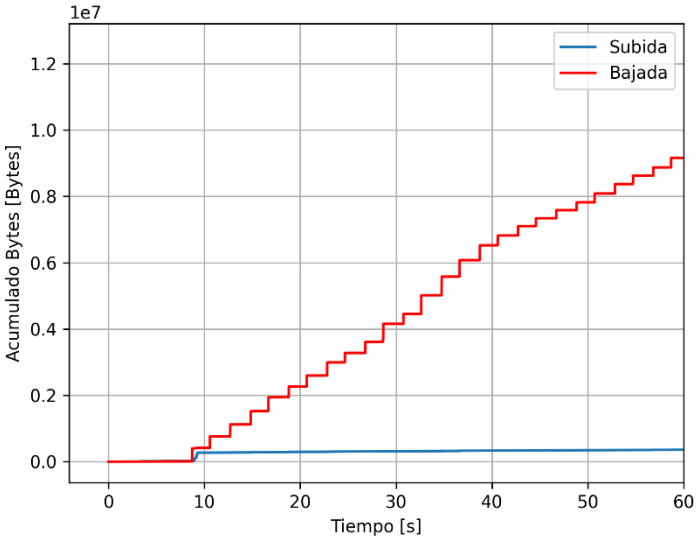


Figura 36. Representación del acumulado de bytes correspondiente a la muestra de tráfico de Prime Video.

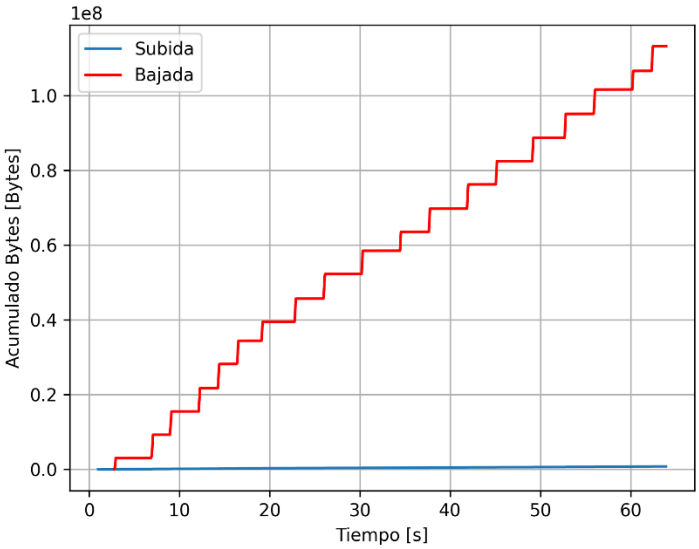


Figura 37. Representación del acumulado de bytes correspondiente a la muestra de tráfico de YouTube

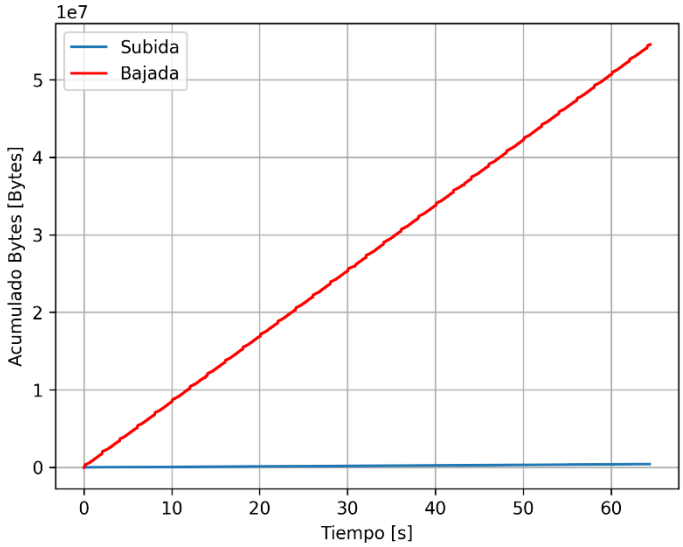


Figura 38. Representación del acumulado de bytes correspondiente a la muestra de tráfico de Twitch.

De esta forma se confirma la suposición anterior. El perfil de tráfico de Netflix [Fig. 1] no se caracteriza por un escaso volumen de paquetes, sino que tal y como se ha comprobado, el flujo se produce en ráfagas de baja periodicidad en las que se descargan una cantidad significativamente alta de datos. No se puede determinar por el momento si esto ofrece un resultado distinto a los demás servicios, de forma favorable a Netflix, respecto a la eficiencia energética, pero resulta llamativo caracterizar este perfil de tráfico.

De manera opuesta, se puede concluir caracterizando también los perfiles de los demás servicios. Prime Video y YouTube también demuestran esa forma de escalera en el acumulado de bytes, en el que la periodicidad de las ráfagas es mayor y el volumen de datos descargados en cada una es mucho más pequeño que el de Netflix.

Es curioso también observar que la muestra de YouTube no es solo la muestra con mayor número de paquetes sino la muestra con mayor volumen de datos, siendo que a la hora de realizarla se ha buscado reproducir un contenido idéntico en resolución (4K) Resulta, cuanto menos interesante, la relación entre las resoluciones de estas plataformas y el *throughput* calculado a raíz de la captura. Tanto en el caso de Netflix como Prime Video, se obtiene un volumen de datos, y un valor de throughput considerablemente menor al resto, siendo que la captura de YouTube también se realizó reproduciendo contenido en 4K, y lo más llamativo, siendo que la de Twitch era FullHD. De nuevo, la única justificación se encuentra en el número de paquetes de la captura (8000 y 11000 – Netflix/Prime Video vs 45000 y 98000 – Twitch/YouTube) .

Se confirma además, que a pesar de tener aspectos similares, los perfiles de tráfico de subida y tráfico de bajada de todos los servicios, generan un volumen de datos distinto, siendo significativamente menor el volumen del tráfico de subida. Finalmente, se confirma también que el perfil de tráfico característico de Twitch coincide con un perfil de descarga lineal y continua [Fig. 38].

5.3 – Resultados obtenidos del simulador de impacto energético.

En esta sección, se deciden comparar los tres valores que se devuelven en el simulador al ser ejecutado para cada captura de tráfico de los servicios de streaming multimedia en cuestión. Una vez caracterizado el perfil de flujo de paquetes en la sección anterior, resulta conveniente observar cómo el tráfico de cada servicio impactaría en el consumo energético de un dispositivo si este sucediera en el mismo. De ahí la utilidad del simulador. Como ya se ha desarrollado en el capítulo cuarto, en su sección correspondiente, el valor analítico que aporta el simulador viene determinado por los tres valores que devuelve al ser ejecutado. Siendo que en el simulador se decide configurar el mecanismo DRX con los siguientes parámetros:

Parámetro	**Valor [ms]**
Inactivity timer	200
On duration timer	100
Short DRX Cycle	256
Long DRX Cycle	512

Una vez configurado el mecanismo DRX, se ejecuta el simulador habilitando y deshabilitando dos variables para cada captura de tráfico. La primera variable es “pss” (*power saving signal*). En base a su valor de tipo booleano (*True / False*) habilita o deshabilita la oportunidad de que el simulador aplique el mecanismo DRX. Y la segunda, la lista de diferencia de tiempos de paquetes de tráfico de subida, que en base a estar habilitada o no, ejecuta el simulador teniendo en cuenta los eventos de tráfico de subida o no.

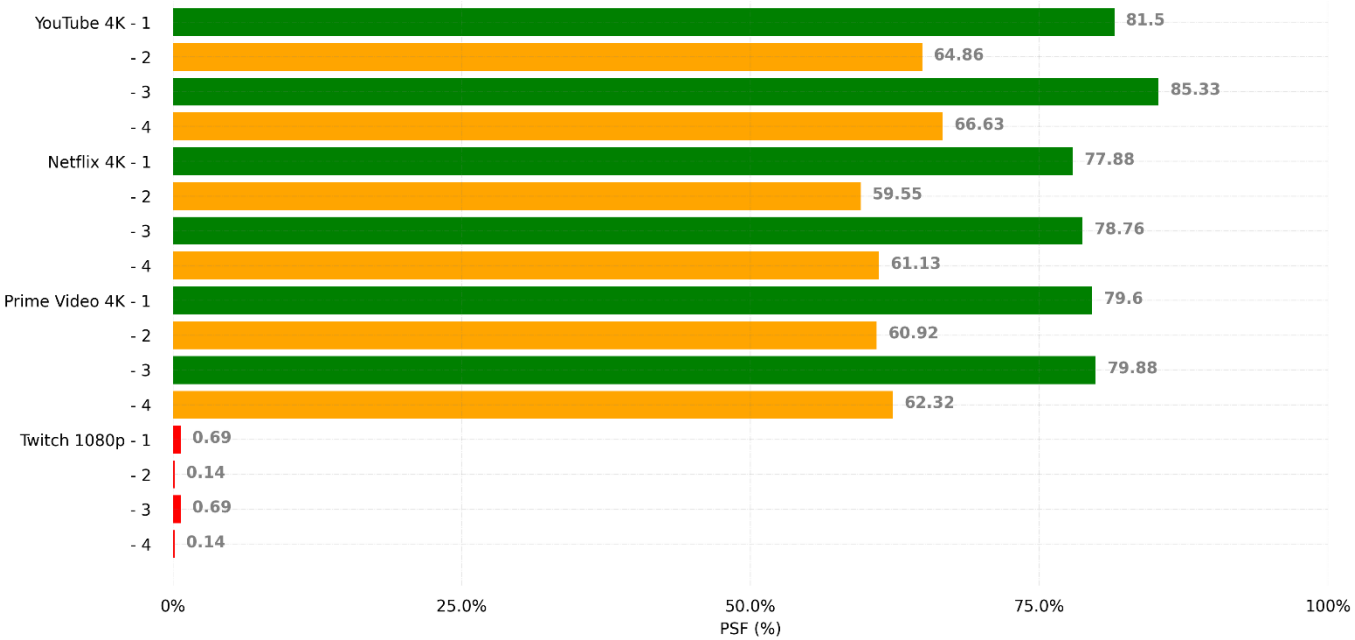


Figura 39. Comparativa de los valores de PSF obtenidos para cada servicio de streaming capturado

Color	Valor PSF
	>70%
	>40% &<70%
	<40%

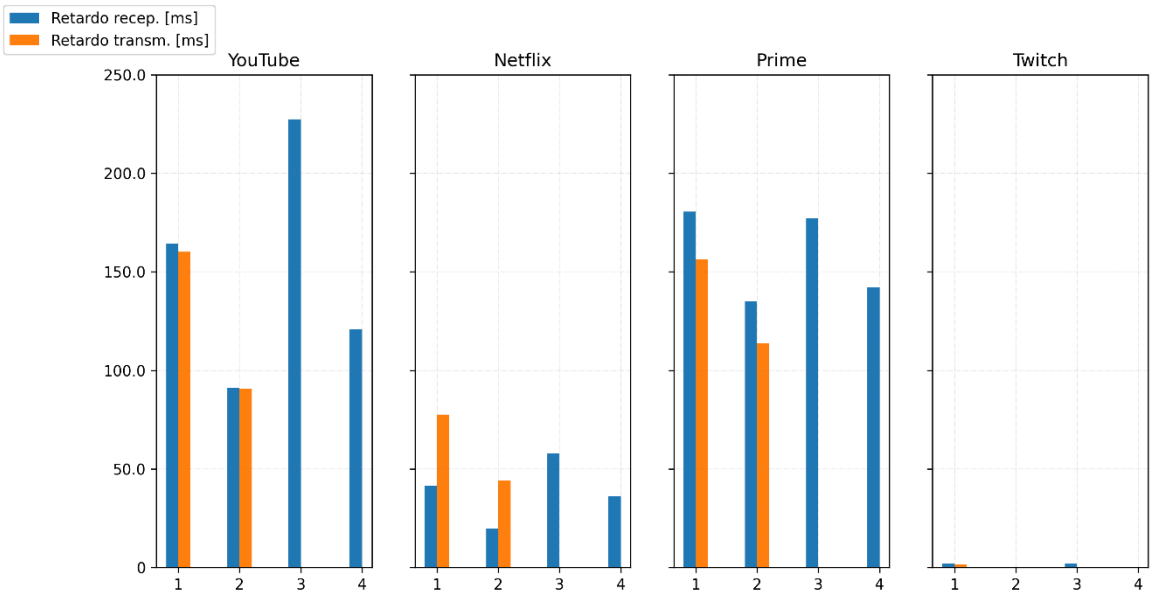


Figura 40. Comparativa de los valores de retardo obtenidos para cada servicio de streaming capturado.

	Pss	**Tráfico Subida**
1	True	Habilitado
2	False	Habilitado
3	True	Deshabilitado
4	False	Deshabilitado

En las dos figuras se han utilizado números para identificar cada resultado del simulador en base a los valores establecidos para "pss" y para el tráfico de subida, quedando resumidos en la tabla anterior.

Respecto a la comparativa de valores de PSF, se pueden extraer diversas conclusiones. En primer lugar, queda evidencia de la eficacia del mecanismo DRX como estrategia para el ahorro energético, debido a la notable diferencia existente entre los valores de PSF que devuelve el simulador cuando se está ejecutando con oportunidad de aplicar PSS (*pss=True*) y cuando no (*pss=False*).

Aunque es YouTube el servicio que aparentemente obtiene mejores resultados en cuanto a eficiencia energética, se observa que los demás también obtienen unos resultados muy similares, a excepción de Twitch, cuyo perfil de tráfico de descarga continua impide al terminal ahorrar energía, ya que siempre debe permanecer en escucha de nuevos paquetes de bajada.

Ahorrar más energía con DRX sin embargo también requiere sacrificar la comunicación inmediata entre el dispositivo y el servicio, produciéndose retardo. Es por esto que existe una relación directa entre el valor del PSF y los valores de retardo, en recepción y transmisión; a mayor valor de PSF (mayor ahorro energético), mayor retardo resultante, y viceversa. Por ello, es interesante también tener en cuenta el retardo de cara a ampliar el análisis de eficiencia de estos servicios de streaming en dispositivos. En este caso, siendo YouTube el servicio con mejores resultados en ahorros energéticos de forma aparente (valores de PSF más altos), es también el servicio con peores resultados en retardos en la comunicación, es decir, tiene los valores más altos también siendo que, cuanto más bajo sea el retardo mejor.

Por ello, se debe tener en cuenta un compromiso entre los valores de PSF y de retardo en este análisis de eficiencia energética. De nuevo, YouTube parece que obtiene los mejores resultados de PSF y Twitch los mejores resultados en retardos, sin embargo, sacrifican esta relación entre los valores de ahorro energético. A causa de obtener lo mejor de un valor, obtienen lo peor del otro. Por ello, en cuanto a eficiencia energética sería razonable destacar los resultados de Netflix. Sin tener en cuenta los resultados de Twitch, cuyo perfil de tráfico y tecnología streaming son totalmente diferentes, Netflix no obtiene los mejores valores de PSF, aunque no dejan de ser buenos, y muy similares al resto, pero obtiene valores de retardo notablemente menores que YouTube y Prime Video.

Por otro lado, respecto a los resultados obtenidos, es también interesante destacar la influencia del tráfico de subida. Como ya se mencionó en el capítulo segundo de la memoria, cuando se decide implementar DRX como mecanismo de ahorro energético en redes, muchas veces se opta por no considerar el tráfico de subida, ya que supuestamente reduciría la posibilidad de aprovechar DRX y ahorrar energía consecuentemente. Pues bien, se observa en este caso, que considerar también el tráfico de subida, no afecta tanto a los valores de PSF sino a los valores de retardo en cuanto a influencia en el ahorro energético se refiere. En la mayoría de casos se obtienen valores de retardo superiores cuando se omite el tráfico de subida en el simulador, sin embargo, también se consigue obtener valores de PSF ligeramente superiores que cuando no es omitido.

5.3.1 – Análisis de sensibilidad

Como complemento al análisis del impacto energético del tráfico capturado por los distintos servicios en dispositivos, se ha decidido también comparar para cada servicio, el comportamiento de dos parámetros de configuración del mecanismo DRX del simulador (*inactivity timer / on duration timer*) respecto al valor del PSF en este caso, para así descubrir la sensibilidad del simulador a la variación de estos parámetros. Gracias a la configuración automática del simulador, utilizada para establecer el valor de dichos parámetros (desarrollada en el capítulo cuarto), ha sido posible estudiar este comportamiento en base a ejecutar el simulador con todos los posibles valores de estos parámetros, descritos en el estándar 3GPP [11][12], para así obtener los resultados plasmados en las figuras siguientes:

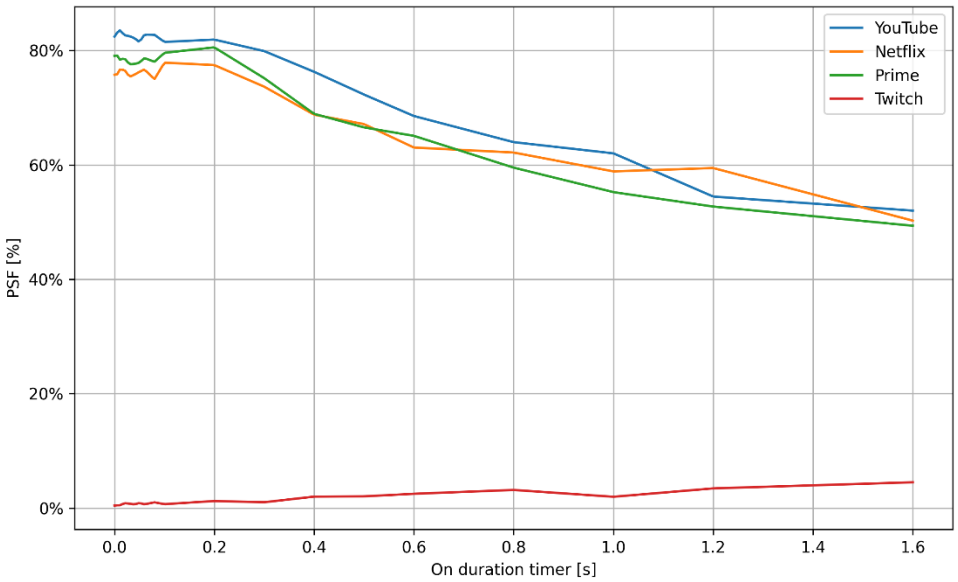


Figura 41. Comparativa entre servicios de la variación "PSF [%]" respecto a "on duration timer [s]"

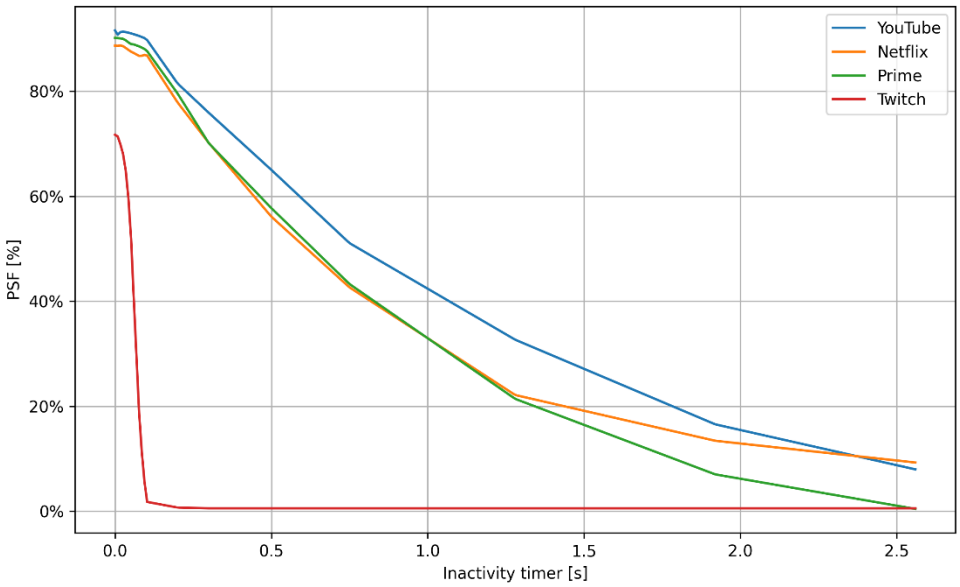


Figura 42. Comparativa entre servicios de la variación "PSF [%]" respecto a "inactivity timer [s]"

Estos resultados nos ayudan a comprender cómo se comporta cada parámetro y su influencia en la eficiencia energética de cada servicio, justificando la implementación de la configuración automática de parámetros DRX del simulador.

Tal y como se puede observar, se podría asumir que estos dos parámetros van de la mano, es decir, los resultados de PSF obtenidos tienden a decrecer cuanto mayor es el valor del parámetro, reflejándose de manera más acentuada en el caso de *inactivity timer*. Se demuestra así que un valor alto de estos parámetros no deja al simulador la oportunidad de aprovechar el mecanismo DRX para reducir el consumo de energía. No se ha considerado ni el comportamiento del resto de parámetros de configuración (*short/long DRX cycle*) ni su influencia en los retardos, también considerados en el análisis sobre el impacto energético, puesto que el simulador en este caso ha ofrecido resultados demasiado complejos de analizar.

Además, siendo que la configuración automática del simulador únicamente modifica un parámetro respecto a los demás, no es del todo razonable considerar estos resultados dentro del análisis sobre la eficiencia energética de cada servicio, ya que tres de los parámetros de configuración del DRX permanecen estáticos en un valor medio, y el que se configura automáticamente varía entre valores muy pequeños y muy grandes en comparación al resto, resultando en esta complejidad de resultados. No obstante, era interesante destacar el comportamiento detectado de estos parámetros.

5.3.2 – Comparativas y conclusiones finales

Finalmente, se decide terminar de obtener resultados con el simulador utilizando tres configuraciones de parámetros de entrada del DRX derivadas de este estudio realizado acerca de su comportamiento, con el objetivo de ofrecer conclusiones efectivas respecto al impacto del tráfico capturado por cada servicio de streaming multimedia. Son tres las configuraciones escogidas para poder abarcar el rango de valores posibles de todos los parámetros DRX con un valor pequeño, uno mediano y uno grande:

Nº Config / Valor [ms]	**1**	**2**	**3**
Inactivity timer	20	200	1280
On duration timer	10	100	600
Short DRX cycle	40	256	640
Long DRX cycle	80	512	1280

Existe una relación entre los valores de cada configuración para intentar asegurar unos resultados concluyentes para cada servicio. Esta relación es la siguiente; en la medida que el estándar lo permita, *inactivity timer* siempre es dos veces el valor de *on duration timer*, de la misma manera que *Long DRX Cycle* es siempre el doble que *Short DRX Cycle*. Mediante estas configuraciones se obtienen los siguientes resultados respecto a variaciones de PSF y retardos para cada servicio:

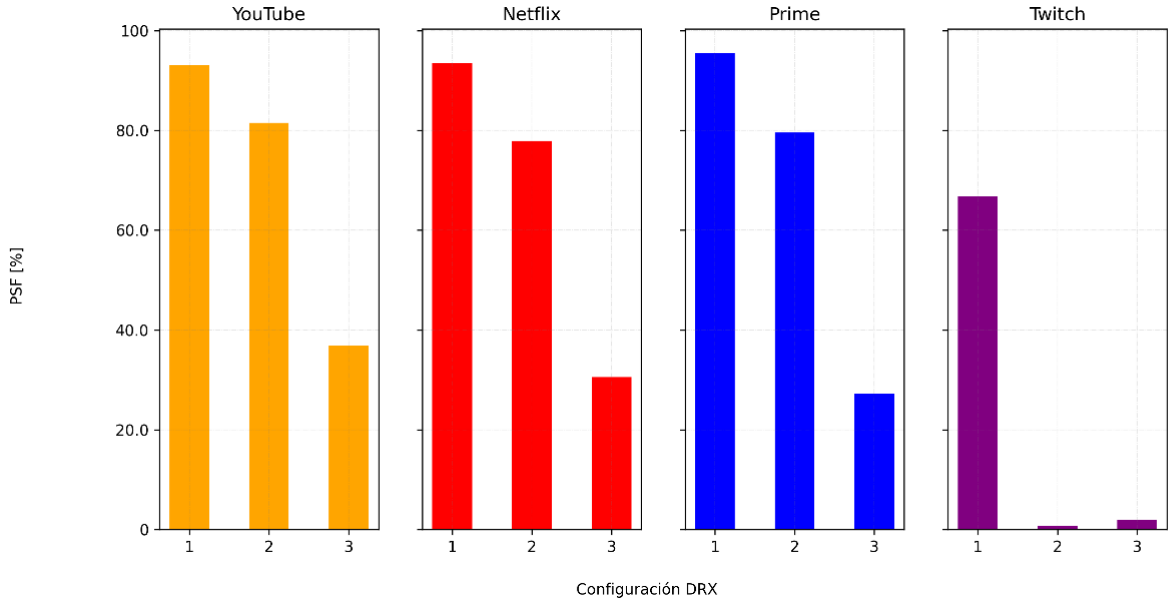


Figura 43. Comparativa entre servicios de la variación del PSF en cada configuración

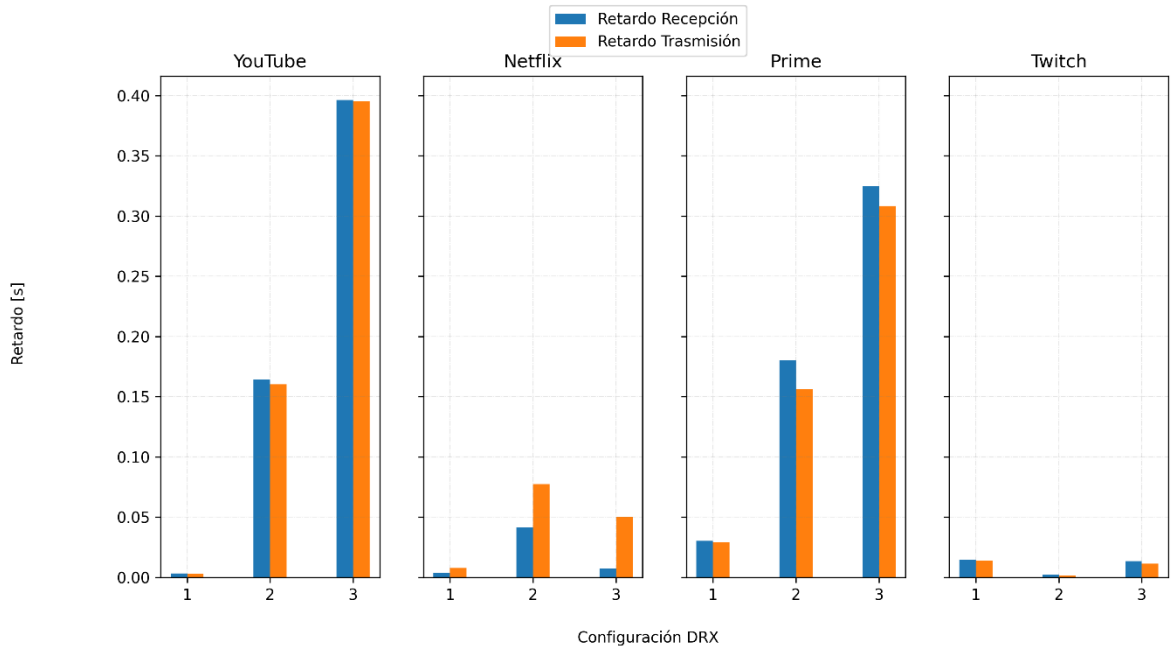


Figura 44. Comparativa entre servicios de la variación del retardo en cada configuración

Aunque estos resultados parezcan sencillos de interpretar, hay ciertos aspectos que hacen que sea más complejo de lo que parece. Esto es, hasta ahora, lo descubierto con los resultados indica que, si se incrementan los valores de *inactivity timer* y *on duration timer*, disminuye el valor del PSF y de los retardos, es decir, se ahorrará menos potencia con el mecanismo DRX. Sin embargo, aumentar los tiempos de DRX (*short / long*) implicaría justo lo contrario, aumentarían los tiempos de retardo y se ahorraría más energía (se darían valores de PSF más altos), pero lo observado en estas gráficas no representa precisamente esto.

En su lugar, se obtiene que aumentando el valor de los cuatro parámetros de forma simultánea, el valor del PSF disminuye, pero el de los retardos aumenta. Es posible por tanto que la relación existente entre los parámetros de configuración del DRX, sumada a las características del tráfico correspondiente de cada servicio, deriven en unos resultados contrarios a lo esperado del mecanismo DRX, y que por tanto son complejos de interpretar.

Debido a este hecho, se ha decidido finalizar el análisis de resultados con una última prueba en la que se cambia precisamente la relación existente entre los parámetros DRX. De esta manera, ahora el simulador se ejecuta con la variación de los parámetros pero esta vez de manera inversa. Mientras que con cada configuración *inactivity timer* y *on duration timer* crecen, los ciclos temporales de DRX decrecen. Así, se busca encontrar en las figuras resultantes una relación que no terminará de esclarecer estrictamente la relación existente entre los parámetros de configuración de DRX, pero que permitirá interpretar el resultado de la simulación de manera más sencilla, logrando extraer las conclusiones que nos interesan sobre la eficiencia energética de cada servicio de streaming:

Nº Config	**1**	**2**	**3**
Valor [ms]			
Inactivity timer	20	200	1280
On duration timer	10	100	600
Short DRX cycle	640	256	40
Long DRX cycle	1280	512	80

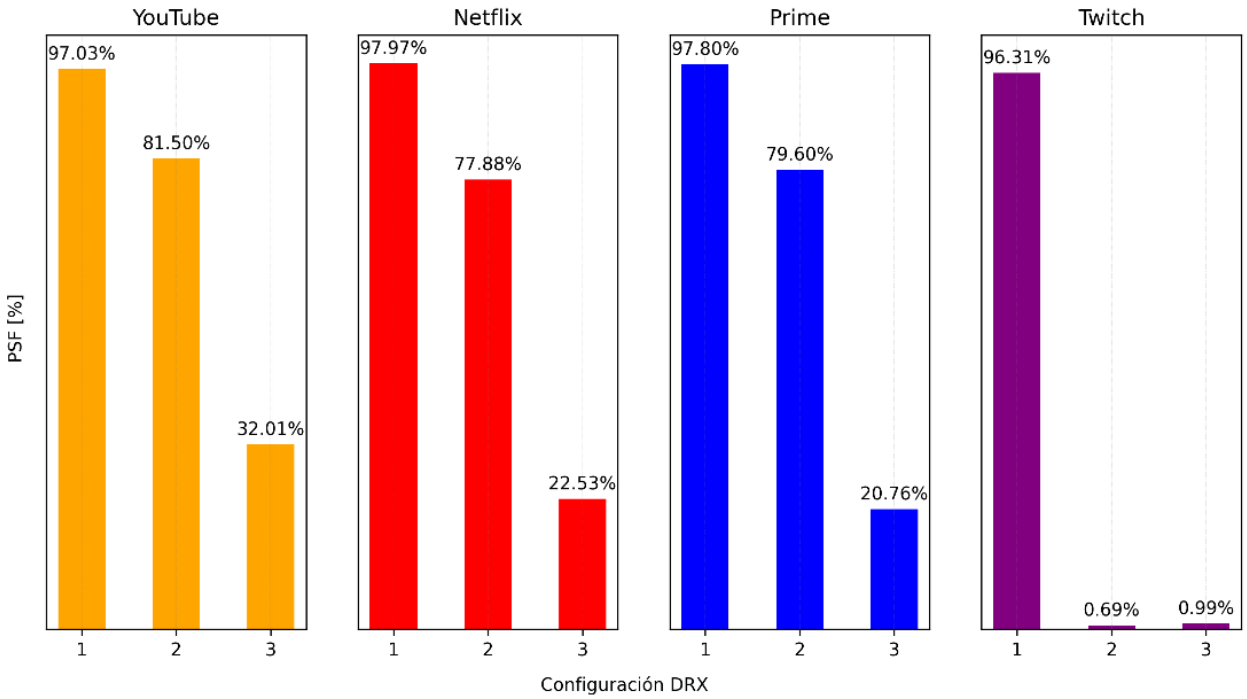


Figura 45. Comparativa definitiva entre servicios de la variación del PSF en cada configuración

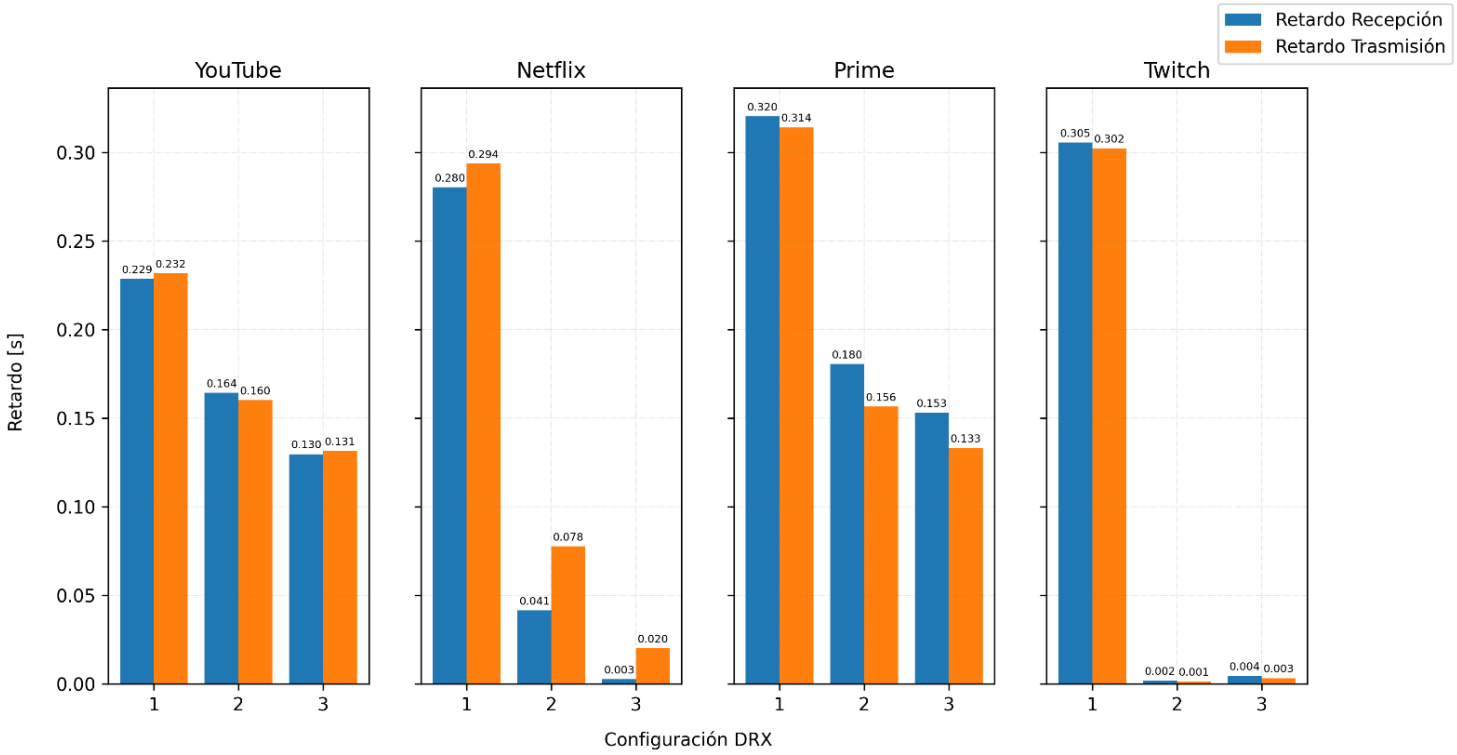


Figura 46. Comparativa definitiva entre servicios de la variación del retardo en cada configuración

Con esta relación entre parámetros definitiva, finalmente se consiguen resultados acordes con la teoría del funcionamiento del mecanismo DRX, además de lo estudiado y descubierto a lo largo del desarrollo de esta memoria. Al disminuir el valor del PSF, se ahorra menos energía, y por tanto existe un menor retardo en la comunicación.

Dentro de las conclusiones que podemos extraer sobre la eficiencia energética de los servicios de streaming de contenido multimedia capturados, encontramos de nuevo la similitud de YouTube, Netflix y Prime Video en sus comportamientos respecto a la variación de los valores resultados de PSF y retardos. Aún así, de nuevo, de entre estos tres resalta la eficiencia de Netflix en el momento de disminuir retardos a costa de reducir ahorro energético. Es cierto que esta situación brindada por la configuración número tres, es la menos deseada en cuanto a impacto energético, puesto que según los valores de PSF es la que implica un menor uso del mecanismo DRX. Sin embargo, las características del tráfico de Netflix parecen determinar una mejor gestión en la comunicación de cara a ahorrar energía, ya que siendo que obtiene un valor de PSF muy similar a YouTube y Prime Video, también obtiene unos valores de retardo ínfimos, notablemente inferiores en la descarga que en la subida de paquetes.

De manera evidente, e igual que anteriormente, no se han considerado los valores resultado de Twitch puesto que su tecnología de streaming (en directo) y su correspondiente perfil de tráfico no son comparables al de los otros tres. Sin embargo, y a modo de finalizar el análisis de resultados de este capítulo, es necesario destacar el hecho de que a pesar de esta tecnología de streaming en directo de Twitch, que obliga a una descarga continua de paquetes, es posible ahorrar energía en los dispositivos, siempre y cuando se utilicen valores muy pequeños para los parámetros de configuración del DRX. Aún así, esta posibilidad no garantiza la necesidad de ahorrar energía utilizando este mecanismo, ya que en el mismo caso en el que vemos un posible valor alto de PSF, existe también un valor alto de retardo (alrededor de 0,3 segundos) en la comunicación, que al tratarse de una comunicación en directo, resulta inválido.

Con estos resultados se da por finalizado el proceso de caracterización de los distintos modelos de consumo energético en UEs creados a partir de la reproducción de contenido multimedia en distintos servicios de streaming. De la misma manera, sirva todo lo expuesto aquí para concluir este proyecto, con la certeza de que los resultados obtenidos fruto de la motivación inicial permiten esclarecer con detalle tanto los distintos perfiles de tráfico que generan los principales servicios de streaming de contenido multimedia del panorama actual como una idea del consumo de recursos generado desde un punto de vista de eficiencia energética.

Bibliografía:

[1]: Global Internet Phenomena Report. Sandvine.com. Recuperado el 6 de septiembre de 2023, de https://www.sandvine.com/hubfs/Sandvine_Redesign_2019/Downloads/2023/reports/Sandvine%20GIPR%202023.pdf

[2]: Cisco Annual Internet Report (2018–2023) white paper. (2022, enero 23). Cisco. https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[3]: Mobile data traffic forecast – Mobility Report. (2020, November 25). Www.ericsson.com. https://www.ericsson.com/en/reports-and-papers/mobility-report/dataforecasts/mobile-traffic-forecast

[4]: Ortiz, J. (2017, septiembre 12). Formato de video - los más utilizados en video online. Javier Ortiz. https://javierortiz.mx/vod/formato-de-video-mas-utilizado/

[5]: Formatos y códecs para streaming. (2018, noviembre 28). Contenidos-digitales.es. https://contenidos-digitales.es/formatos-y-codecs-de-video-para-live-streaming/

[6]: ¿Por qué debes comenzar a usar el códec AV1? (2018, octubre 24). Contenidos-digitales.es. https://contenidos-digitales.es/por-que-debes-comenzar-a-usar-el-codec-av1/

[7]: HLS vs. MPEG-DASH – HTTP Video Streaming Protocols Compared (2021, mayo 31). Ottverse.com. https://ottverse.com/hls-vs-mpeg-dash-video-streaming/

[8]: Apple Inc. (n.d.). WWDC16. Apple.com. Retrieved September 6, 2023, from https://developer.apple.com/videos/wwdc2016

[9]: Understanding the HTTP live streaming architecture. (n.d.). Apple Developer Documentation. Retrieved September 6, 2023, from https://developer.apple.com/documentation/http-live-streaming/understanding-the-http-live-streaming-architecture

[10]: QUIC: los entresijos del protocolo experimental de Google. (2023, March 1). IONOS Digital Guide; IONOS. https://www.ionos.es/digitalguide/hosting/cuestiones-tecnicas/quic-protocolo-de-transporte-de-internet-basado-en-udp/

[11]: K. -H. Lin, H. -H. Liu, K. -H. Hu, A. Huang and H. -Y. Wei, "A Survey on DRX Mechanism: Device Power Saving From LTE and 5G New Radio to 6G Communication Systems," in IEEE Communications Surveys & Tutorials, vol. 25, no. 1, pp. 156-183, Firstquarter 2023, doi: 10.1109/COMST.2022.3217854.

[12]: L. Zhou, H. Xu, H. Tian, Y. Gao, L. Du and L. Chen, "Performance Analysis of Power Saving Mechanism with Adjustable DRX Cycles in 3GPP LTE," 2008 IEEE 68th Vehicular Technology Conference, Calgary, AB, Canada, 2008, pp. 1-5, doi: 10.1109/VETECF.2008.312.

[13]: Agile Alliance. (2001). Manifesto for Agile Software Development. Agilemanifesto.org. https://agilemanifesto.org/

[14]: Silverman, J. (2023, July 26). dpkt_doc: Documentation, sample inputs, and sample programs that use the dpkt library. jeffsilverm/dpkt_doc. GitHub. https://github.com/jeffsilverm/dpkt_doc

[15]: csv — Lectura y escritura de archivos CSV — documentación de Python - 3.8.17. (n.d.). Python.org. Retrieved September 6, 2023, from https://docs.python.org/es/3.8/library/csv.html

[16]: pandas.DataFrame — pandas 1.2.4 documentation. (n.d.). Pandas.pydata.org. https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html

[17]: matplotlib.pyplot — Matplotlib 3.5.3 documentation. (n.d.). Matplotlib.org. https://matplotlib.org/3.5.3/api/_as_gen/matplotlib.pyplot.html

[18]: matplotlib.pyplot.stem — Matplotlib 3.7.2 documentation. (n.d.). Matplotlib.org. Retrieved September 6, 2023, from https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.stem.html

[19]: matplotlib.pyplot.subplots — Matplotlib 3.7.1 documentation. (n.d.). Matplotlib.org. https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.subplots.html#matplotlib.pyplot.subplots

[20]: matplotlib.pyplot.hist — Matplotlib 3.5.1 documentation. (n.d.). Matplotlib.org. https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.hist.html

[21]: matplotlib.pyplot.bar — Matplotlib 3.4.3 documentation. (n.d.). Matplotlib.org. https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.bar.html

[22]: How to calculate and plot a Cumulative Distribution function with Matplotlib in Python ? (2021, January 22). GeeksforGeeks. https://www.geeksforgeeks.org/how-to-calculate-and-plot-a-cumulative-distribution-function-with-matplotlib-in-python/

[23]: numpy.sort — NumPy v1.23 Manual. (n.d.). Numpy.org. https://numpy.org/doc/stable/reference/generated/numpy.sort.html

Apéndice. Objetivos de Desarrollo Sostenible.

Grado de relación del proyecto con los Objetivos de Desarrollo Sostenible Agenda 2030 (ODS):

Objetivos de Desarrollo Sostenible	Alto	Medio	Bajo	No Procede
ODS 1. Fin de la pobreza.				X
ODS 2. Hambre cero.				X
ODS 3. Salud y bienestar.				X
ODS 4. Educación de calidad.				X
ODS 5. Igualdad de género.				X
ODS 6. Agua limpia y saneamiento.				X
ODS 7. Energía asequible y no contaminante.				X
ODS 8. Trabajo decente y crecimiento económico.				X
ODS 9. Industria, innovación e infraestructuras.				X
ODS 10. Reducción de las desigualdades.				X
ODS 11. Ciudades y comunidades sostenibles.				X
ODS 12. Producción y consumo responsables.		X		
ODS 13. Acción por el clima.				X
ODS 14. Vida submarina.				X
ODS 15. Vida de ecosistemas terrestres.				X
ODS 16. Paz, justicia e instituciones sólidas.				X
ODS 17. Alianzas para lograr objetivos.				X

Se considera un grado de relación media con el “ODS 12. Producción y consumo responsables”, puesto que una parte de los objetivos de este trabajo busca el entendimiento de la eficiencia energética de los distintos servicios de streaming multimedia estudiados. Mediante los resultados obtenidos de la utilización del mecanismo DRX como representaciones de ahorro energético, se observa cómo con el desarrollo incipiente que se está llevando a cabo en la tecnología de las nuevas redes inalámbricas 5G/6G, permite el consumo en masa de contenidos, con un todavía esperado auge, buscando la utilización de los mínimos recursos posibles, o lo que es lo mismo, hacer más con menos. Precisamente esto último es lo que resume este Objetivo de Desarrollo sostenible redactado por la Organización de las Naciones Unidas (ONU), de ahí que se haya relacionado de esta manera con el trabajo llevado a cabo en este proyecto.