Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Uso de Códigos Raptor para mejorar la robustez de flujos multimedia

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Álvaro Ibáñez Galiana

Tutor: José Salvador Oliver Gil

2018/2019

Resumen

Reproducir contenido multimedia años atrás conllevaba la descarga integra del archivo que se quería reproducir. Desde hace unos años se ha ido sustituyendo por la transmisión en vivo pudiendo acceder a su consumo en cualquier momento y desde cualquier lugar.

La popularidad de la transmisión de video en directo estos últimos años está en pleno crecimiento. Redes sociales, marcas, periodistas, figuras públicas e incluso equipos de futbol.

Cientos son las organizaciones y plataformas que utilizan el streaming y la transmisión en vivo para informar de alguna noticia, vender algún producto e incluso ver la sesión del Congreso de Diputados a través de Internet.

Este crecimiento desmesurado de la tecnología y de las plataformas también afecta al consumidor. Ante la alta demanda de este tipo de contenido, las empresas están teniendo que desarrollar e implementar métodos para que ningún posible consumidor sufra problemas a la hora de consumir este contenido. El consumidor espera poder acceder al mismo de una manera uniforme y poder acceder desde cualquier sitio, momento y dispositivo. Esto genera dificultades a la hora de garantizar una transmisión sin saltos ni problemas en caso de que, por ejemplo, se supere la capacidad del servidor durante la reproducción de un evento generando descontento en el consumidor y dejando una imagen negativa sobre la marca, persona o producto.

Todo lo comentado previamente desemboca en que una de las maneras más fáciles y económicas de mejorar la transmisión es directamente en el propio flujo multimedia, ya que no se ha de invertir tanto tiempo y dinero. Recursos que si que se tendrían que invertir en caso de querer modificar la propia infraestructura.

En este proyecto el principal objetivo es desarrollar técnicas de mejora para la transmisión de flujo multimedia mediante códigos Raptor de manera que en un escenario en el que se perdiesen paquetes, o hubiera algún tipo de problema a la hora de reproducir el contenido, el consumidor sufra esta pérdida de calidad de reproducción de la menor manera posible.

Se realizará un estudio comparativo de estas técnicas con otras más FEC más simples como bien puede ser los códigos LT o simplemente el uso de bits de paridad para valorar cual sería más beneficioso.

Palabras clave: Multimedia, transmisión, streaming, Raptor Codes, LT codes, FEC, flujos

Abstract

Years ago, playing media force you to download the whole file if you wanted to use or watch it. Since then, that method is being replaced by livestreaming video allowing users enjoy that media from any place and any time they want.

Video streaming popularity has increased lately and nowadays is growing a lot. Social networks, popular brands, journalists, public figures and even football teams.

Hundreds of organizations and platforms use streaming and livebroadcasting to report news, promote one of the products they're trying to sell or announce or simply to broadcast the Congress through the Internet.

This exaggerated growth of technology and all the platforms involved also affects consumers. Following the high demand of this kind of content, companies have been trying to develop and implement methods and ways so every client doesn't experience problems while watching their content. Clients want to enjoy the content they're watching without any problems and being able to play it in any place, moment and from any device they use. This implies some difficulties to guarantee a stable streaming and with zero problems in case that, for example, server capacity is full during the event. This will make clients angry and annoyed because the connection will have problems or even cuts and leaving a bad impression about the brand, person or product. And that's what companies want to avoid.

All these things made developers look for an economic and viable way to improve the livestreaming of content directly in to the media streams, because it's easier than improving the infraestucture or the consumer devices or connections, since that's a lot more expensive and a lot harder.

In this project, the main objective is developing techniques to improve the broadcast of media streaming through Raptor Codes in a way that makes

packet loss, or any other kind of problem when consuming the content, less problematic to the consumer so that problems won’t leave a bad impression on them or the inability of enjoying it.

We’ll do a comparison between these kind of techniques and other simpler FEC techniques like LT codes or parity bits so we can evaluate which one of these are easier to implement and more benefitial for the context we’re in.

Keywords : Multimedia, transmission, streaming, Raptor Codes, LT codes, FEC, streams.

Tabla de contenidos

1. Introducción

El consumo de flujos multimedia, ya sea video o audio ha estado creciendo exponencialmente desde su primera fase de desarrollo. Esta nueva forma de comunicación afectó y afecta al 100% de los usuarios de Internet, así como a los dispositivos que tenemos, protocolos de comunicación interacción entre usuarios y hasta las infraestructuras.

Hablar sobre la transmisión de flujos multimedia o livestreaming no es una materia breve. La gran variedad de métodos de implementación y formas de consumir el contenido provoca que el rango de precios varíe mucho tanto en hardware como en software.

Nosotros nos vamos a centrar en una variante gratuita modificando a nivel de aplicación el código para intentar mejorar la calidad lo máximo posible.

Comenzaremos situando el contexto del proyecto y el por qué es necesaria la utilización de tipos de técnicas para evitar todas estas problemáticas pasando a explicar las mismas.

La sección posterior se centrará en los detalles de la implementación y las tecnologías utilizadas para llevar a cabo el proyecto junto con los tecnicismos en cuanto a los códigos utilizados.

Para concluir, se procederá a incluir comparativas entre las técnicas mas populares junto a las conclusiones del proyecto e ideas para futuros usos en otros proyectos de este tipo de técnicas y, por último,

1.1 Motivación

Hoy en día, para bien o para mal, Internet está en todas partes.

El desarrollo de las tecnologías y la accesibilidad haciendo cada vez más asequible para todos estos tipos de plataformas ha convertido Internet en un foco para la mayoría de empresas y departamentos de marketing, provocando que la mayoría de estas centren sus esfuerzos y presupuestos en potenciar sus imágenes a través de este.

Es muy común ver publicidad en las redes sociales como anuncios entre los perfiles de Instagram o navegando en las páginas de Facebook o Twitter encontrando anuncios o clips de video promocionando productos o eventos.

En este tipo de contenido, las personas tenemos un umbral de paciencia bastante bajo.

Todo esto puede provocar que ante una mala experiencia utilizando cierta plataforma consideremos no volver a usarla si encontramos otra mejor. Este tipo de mejora que vamos a desarrollar permite que el factor más importante (la calidad de video) se vea afectado lo mínimo posible haciendo la visualización del contenido lo más cómoda posible.

El mundo de la multimedia, tanto video como audio, siempre ha sido una de mis principales aficiones, llevándome a tener trabajos parciales mientras estudiaba editando videos e imágenes para algunas compañías pequeñas de Valencia y “quitándome” una gran parte de mi tiempo libre pasando horas investigando y probando software y hardware para intentar mejorar. Todo esto también se vió potenciado al tener una familia que ha estudiado y/o trabaja en el mundo de la tecnología y la informática permitiéndome acceder a herramientas en casa que de otra manera no tendría o no habría mostrado curiosidad por ellas hasta el punto de debatirme entre cursar Ingeniería Informática o Ingeniería Multimedia hasta el último momento.

Con este trabajo, en cierta medida, intento aprovechar la última parte de la carrera para hacer algún trabajo que mezcle dos de mis mayores aficiones o intereses.

1.2 Objetivo general

El objetivo general del proyecto es emplear técnicas de corrección de errores hacia adelante (Forward Error Correction) para mejorar la robustez de un flujo de video.

Implementaremos la solución mediante una librería de código abierto para los códigos Raptor incorporándolo en un marco de transmisión.

1.3 Objetivos

- El objetivo principal de este trabajo es robustecer la transmisión de flujos multimedia mediante técnicas FEC (RaptorQ) estandarizado por la IETF bajo el RFC 6330. Se realizará:

    1. Explicar en qué consisten las técnicas de robustecimiento que se estudian y se van a implementar. En el proyecto se ha decidido por utilizar la librería de código abierto OpenRQ encargada de codificar y decodificar.
    2. Implementar estas técnicas con códigos raptor e incorporarlas a un escenario con pérdida de paquetes. Como hemos definido en el punto anterior se utilizará Java, que es el lenguaje que utiliza la librería que hemos mencionado y la podemos adaptar a cualquier sistema.
    3. Conclusiones a raíz del desarrollo del código y estudio de los beneficios del mismo durante la transmisión de un flujo multimedia

1.4 Estructura de la memoria

Esta memoria está constituida por siete capítulos.

- Introducción: Se presenta el proyecto a desarrollar, los objetivos principales y la motivación para escoger y desarrollar dicho proyecto. Sirve como preámbulo para el resto de capítulos que constituye el proyecto.
- Estado del arte: Estudiaremos el estado actual del mercado y de las tecnologías que se utilizan en este tipo de plataformas y sus características para la comparativa y ver que aplicaciones puede tener nuestras técnicas en el mundo actual.
- Códigos Raptor: Analizamos las características principales de nuestra tecnología, la librería de código abierto que vamos a utilizar y algunas de las peculiaridades, tanto de la tecnología como de la librería para ver sus ventajas y sus desventajas.
- Tecnologías utilizadas: Este capítulo está enfocado a describir que herramientas hemos utilizado para desarrollar nuestro proyecto y que usos les hemos dado.
- Implementación: Apartado con descripciones y capturas del código donde vemos paso a paso todo el desarrollo y los resultados.
- Conclusiones: El último apartado de la memoria donde se cierra el proyecto con las conclusiones sacadas a partir del desarrollo del mismo, posibles trabajos futuros y la bibliografía que se ha utilizado.

1.1 1.5 RFCC 6330

El estándar sobre el cual se basa este trabajo es el RFCC 6330.

El documento describe en su integridad el esquema de Corrección de errores hacia Adelante para el código FEC RaptorQ y su aplicación a la entrega asegurada de objetos de datos.

Describe los nuevos códigos Raptor como códigos que otorgan flexibilidad superior, soporte para tamaños de bloque de origen más grandes y mejor eficiencia de código que los códigos raptor en el RFC 5053. RaptorQ también es un Fountain Code de manera que se pueden generar tantos símbolos codificadores al momento a partir de los símbolos fuentes de un bloque de datos.

El decodificador es capaz de recuperar los bloques fuente de casi cualquier conjunto de símbolos codificadores de suficiente cardinalidad. En la mayoría de casos, con un número equivalente de símbolos fuente valdría, salvo en casos especiales donde se necesita un numero ligeramente superior.

Además de este pequeño resumen, el resto de aspectos estandarizados se resumen en ese documento [1] donde se representa el consenso de la IETF (Internet Engineering Task Force donde se ha aprobado por revisión publica y aprobado para la publicación por el IESG (Internet Engineering Steering Group)

2. Materia relacionada

2.1 Manejo de la bibliografía

Se han utilizado las siguientes referencias bibliográficas para su consulta:

- La primera referencia [1] https://openrq-team.github.io/openrq/ es el centro del proyecto donde aparece la página principal del OpenRQ con una pequeña explicación de los Raptor Codes donde hemos cogido referencias para la explicación y hemos basado el desarrollo. También hemos encontrado enlaces a otras referencias.

- La segunda referencia [2] https://tools.ietf.org/html/rfc6330 la hemos utilizado para estudiar el estándar sobre el que está basado el proyecto y entender mejor cómo funciona el trasfondo de la librería de código abierto.
- La tercera referencia [3] nos ha servido a lo largo de la carrera y para el proyecto para entender diferentes conceptos del networking y cerrar algunos conceptos previos a la elaboración del mismo.
- La cuarta referencia [4] nos hemos apoyado para centrar los conceptos de los protocolos de transmisión.
- La quinta referencia [5] es principalmente uno de los pilares sobre los que se apoyan los raptor codes. La obra de este autor y parte de sus códigos fueron comprados después por Qualcomm que los tiene en propiedad a día de hoy.
- La sexta referencia [6] es la obra de los autores de OpenRQ que hemos mirado para entender los códigos raptor. Se ciñe a explicar la robustez de los códigos raptor en entornos donde la red puede presentar problemas de conectividad.
- La séptima referencia [7] es la adaptación a C++ de OpenRQ que hemos empleado para entender ciertas partes del código que en Java no hemos visto excesivamente claras.
- La octava referencia [8] hemos buscado referencias a los Fountain Code que hemos incluido en el apartado teórico explicativo de los códigos raptor.
- La novena referencia [9] la hemos utilizado como el repositorio de información que es, buscando conceptos a lo largo de todo el proyecto contrastándolos con la documentación oficial para sintetizar la información.
- La décima referencia [10] nos ha servido para entender la ingeniería del backend de Twitch, una de las plataformas de livestreaming mas importantes, si no la más, del mundo y como está montada para situarnos en el contexto actual de la tecnología.
- La undécima referencia [11] hemos encontrado un gran número de estadísticas para respaldar el contexto actual de la tecnología del streaming por internet.
- La duodécima referencia [12] nos ha aportado definiciones e información resumida de diversos términos tecnológicos.
- La decimotercera referencia [13] nos hemos apoyado para contrastar los datos sobre el tráfico de video por internet en los últimos años.
- La decimocuarta referencia [14] explica cómo funciona el backend de Netflix y algunos de los diagramas de su funcionamiento.
- En la decimoquinta referencia [15] hemos usado la documentación para consultar algunos comandos de UNIX durante nuestras pruebas.

2.2 Acrónimos y términos utilizados

- **RFC :** Request For Comments. Es un documento formal montado por la IETF que describe las especificaciones de una tecnología en particular. Cuando es ratificada por una organización o una licencia publica pasa a ser un documento formal. Su primer uso fue con los protocolos ARPANET de lo cual apareció el internet actual.
- **IETF :** Internet Engineering Task Force. Es el principal organismo que define ciertos estándares para protocolos operativos de Internet como puede ser el de nuestro proyecto, o bien otros más establecidos como TCP/IP.
- **FEC :** Forward Error Correction. Es un método de obtener control sobre los errores en transmisión de manera que el emisor envía información redundante y el destinatario solo reconoce aquellos segmentos de datos que no contienen errores.
- **Playthrough:** Termino que se utiliza normalmente para hacer referencia a la retransmisión en video de un jugador completando un videojuego en su totalidad.
- **CDN:** Red de Distribucion de Contenidos. Es un tipo de infraestructura informática que se organiza entrelazando varios ordenadores repartidos geográficamente en diversos datacenters. Almacenan parte de la información para que el usuario final disfrute más cómodamente del contenido.
- **POP:** Point of Presence. Es un lugar físico donde un proveedor de servicios tiene montado hardware para replicar la transmisión.
- **VOD:** Video on Demand: Video bajo demanda. Es una modalidad de difusión multimedia que permite al usuario reproducir el contenido que quiera en cualquier momento reproduciéndolo online en su dispositivo.
- **EC2:** Elastic Compute Cloud: Es un servicio web que proporciona capacidad informática en la nube segura y de tamaño modificable. Diseñado principalmente para facilitar a los desarrolladores el escalado de la informática en la nube.
- **Netcat:** Netcat es una herramienta de red que permite a través de la línea de comandos abrir puertos en un HOST quedando netcat a la escucha asociando una Shell a un puerto en concreto.
- **Parsear:** Proceso de analizar una secuencia de símbolos a fin de determinar su estructura gramatical definida.

3. Estado del arte

Debido justamente a la gran popularidad que tiene actualmente el consumo de multimedia hoy en día se han desarrollado multitud de plataformas y un gran número de horas por parte de los profesionales para intentar exprimirlo al máximo.

Tácticas como la optimización de los motores de búsqueda o el marketing de contenido consumen muchos recursos tanto de tiempo como económicos.

Utilizar plataformas de livestreaming es prácticamente gratuito, sobre todo para empresas que ya tienen la infraestructura para retransmitir en otros medios de comunicación ya que prácticamente no tienen que hacer ningún cambio en sus medios para aprovecharlo.

3.1 Situación actual

Vamos a presentar algunas de las estadísticas más importantes respecto a este sector para situarnos y entender por qué mejorar la calidad de la transmisión multimedia puede ser vital para nuestra marca o empresa y mejorar nuestros resultados exponencialmente.

Desde 2016, la industria entró en una espiral de crecimiento.

De 2015 a 2016 se duplicó la cantidad de contenido reproducido a través de Internet. Partiendo ya de una industria de 30 billones de dólares se espera que en 2021 supere los 80 billones de dólares.

En 2019, representa más del 80% de trafico de uso de internet doméstico.

Consumer Internet Video 2014–2019							
	2014	2015	2016	2017	2018	2019	CAGR 2014–2019
By Network (PB per Month)							
Fixed	20,485	25,452	32,981	43,226	56,771	74,319	29%
Mobile	1,139	2,014	3,475	5,842	9,407	14,999	67%
By Category (PB per Month)							
Video	18,437	22,940	30,242	40,907	55,931	76,771	33%
Internet video to TV	3,188	4,526	6,214	8,160	10,248	12,548	32%

Ilustración 1. *Streaming de video y por descarga crecerá hasta más del 80% de todo el tráfico domestico de internet en 2019 (datos vía Cisco)*

Este aumento en el share también produce que los consumidores (o los usuarios, dependiendo del tipo de contenido) busquen antes un producto a través de un video promocional o informativo antes que un texto, review o imágenes.

De la misma manera, según BrightCove, alrededor de 2 de cada 3 usuarios se llevan una imagen negativa de un producto o persona que utilice o publique un video retransmitido con problemas o con mala calidad.

Aquí es donde entra nuestro proyecto, y el de muchas otras empresas que intentan lo mismo para evitar la pérdida de estos posibles futuros clientes.

Situemos este contexto desde un punto de vista empresarial comparando algunas de las empresas cuyo auge ha ido a la par de la tecnología de publicación de video en internet.

3.2 Contexto empresarial

Comenzando por una de las plataformas más recientes y con mayor crecimiento y expansión desde su lanzamiento

3.2.1 Twitch.tv

Ilustración 2. Logo de Twitch.tv

Twitch.tv es una plataforma que ofrece un servicio de streaming de video en vivo en propiedad de Amazon.com. El objetivo original de la plataforma era que el público pudiese ver sus videojuegos favoritos en directo siendo un apartado de otra plataforma (Justin.tv). El contenido original consistía en “playthroughs” de juegos jugados por los propios usuarios de la página, eventos de eSports, como los últimos torneos mundiales del videojuego League of Legends que tuvo lugar en Corea del Sur y tuvo un pico de 2 millones de usuarios simultáneos.

A día de hoy la plataforma tiene categorías de todo tipo donde usuarios retransmiten su contenido desde como cocinar una tarta hasta ellos mismos subiendo a la Torre Eiffel. Todo esto retransmitido en tiempo real.

Twitch.tv reunió el pasado Julio a medio millón de personas para ver el tráiler de lanzamiento de un videojuego en primicia durante la conferencia anual de videojuegos más famosa, la E3.

Todas estas retransmisiones simultaneas fuerzan a Twitch a centrar la mayor parte de su ingeniería a mejorar el mismo aspecto en el que nos estamos centrando. Los flujos multimedia.

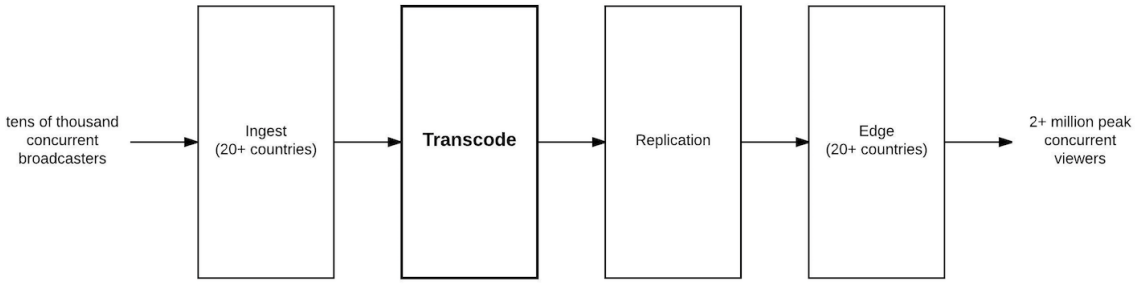


Ilustración 3. Arquitectura del CDN de Twitch.tv

Twitch utiliza el protocolo RTMP (Real-Time Message Protocol). Es un protocolo diseñado principalmente para la transmisión de video y audio en tiempo real y utilizado principalmente para la transmisión punto a punto. Para el escalado a un numero incontable de espectadores se utilizan protocolos HTTP Live Streaming (HLS) y comunicaciones basadas en HTTP que utilizan la mayoría de sitios web.

En el procesado del contenido por el codificador se encarga de convertir el RTMP entrante en el formato HLS con multiples perfiles. Estos perfiles tienen diferentes bitrates de manera que todo tipo de espectadores pueden disfrutar del contenido en calidades diferentes.

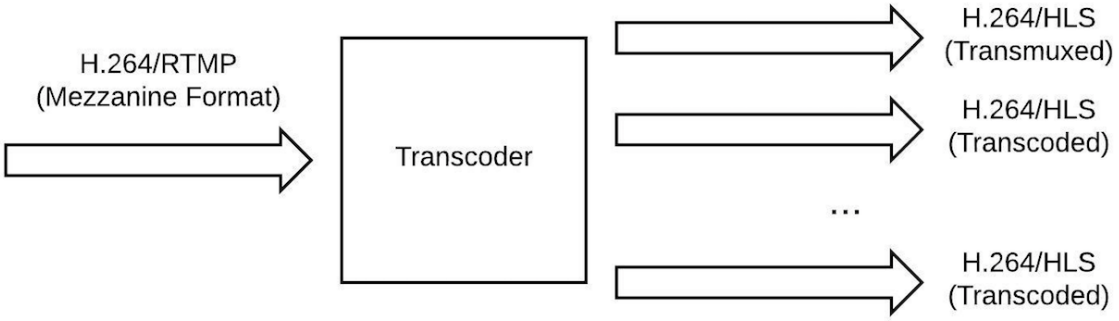


Ilustración 4. Input y Output del Codificador

A grosso modo funciona de la siguiente manera:

2. Input de video: Recibe el video por RTMP y lo pasa el sistema de decodificación.
3. Codificación/Decodificación: Coge el flujo RTMP del retransmisor y se transforma en varios flujos HLS. Esto se implementa mediante una combinación de C/C++ y Go. En este apartado nosotros usaríamos Raptor Code para evitar la pérdida de calidad entre emisor y receptor.
4. Distribución: Distribuye los flujos HLS a los POP distribuidos geográficamente para tener la mejor experiencia como espectador. También la mayor parte del código es en Go.
5. VOD: Se almacena todos los flujos entrantes de video en el sistema de Twitch de almacenamiento de VODs.

3.2.2 Netflix

Ilustración 5. Logo de Netflix Inc.

Netflix es una empresa de entretenimiento estadounidense cuyo servicio principal es distribuir contenidos audiovisuales siguiendo el sistema de Video On Demand por streaming. En sus orígenes comenzó únicamente como una empresa para alquilar DVDs mediante suscripción y pasó al sistema actual a partir de 2007 expandiendose a casi todo el mundo llegado 2016.

Al tener un público tan amplio repartido por todo el mundo y accediendo de tantos dispositivos diferentes.

Netflix recibe fuentes de video de la más alta calidad de shows de televisión y películas y los codifica de manera que los espectadores disfruten de los mismos en la mejor calidad posible en función del dispositivo y del bando de ancha disponible. Debido al crecimiento desmesurado de la plataforma la función principal de los ingenieros de Netflix ha sido la misma que los del resto de plataformas, centrarse en los flujos multimedia.

El sistema de Netflix está diseñado de manera que sea escalable y fácilmente adaptable a la hora de modificarlo para llegar a todos los usuarios del mundo.

Codificacion:

El diagrama de codificación de Netflix funciona en sistemas cloud EC2 de Linux. La elasticidad de la nube permite expandirlo cuando los títulos a procesar aumentan. Este tipo de codificacin busca lo mismo que nuestro proyecto con Codigos Raptor en Java: la adaptabilidad sin tener que hacer modificaciones en el hardware, ya que no requieren del mismo y pueden correr tantas instancias como sean necesarias.

Una vez codificados son entregados a un CDN para el streaming.

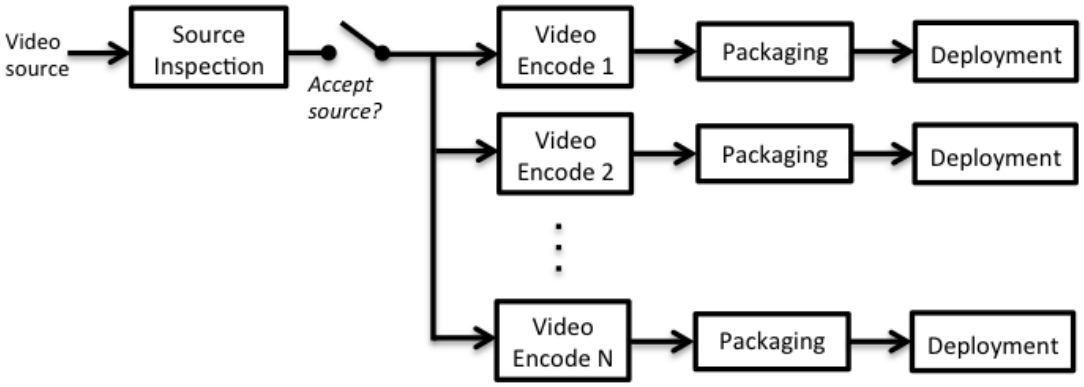


Ilustración 6. Inspeccion del alto nivel del sistema de Netflix

Durante una sesión de streaming, el cliente pide los codecs que puede reproducir y adaptativamente va cambiando entre ellos según las posibilidades que otorgan las condiciones de la red.

En los formatos modernos que recibe Netflix, como puede ser 4k, necesita del correcto fragmentado para poder ser retransmitido eficientemente. Los segmentos del fichero son procesados individualmente en paralelo por diferentes instancias EC2. Para cada segmento se analizan a nivel de flujo de bits y de pixel para detectar errores y generar metadata y una vez analizados se vuelcan sobre el fichero 4K.

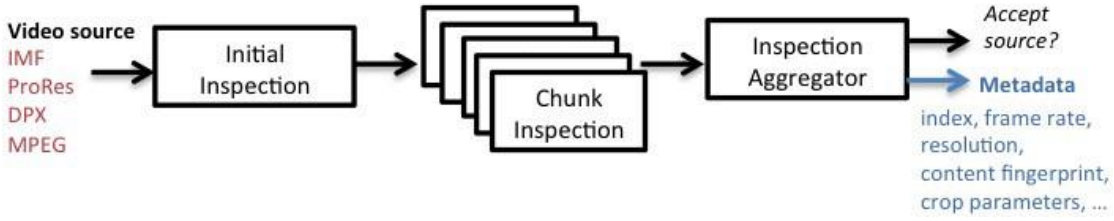


Ilustración 7. Flow detallado de la inspección de los segmentos

Todo este proceso junto con el poder computacional de Netflix es parte del proceso de inversión de la compañía para maximizar las medidas de calidad de los videos y mejorar la resistencia a errores del sistema.

3.3.3 Qualcomm

Ilustración 8. Logo Qualcomm

Por ultimo vamos a centrarnos en Qualcomm.

Qualcomm es una multinacional americana de semiconductores y de equipamiento de telecomunicaciones que diseña y comercializa productos para comunicaciones inalámbricas y servicios. Qualcomm tiene varias subsidiarias encargadas cada una de ellas de diferentes ramas de la empresa.

Una de las ramas de Qualcomm se centra en el apartado de Software y a nivel de aplicación.

Qualcomm adquirió la empresa de Fremont, Digital Fountain en 2009, y desarrolló la última generación de códigos Raptor llamada RaptorQ (sobre la cual basamos nuestro proyecto).

Tras esta adquisición, uno de sus productos pasó a ser la propia codificación mediante este tipo de códigos que sigue manteniendo a dia de hoy.

Ilustración 9. Imagen publicitaria RaptorQ de Qualcomm

4. Códigos Raptor

A continuación, vamos a analizar los Códigos Raptor.

Los códigos FEC son una técnica para recuperar errores en entornos poco favorables para la comunicación o canales con mucho ruido.

Se dividen principalmente en dos grandes grupos. Estos grupos se dividen según el tipo de bloques que transmitan.

- Tamaño fijo de bloque: Codificados y divididos antes de enviarlo. De cada bloque se incluye un fragmento de recuperación para que el cliente sea capaz de recuperar la información original en caso de fallo.
- Tamaño de bloque variable: A diferencia del anterior, en este tipo no se divide el flujo de bits en bloques. El algoritmo de Viterbi es uno de los más clásicos a la hora de decodificar eliminando en cada iteración del sistema aquel camino con menor parentesco.

El centro de este proyecto son los Códigos Raptor. Estos códigos son de tipo FEC (Forward Error Correction o corrección de errores hacia delante) tiene como característica principal permitir enviar paquetes (en este caso con contenido multimedia) siendo tolerante a fallos, pudiendo reestructurar el contenido enviado a pesar de haber perdido información. Son utilizados en sistemas de transmisión en tiempo real, que es el principal enfoque de este trabajo.

La idea principal es la corrección de errores codificando el mensaje original incluyendo bits de redundancia. Estos bits de redundancia permitirán que el receptor pueda corregir errores sin necesidad de un canal inverso para solicitar la información de nuevo. Como desventaja se observa un incremento en el ancho de banda utilizado debido al envío de datos adicionales sobre el mensaje original.

Antes de centrarnos en los Raptor Codes, vamos a introducir algunos términos:

4.1 Erasure Codes

Los Erasure Codes son un tipo de códigos FEC capaces de recuperarse de pérdidas en las comunicaciones. Los datos son divididos en K símbolos,

transformados posteriormente en un numero de símbolos N más largo de manera que una vez recibido el mensaje se puede recuperar de los errores o perdidas sobre este grupo de símbolos. Este tipo de códigos pertenecen al primer grupo de códigos FEC que hemos comentado previamente heredando las ventajas y desventajas sobre la recuperación de fallos y el ancho de banda.

4.2 Fountain Codes

Los Fountain Codes son un tipo de Erasure Codes con dos propiedades principales:

- Se puede producir un numero arbitrario de símbolos sobre la marcha simplificando la adaptación a diferentes ratios de pérdidas.
- La información se puede reconstruir con alta probabilidad de éxito de casi cualquier conjunto de símbolos de codificación.

Un caso típico de este tipo de códigos se da lugar cuando se intenta hacer una sola fuente hace un multicast a muchos destinos. En este escenario no podríamos usar canales TCP ya que el emisor necesita hacer un seguimiento de todos los canales para saber si el receptor ha recibido todos los paquetes. Por otra parte, utilizando UDP eliminaríamos esta restricción, pero no tendríamos la fiabilidad de TCP. Utilizando UDP y Fountain Codes se permite a cada receptor recuperar la información perdida en su canal, evitando pedir de nuevo al emisor los paquetes perdidos por un canal inverso.

4.3 Raptor Codes

Llegados a este punto aparecen varios inconvenientes. El principal problema detrás de esta tecnología es la incapacidad de conocer cuántos paquetes se perderán y cuantos paquetes llegan al cliente correctamente. Si pudiéramos estimar este número podríamos saber cuántos paquetes enviar al cliente y cuantos paquetes con información redundante tenemos que enviar.

La otra gran problemática de esta tecnología es el coste computacional asociado a la creación de símbolos de recuperación y también al propio coste computacional de la recuperación.

Estos dos problemas llevaron a la investigación de una versión diferente de FEC, entre las que destacan los Raptor Codes.

Los Raptor Codes son una solución más eficiente a los códigos LT. La principal mejora respecto al resto de códigos es un tiempo de codificación y decodificación lineal respecto al tamaño del mensaje con un sobrecoste cercano a cero.

Partimos del fragmentado de la información en bloques que codificaremos en el origen y decodificaremos en el destino.

Cada bloque se divide en símbolos para aumentar la flexibilidad y la optimización. Los llamaremos símbolos fuente. Estos símbolos junto con los símbolos de recuperación, que son del mismo tamaño que los símbolos fuentes, irán juntos en la transmisión formando los símbolos de codificación.

El cliente recibirá un subconjunto de estos símbolos suficientemente grande como para poder reconstruir la información. La probabilidad de recuperación es la probabilidad de que el símbolo fuente sea completamente recuperado al recibir un numero aleatorio de símbolos codificados del bloque fuente.

Con la información particionada en k símbolos, el decodificador RaptorQ puede según el código Raptor especificado en IETF RFC 6330 tener la siguiente relación entre símbolos codificados y probabilidad en la recuperación:

- Con un conjunto k de símbolos codificados, éxito en la decodificación del 99%.
- Con un conjunto k+1 de símbolos codificados, éxito en la decodificación del 99,99%
- Con un junto k+2 de símbolos codificados, éxito en la decodificación del 99,9999% (una posibilidad entre 1 millón)

4.4 RaptorQ

Una vez explicado el objetivo principal detrás del desarrollo de los códigos raptor vamos a centrarnos en los conceptos más importantes de este tipo de códigos y cada uno de sus parámetros y características.

Datos fuente

Es el término general que utilizamos para nombra a los datos que se van a codificar/decodificar. Puede ser un **archivo**, una **secuencia de bits** o un **stream de bits**.

Bloques fuente

Los datos fuentes son segmentados en bloques, llamados **bloques fuente**, para permitir la codificación eficiente en base a la memoria disponible para trabajar.

Cada bloque tiene un par codificador/decodificador independiente y se pueden procesar en paralelo. En función de la memoria disponible se pueden adaptar planes de trabajo diferentes. Por ejemplo, si el receptor tiene memoria limitada puede indicar un tamaño de bloque más pequeño de manera que el emisor se ve forzado a segmentarlo en bloques más pequeños. Una vez el receptor tiene todos los bloques los reestructura para poder procesarlos.

Simbolos

Un símbolo es la unidad más básica de datos utilizada en RaptorQ.

Se categorizan en:

- **Símbolo fuente**: Si es una porción contigua de un bloque de datos.
- **Símbolo reparador**: Si es una porción generada de manera redundante para asegurar la recuperación ante errores con una alta probabilidad.

Tanto los símbolos fuente como los símbolos reparadores forman parte de un conjunto llamado **símbolos codificadores**.

Un paquete codificador contiene uno o más símbolos codificadores e información para que el decodificador para identificarlos individualmente. Una vez lo símbolos son recibidos e identificados, si son suficientes, empezará la decodificación con alta probabilidad de resultado positivo.

Diagrama

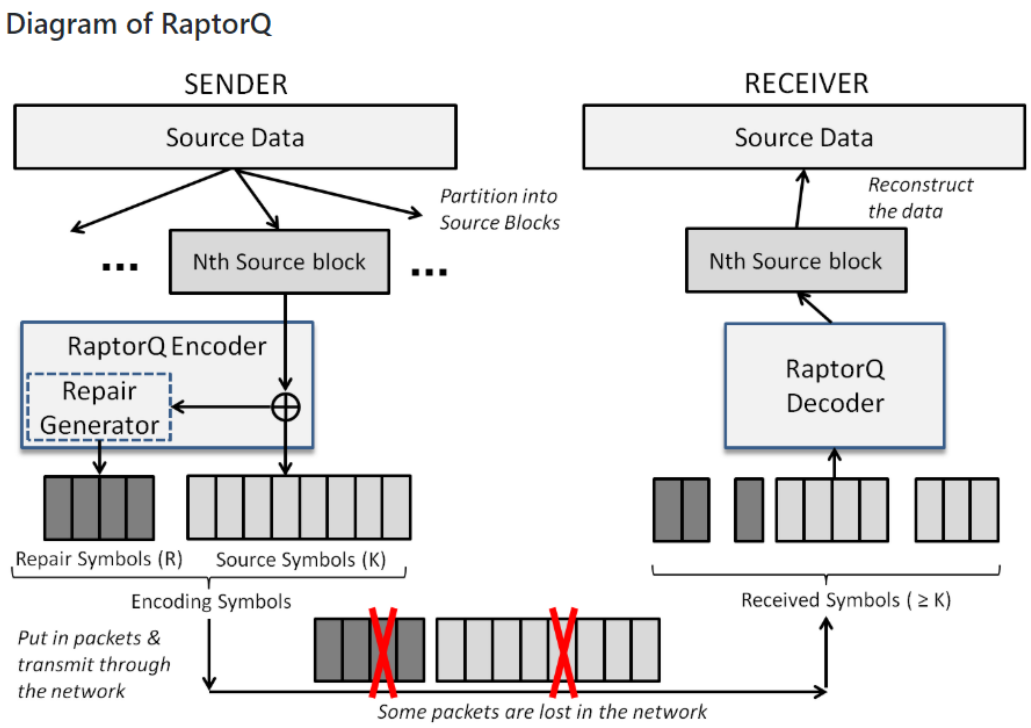


Ilustración 10. Diagrama RaptorQ

1. Los datos son **particionados** en bloques fuente.
2. Estos bloques pasan por el codificador RaptorQ y forman parte de los símbolos fuente a los cuales se les añaden los símbolos de reparación formando los nombrados anteriormente, **símbolos de codificación**.
3. Todos estos datos pasan a ser paquetes y son **trasmitidos** por la red.
4. En este proceso de transmisión pueden **perderse paquetes** en la red.
5. Los paquetes recibidos en el receptor **no son exactamente los mismo**s que los enviados por el emisor.
6. Comienza el **proceso de decodificación** en el decodificador RaptorQ que utiliza los símbolos de reparación para recuperar, en la mayor medida posible, los datos enviados.
7. **Se reconstruyen** los bloques fuente y se leen los datos recibidos.

Parámetros FEC (dentro de RaptorQ)

Cuando transmitimos datos codificados desde el emisor al receptor, el receptor debe saber un numero de **parámetros FEC** que permita un envío óptimo de los bloques. Se deben saber algunos parámetros como tamaño de bloque, en cuantos bloques se ha particionado e incluso el número de símbolos que hay dentro de cada uno de ellos. Todos estos parámetros son lo que llamamos parámetros FEC.

Los describimos a continuación:

- Tamaño de dato fuente: Tamaño de los datos enviados, en bytes. Este parámetro es importante ya que se pueden recibir bytes de relleno adicionales. Sabiendo el tamaño exacto de los datos fuente el decodificador puede calcular el tamaño real de los datos enviados y cuales son de relleno.
- **Tamaño de símbolo**: Como el mismo nombre indica representa el tamaño del símbolo en bytes ya sea reparador u origen. Hay una excepción, y es la del ultimo símbolo fuente (que puede ser de un tamaño menor ya que es el que termina con el envío).
- **Numero de bloques fuente**: Numero de bloques fuente enviados y particionados en los datos fuente. Cada bloque es codificado/decodificado individualmente y no todos contienen el mismo número de símbolos.
- **Longitud de intercalado**: Numero de sub-bloques que aparecen entre cada bloque fuente. Este valor influye en el intercalado a la hora de la codificación. A día de hoy el intercalado no se utiliza por lo que este valor es siempre 1, que indica que esta deshabilitado.

Sobrecoste de símbolos.

Como hemos visto los bloques fuente se dividen en N símbolos fuente. Siendo K los símbolos de codificación recibidos por el decodificador RaptorQ.

Ya que RaptorQ es un código sistemático, si todos los símbolos fuentes son recibidos se procede directamente a la decodificación de los mismos. Pero

cuando no es el caso, el decodificador intenta rellenar los vacíos con símbolos reparadores.

Se reciban todos los símbolos fuente o no, es posible que N símbolos recibidos de codificación no sean suficientes, ya que también se pueden perder símbolos reparadores. Para aumentar esta probabilidad se puede retrasar el decodificado hasta que el número de símbolos de codificación recibidos sea mayor que el de los símbolos fuente recibidos. Cuanto mayor sea esta diferencia, mayor probabilidad habrá de un decodificado exitoso. Esta diferencia es lo que llamamos **sobrecoste de símbolos**.

Fallo en la decodificación

A pesar de todo lo explicado previamente, pueden darse situaciones en los que el código falle. Como hemos explicado, esta probabilidad de fallo disminuye cuando mayor sea el sobrecoste de símbolos. Hay múltiples estrategias para lidiar con estos fallos. La estrategia usada depende del contexto, de la aplicación o del canal utilizado en la comunicación, entre otros. Algunas estrategias utilizadas son:

- **Pedir símbolos fuente perdidos:** Esta estrategia funciona bien si hay un canal inverso para la petición de datos. El receptor también puede indicar que símbolos reparadores se han recibido para así solo pedir aquellos de los que no se tenga ninguno de los dos símbolos.
- **Esperar más símbolos de codificación:** Esta estrategia es la utilizada cuando no hay canal inverso de comunicación y el emisor envía símbolos de manera cíclica.
- **Trabajar con los datos recibidos.** En algunos escenarios o aplicaciones se puede permitir hasta cierto punto la simple perdida de símbolos. Por ejemplo, en aplicaciones de streaming de video se tolera la perdida de algunos frames ya que, a pesar de ello, se puede consumir el contenido de manera cómoda.

4.5 OpenRQ

Este proyecto se mueve hacia el modelo de OpenRQ entre los diferentes que existen para códigos Raptor.

OpenRQ proporciona una manera de codificar y decodificar datos. Para ser más exactos, como hemos explicado anteriormente, es una implementación de *Forward Error Correction Scheme for Object Delivery (Esquema de corrección de errores por adelantado para la entrega de objetos)* como se define en RFC 6330.

OpenRQ proporciona una API Java de código abierto a los desarrolladores para incorporar erasure codes en sus protocolos de entrega.

El propósito de uso es principalmente canales con mucho ruido o con poca fiabilidad de entrega o comunicación.

Los pasos que sigue son similares al protocolo general de RaptorQ

1. Datos codificados en paquetes individuales utilizando codificadores
2. Se transmiten a uno o más receptores utilizando cualquier protocolo de comunicación
3. Los paquetes codificados son recogidos por los decodificadores hasta que se han recogido un numero de paquetes suficiente.
4. Se decodifican los datos y se obtiene la información original.

Los propios desarrolladores de OpenRQ recomiendan el uso de esta implementación en escenarios donde las retransmisiones son costosas como en broadcasting a múltiples destinatarios o cuando los enlaces de comunicación son unidireccionales.

La API 3.3.2 de OpenRQ está basada en cuatro paquetes:

- net.fec.openrq : Es el paquete principal de la API de OpenRQ.
- net.fec.openrq.decoder: Contiene las clases relacionadas con el decodificador de OpenRQ.
- net.fec.openrq.encoder: Contiene las clases relacionadas con el codificador de OpenRQ.
- net.fec.openrq.parameters: Contiene las clases relacionadas con los parámetros FEC.

Codificación

A la hora de codificar los datos destacamos las siguientes clases del paquete de codificación:

- net.fec.openrq.encoder.SourceBlockEncoder

Nos provee de la interfaz principal para codificar los bloques fuente individualmente.

- net.fec.openrq.encoder.DataEncoder

Nos provee de la interfaz principal para acceder a las instancias de los codificadores que hemos creado.

- net.fec.openrq.OpenRQ

  Nos provee de la factoría estática de métodos para crear instancias de codificadores.

Acceder a las diferentes instancias de los codificadores se puede hacer de manera individual o de manera iterativa según nuestro propósito.

Crear las instancias pueden ser o bien con offset o sin offset. Solamente necesitaríamos como parámetro un array de entrada y los parámetros FEC pertinentes para la creación.

Parametros FEC

Respecto a los parámetros FEC de la API disponemos de la clase FECParameters en el paquete net.fec.openrq.parameters.

Es una clase inmutable y solo se pueden crear instancias con los parámetros válidos.

Para la creación de las instancias debemos tener o bien el tamaño del símbolo y la longitud de los datos, o, por otro lado, parámetros derivados del protocolo de transmisión como son:

- Longitud del payload : Indica la longitud minima del payload contiendo un símbolo de codificación.

- El tamaño máximo de un bloque: Relacionado con la memoria del receptor para propósitos de control del rendimiento.

A la hora de transmitir estos parámetros disponemos de numerosos métodos para transmitirlos y parsearlos. Los métodos de lectura/escritura son:

- Byte[] : Conversion, lectura y escritura
- ByteBuffer : Conversion, lectura y escritura
- DataOutput : Escritura.
- DataInput: Lectura.
- WritableByteChannel : Escritura.
- ReadableByteChannel : Lectura.

5. Tecnología utilizada

Para la realización de este proyecto se han estudiado diferentes tecnologías para los distintos aspectos del mismo.

5.1 Java

Ilustración 11. Logo de Java

Java es un lenguaje de programación de propósito general, basado en clases y orientado a objetos.

Una de sus principales características es la de tener el menor número de dependencias posibles. Esta característica intenta que los desarrolladores escriban una vez el código de su aplicación y puedan ejecutarlo en cualquier plataforma sin necesidad de compilarse de nuevo ni modificar el código.

La posibilidad de poder usar este código en diversas plataformas sin modificar el código fuente se consigue mediante el lenguaje intermedio llamado bytecode. Para la ejecución del bytecode en cada plataforma se encuentra la JVM (Java Virtual Machine).

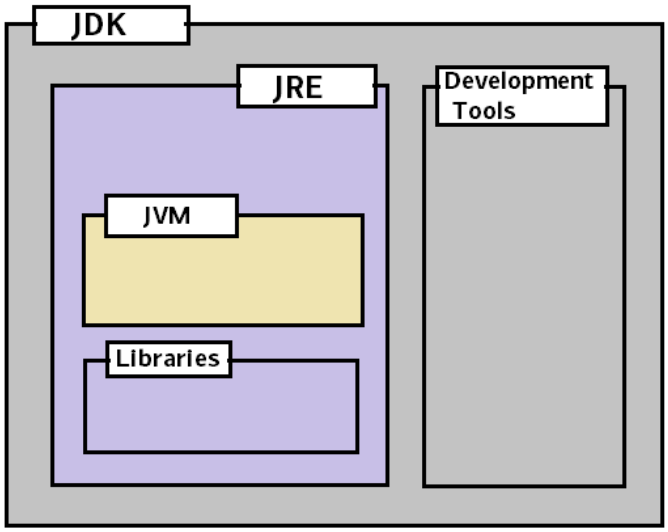


Figura 2. Arquitectura Java y JVM.

Java fue desarrollada originalmente por James Gosling de Sun Microsystems (Adquirida después por Oracle Corporation). Basado principalmente en C y C++, pero con un nivel de posibilidades a bajo nivel inferior a los citados.

Oracle es el actual propietario de la implementación de la plataforma Java SE, después de su adquisición. Basado en la implementación original está disponible para Microsoft Windows, MAC OS X, Linux y Solaris.

La implementación de Oracle está paquetizada en dos distribuciones diferentes:

1. JRE (Java Runtime Environment) que contiene las partes requeridas para la ejecución de las aplicaciones Java y esta orienta principalmente al usuario final.
2. JDK (Java Development Kit) contiene lo necesario para el trabajo de los desarrolladores e incluye herramientas como el compilador de Java, Javadoc, Jar, y un debugger.

En este proyecto nos hemos centrado principalmente en esta tecnología ya que vamos a hacer uso de una librería de código abierto como es OpenRQ, que está basada en Java. Además de esto, nos otorga la libertad de poder hacer uso de este código en diferentes entornos para hacer diferentes pruebas sin tener que cambiar ningun elemento del código debido a las ventajas que hemos comentado previamente.

5.2 Eclipse

Ilustración 12. Logo de Eclipse IDE

Existen diversos entornos de desarrollo para Java de código abierto, como por ejemplo Eclipse o NetBeans. En este caso hemos elegido Eclipse por ser uno de los más extendidos y la facilidad para encontrar soporte en caso de necesitarlo.

Eclipse es un entorno multiplataforma de desarrollo usado en programación. Según las estadísticas, es el más usado para la programación en Java. Extensible con un número grande de complementos para personalizar la interfaz.

Escrito principalmente por y para Java, aunque permite, mediante diversos complementos, la programación en multitud de lenguajes de otras aplicaciones.

Eclipse trabaja bajo la licencia *Eclipse Public License* (*EPL)* bajo la que se lanzan todos sus proyectos.

Inicialmente fue lanzado bajo la *Common Public License*, pero fue relanzado bajo la EPL tiempo después. La Free Software Foundation confirmó que las dos licencias eran licencias para software libre, pero eran incompatibles con la GNU General Public License (GPL).

Hemos elegido Eclipse principalmente para manejar las clases y la librería de OpenRQ asi como, inicialmente, hacer una parte de las pruebas de conectividad entre cliente-servidor lanzándolas dentro del propio IDE para testear los resultados.

6. Implementación

En esta parte de la memoria vamos a explicar el código elaborado para la implementación del proyecto. Explicaremos tanto la parte del servidor a la cual se conecta y está pendiente para distribuir el contenido al cliente que se conecta a él, como la parte del cliente que recibe todos los datos desde el servidor una vez que conecta y los procesa para reproducirlo después.

6.1 Servidor

En este apartado vamos a indicar los pasos que hemos seguido para montar el servidor y el código para la configuración del mismo.

Las primeras variables que hemos definido ha sido la del servidor en nuestra red montada, el puerto para las conexiones y el tamaño de los símbolos:

```
private final static int port = 5252;
private static final int SYMB_SIZE = 1420;
private static final String ipaddress = "192.168.0.2";
```

La red está configurada de manera que tenemos 192.168.0.2 en el servidor y el cliente 192.168.0.3 conectados a través de un switch. El puerto hemos configurado el 5252 para la salida/entrada de la conexión y para evitar los problemas de la fragmentación a nivel de red con el protocolo hemos puesto el tamaño máximo de símbolo a 1420 (<1500).

Una vez configurado los parámetros del servidor hemos creado el DatagramSocket recibiendo como parámetros los arrays de bytes. Tras esto reciben la dirección a través de la conexión del cliente y lo vincula con el socket.

A la hora de procesar el archivo a transmistir, en este caso un fichero multimedia, se siguen los siguientes pasos:

```
byte[] buf = new byte[1024];
FileInputStream fis = new FileInputStream(video);
ByteArrayOutputStream baos = new ByteArrayOutputStream();
for (int readNum; (readNum = fis.read(buf)) != -1;) {
    baos.write(buf, 0, readNum);
}
byte[] bytes = baos.toByteArray();
```

Como vemos en la imagen, se crea un array de bytes como objeto auxiliar seguido de un stream de entrada para procesar el archivo de video (que apunta a este fichero en el sistema) y este pasa de ser un archivo a ser un array de bytes.

Una vez procesado todo el fichero y las conexiones pasamos al apartado sobre el que se centra el proyecto que es la corrección de errores hacia delante.

Primero de todo creamos los parámetros FEC para que tanto cliente como servidor sepan qué tipo de corrección se va a seguir y vayan sincronizados en ese aspecto según el fichero que se procesa.

Creamos primero el objeto para pasar del fichero a ser un bloque fuente del cual sacaremos los parámetros FEC y posteriormente los elementos codificadores. Le pasamos como parámetros el fichero, el tamaño del fichero y la constante de tamaño de símbolo que hemos definido al principio.

```
FECParameters parametrosFEC = FECParameters.newParameters(bytes.length, SYMB_SIZE, numSBs
```

A partir de estos parámetros FEC ya podremos crear el codificador general para nuestros datos que es el elemento principal de este protocolo.

```
ArrayDataEncoder arrayDataEncoder = OpenRQ.newEncoder(bytes, parametrosFEC);
```

Después de crear el codificador general podemos utilizar el método que hace de iteración sobre todos los bloques fuente, de manera que crearemos los símbolos origen para mandarlos a través de sus respectivos paquetes de UDP para posteriormente decodificarlos y procesar el archivo saliente.

```
for(EncodingPacket encodingPacketSource : sbn.sourcePacketsIterable()) {
    byte[] sArray = encodingPacketSource.asArray();
    DatagramPacket pSource = new DatagramPacket(sArray, sArray.length, IPAddress, port);
    serverSocket.send(pSource);
}
```

Una vez enviados todos estos datagramas recibirían también los símbolos de reparación de la misma manera que hemos enviado los de origen, pero utilizando los métodos de iteración para los bloques de reparación.

6.2 Cliente

El cliente, como es normal en este tipo de escenarios cliente-servidor, recibe los datos que le llegan del servidor una vez que ha establecido conexión con el.

Cuando tiene los datos que ha recibido del servidor a través del socket vinculado con dicha conexión los decodifica y los reproduce según el protocolo que estamos implementando.

En esta sección vamos a explicar lo mismo que hemos explicado en la sección anterior, pero del lado del cliente para verlo desde los dos puntos de la conexión.

Al ejecutar el cliente se declaran las mismas constantes que del lado del servidor, pero cambiando los valores por los correspondientes al cliente.

En este caso le hemos atribuido la ip 192.168.0.3 al asignarle la ip estatica dentro de la red montada.

El cliente procesa todos los paquetes que le llegan, pero no comienza a almacenarlos hasta que reciba los parámetros FEC que son los que indican las directrices de la decodificación y son necesarios para procesarlos correctamente.

```
DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
clientSocket.receive(receivePacket);
Parsed<FECParameters> FECParse = FECParameters.parse(receiveData);
System.out.println("FROM SERVER:" + FECParse.value());
if(FECParse.isValid()) {
      parametrosFEC = FECParse.value();
      arrayDataDecoder = OpenRQ.newDecoder(parametrosFEC, 0);
```

En este código podemos ver como reciben los paquetes UDP a través del socket, crea el objeto establecido por la API para parsear los parámetros e indica por el output estándar los parámetros que le llegan para poder comprobarlos via terminal.

Una vez contrastados y parseados por el cliente se crea el decodificador general de la misma manera que habíamos creado el codificador general anteriormente y el decodificador para el bloque origen con la misma iteración que en el servidor, solo que con la diferencia que ahora ya sabemos cuántos hay al haber recibido los parámetros por UDP.

Teniendo el cliente las directrices para recibir los símbolos de codificación recibidos a través de los parámetros FEC se comienza la recepción de los paquetes que contienen los símbolos de reparación.

Crearemos el proceso iterativo mediante estas guardias para poder inspeccionar todos los paquetes e ir almacenándolos

```
boolean allBlockPacketsRead = false;
Parsed<FECParameters> latestParse = null;
while (allBlockPacketsRead == false && latestParse.isValid() == true) {
```

Sacaremos el valor de cada uno de estos paquetes y lo parsearemos hasta que el método nos devuelva que el bloque origen ya ha sido decodificado. Cuando la llamada del bloque origen nos aparezca como completada parara el proceso como nos aparece en la siguiente imagen:

```
if (latestParse.isValid() == false || (latestBlockDecoder.isSourceBlockDecoded() == true)) {
    allBlockPackets = true;
}
```

Todo esto está dentro del proceso final donde se van almacenando hasta que tenemos suficiente para realizar la decodificación.

Tras decodificar todos los datos, tendremos el fichero en el otro lado decodificado donde podemos elegir reproducirlo.

7. Resultados prácticos

En esta sección vamos a ver los resultados obtenidos a partir de las pruebas realizadas con la red montada y los equipos conectados haciendo intercambios entre ellos. El objetivo es ver la diferencia en prestaciones y cómo afecta el ruido en la red y los errores en la transmisión con las perdidas.

Para realizar las pruebas de velocidad y perdida de paquetes se ha montado una pequeña red aislada con un switch TL-SG1005D al que se han conectado 2 portátiles, un Windows 10 y un Linux (Fedora 30).

Las pruebas consisten en enviar varios ficheros de video de diferente tamaño desde el Windows al Linux mientras se regula el nivel de ruido introducido en la red.

Para facilitar las pruebas se ha limitado el ancho de banda de la conexión en la máquina Linux usando el comando ethtool:

ethtool -s enp0s31f6 speed 10 duplex half

Ethtool es una herramienta de configuración/utilidad para NICs (Network Interface Cards). Este comando permite modificar y configurar varios parámetros de las NICs como puede ser la velocidad, puerto, auto-negociación y otros parámetros. En nuestro caso en concreto simplemente modificamos la velocidad y el dúplex.

De esta forma no es necesario usar ficheros de gran tamaño y disminuye los tiempos de transferencia y de/codificación.

Para la comparativa se ha establecido una línea base enviando el fichero sin ninguna limitación en la red. Para forzar la pérdida de paquetes se introduce ruido en la red enviando datos aleatorios y descartándolos en destino a distintas velocidades. El comando usado es:

dd if=/dev/urandom bs=1M count=1000000 | pv --rate-limit 750K | nc -vvn 192.168.0.3 12346 > /dev/null 2>&1

En este ejemplo concreto se limita el envío a 750K/s

En el comando usado para meter ruido en la red y forzar la perdida de paquetes se ha usado lo siguiente:

7.1 dd

El comando dd (Dataset Definition), es una herramienta sencilla, útil, y sorprendentemente fácil de usar; con esta herramienta se puede hacer lo mismo, sobre dispositivos: discos y particiones, que con programas comerciales. En este caso usamos los parámetros:

- if: Input-File. Básicamente archivo de entrada que se quiere copiar.
- bs: Con la opción bs=1M, se consigue que tanto la lectura como la escritura se haga en bloques de 1 megabyte, (menos, seria mas lento

pero más seguro, y con más nos arriesgamos a perder datos por el camino).

- count: Copia N bloques del archivo origen, en vez de procesar hasta el final.

7.2 pv

El comando pv de unix lo utilizamos principalmente en este pipeline para limitar el ratio de transferencia con el comando --rate-limit. Se le puede añadir un sufijo al parámetro para saber la unidad que se quiere utilizar.

7.3 nc

El comando nc hace referencia al netcat. Netcat es una herramienta de red que permite a través de la línea de comandos abrir puertos en un HOST quedando netcat a la escucha asociando una Shell a un puerto en concreto.

Es una herramienta muy útil para hacer pruebas en red, rastreos de puertos a transferencias bit a bit entre dos equipos.

Originalmente solo existía para UNIX, pero acaba siendo exportada para Windows y MAC OS X.

El funcionamiento básico de netcat es el siguiente:

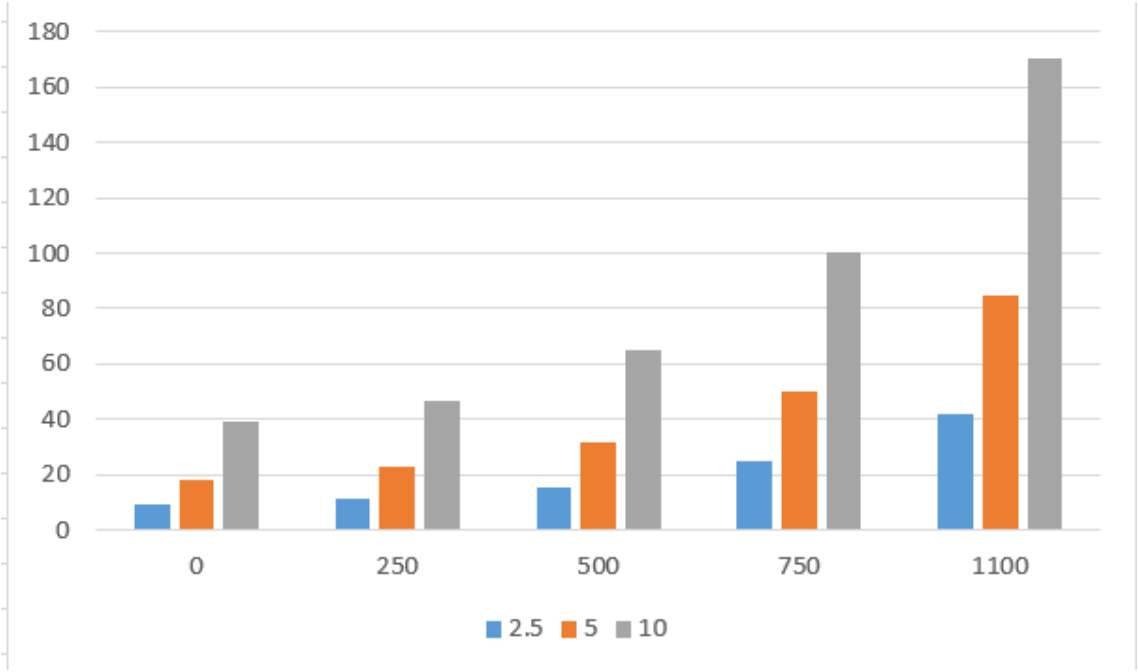
- Crear un socket para conectarse al servidor (o para hacer de servidor como hacemos nosotros ya que tenemos el netcat puesto en los dos equipos)
- Enviar todo lo que entre por la entrada estándar del socket
- Sacar por la salida por defecto todo lo recibido por el socket

En nuestro caso el parámetro –v es para el verbose para recibir más información de lo que ocurre y el –n para no resolver las IPs.

7.4 Tiempo de transmisión

En esta tabla vemos los resultados de las pruebas de velocidad de transmisión.

El tiempo de transmisión son los segundos que tarda el cliente en recibir los paquetes necesarios para el procesamiento del fichero y poder visualizarlo.



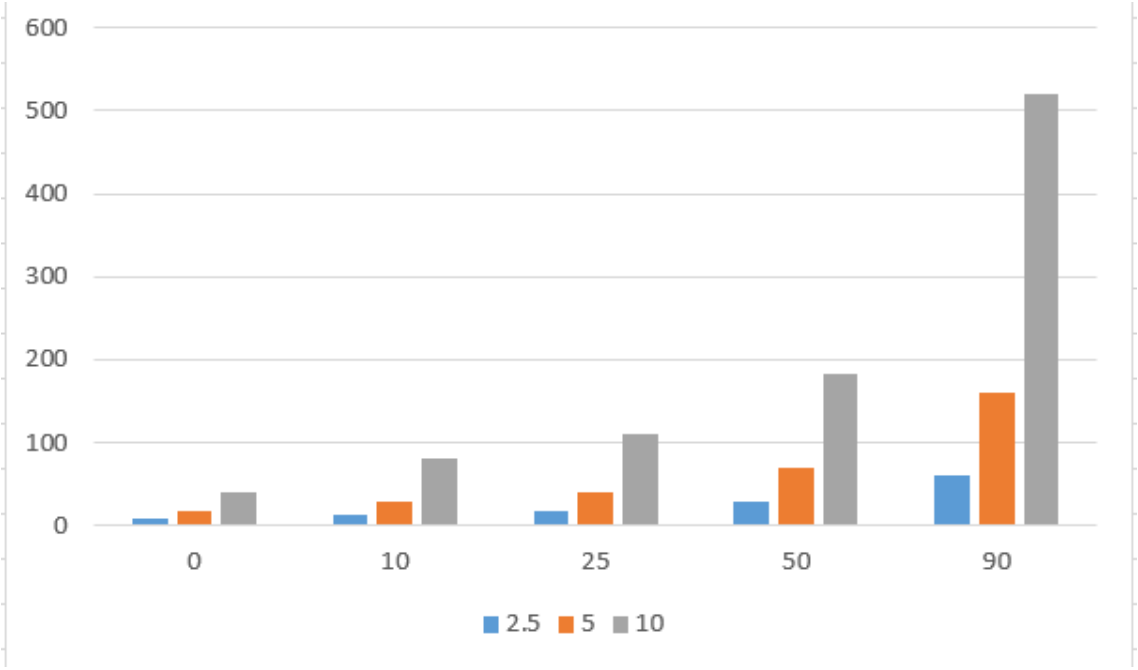
En esta tabla visualizamos los resultados en función del tamaño del archivo, del tiempo y del ruido en la red en un escenario en el que no hemos forzado la perdida de paquetes. El eje horizontal indica el ancho de banda que está ocupado teniendo en cuenta que el NIC está limitado a 10mb/s por lo que el máximo seria aproximadamente 1500. Como podemos ver en la tabla prácticamente no se aprecia diferencias en el tiempo de transmisión al haber ruido en la red en función del tamaño del archivo exceptuando en las cantidades de ruidos más altas donde se ralentiza levemente, pero sin ser realmente apreciable.

Los valores están redondeados para mejorar la comodidad de lectura.

7.5 Variando la perdida de paquetes

Esta tabla nos mostrará los resultados de tiempo para el proceso entero de nuestro proyecto aplicando tasas de packet loss diferentes en cada prueba.

Hemos ejecutado el mismo procedimiento, pero simulando la perdida de paquetes aleatorios durante el procedimiento para poder sacar los resultados. Al igual que en el caso anterior también hemos probado con diferentes tamaños de archivo.



En la tabla vemos como el tiempo crece exponencialmente ante la pérdida de paquetes. Al tener cada vez más paquetes que procesar el tiempo de procesamiento se va multiplicando como consecuencia. Obviamente, al ser el tamaño del archivo más grande también se multiplica el tiempo de procesado.

Con esta tabla podemos ver que a partir de cierto porcentaje de perdida de paquetes el tiempo de procesado se vuelve excesivamente alto, por lo que sería más beneficioso buscar otras alternativas antes que simplemente tratar los errores.

8. Conclusiones

Finalizado el proyecto podemos llegar a ciertas conclusiones sobre el trabajo del mismo y las consideraciones tenidas en cuenta. Todos los proyectos parten desde el día 1 con unos objetivos establecidos y unas marcas por cumplir en cierto tiempo.

La principal base del proyecto ha sido la motivación para unir mis dos aficiones principales.

La conclusión más evidente que podemos sacar del estudio realizado es que la corrección de errores hacia adelante es una de las prioridades más importantes, si no la más importante, ya que va de la mano con la calidad de la transmisión, condición que va a ser determinante para atraer un mayor o menor número de espectadores, llamados también clientes, para tu contenido.

Como hemos explicado durante todo el proyecto, estos últimos años el consumo de multimedia está aumentando exponencialmente haciendo que estas técnicas, entre muchas otras que hay ahora y de mayor presupuesto, sean importantísimas para proveer a los distribuidores de contenido de contenido de mayor calidad para el cliente final.

Hablando de la relación carrera-proyecto, podemos relacionarlas con las ultimas asignaturas cursadas en la especialidad de tecnologías de la información. Las tecnologías de la información se encargan de la adquisición, procesamiento, almacenamiento y difusión de “información” independientemente de su formato (voz, imágenes, textos, numérico) y por medio de la combinación de técnicas de computación y de comunicación.

Uno de los principales objetivos de la especialización es diseñar distribuciones y multimedia, que en parte hemos visto en la asignatura de Sistemas y Servicios en Red, una de mis asignaturas favoritas del Grado y que estaba relacionada directamente con el proyecto.

Además de estas asignaturas hemos tocado conceptos relacionados con otras asignaturas de la carrera al haber muchos conceptos básicos. Entre ellos podemos destacar el diseño del concepto clásico del cliente-servidor que ya vimos en las relacionadas con Redes o todos los conocimientos que hemos ido acumulando sobre el lenguaje Java en este caso o de un entorno IDE como es Eclipse que hemos usado en varias asignaturas a la hora de montar el proyecto, como otros que hemos desarrollado a lo largo de la carrera.

Volviendo al aspecto general del proyecto, en el caso particular de una empresa dedicada a la distribución de contenido hay multitud de opciones modernas para adaptar control de flujo como puede ser la de Qualcomm que hemos añadido en el contexto actual.

Como reflexión final, después de todo el estudio realizado y después de haber estado investigando sobre el tema durante todo el tiempo de preparación del trabajo podemos ver que para compañías grandes como son algunas de las nombradas en esta memoria se requiere de modelos mucho más grandes y ambiciosos como la computación paralela o modelos en la nube para dividir la carga computacional sobre la red previniendo de mejor manera los fallos.

9. Futuros trabajos

La eficiencia en la calidad de la retransmisión es un tópico que a dia de hoy sigue siendo ampliamente cuestionado.

Muchas son las compañías que ofrecen estos servicios y sigue teniendo muchos problemas a la hora de garantizar una buena calidad de retransmisión debido al alto coste que conlleva.

El presente proyecto es simplemente una pequeña aproximación a una mejora asequible y fácil de implementar, pero se puede mejorar bastante sobretodo en espectro de amplitud y en servicios a cubrir.

Como hemos visto a lo largo del proyecto a día de hoy las empresas tienen infraestructuras montadas de gran envergadura con computación paralela por lo que uno de los objetivos podría ser aplicar paralelismo para estar menos limitado por la capacidad de computación de la máquina y aligerar tiempos de decodificación y de corrección de errores.

Otra opción sería la posibilidad de poder aplicar multithreading. Permitiría codificación de los bloques en paralelo, permitiendo una mejora del rendimiento sobre todo en las iteraciones sobre los bloques o paquetes para corregirlos y parsearlos.

Bibliografía

[1] https://openrq-team.github.io/openrq/
[2] https://tools.ietf.org/html/rfc6330
[3] James F. Kurose and Keith W. Ross. Computer Networking: A Top-Down Approach Featuring the Internet, 6th Edition. Addison Wesley, 2013.
[4] Douglas E. Comer. Internetworking with TCP/IP. Principles, Protocols, and Architectures (4th Edition), volume 1. Prentice Hall, 2000.
[5] A. Shokrollahi, “Raptor Codes”, IEEE Transactions on Information Theory, vol. 52, no. 6, pp. 2551–2567, June 2006.
[6] J. Lopes and Nuno Neves, “Stopping a Rapid Tornado with a Puff” 35th IEEE Symposium on Security and Privacy, May 2014.
[7] https://github.com/olanmatt/orq
[8] M. Mitzenmacher (2004). "Digital Fountains: A Survey and Look Forward". Proc. 2004 IEEE Information Theory Workshop (ITW).
[9] https://wikipedia.org
[10] https://blog.twitch.tv/twitch-engineering-an-introduction-and-overview-a23917b71a25
[11] https://livestream.com/blog/62-must-know-stats-live-video-streaming
[12] https://whatis.techtarget.com
[13] https://tubularinsights.com/2019-internet-video-traffic/#ixzz4JU3cxVXv
[14] Anne Aaron and David Ronca (2015), “Netflix Technology Blog”, Medium, December 2015.
[15] Ronald J. Leach (1993), “C and Linux/Unix Commands : Two Books”.