UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

SCALABLE VIDEO CODING SOBRE TCP

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO

ANDRÉS EDGARDO SANHUEZA GUTIÉRREZ

PROFESOR GUÍA:
CLAUDIO ESTÉVEZ MONTERO

MIEMBROS DE LA COMISIÓN:
HUGO MÉRIC
CÉSAR AZURDIA MEZA

SANTIAGO DE CHILE
2015

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE: Ingeniero Civil Eléctrico
POR: Andrés Edgardo Sanhueza Gutiérez
FECHA: Santiago, 2015
PROFESOR GUÍA: Claudio Estévez Montero

Scalable Video Coding sobre TCP

En tiempos modernos la envergadura del contenido multimedia avanza más rápido que el desarrollo de las tecnologías necesarias para su correcta difusión a través de la red. Es por esto que se hacen necesarios nuevos protocolos que sirvan como puente entre ambas entidades para así obtener un máximo de provecho del contenido a pesar de que la tecnología para distribuirlos aún no sea la adecuada.

Es así, que dentro de las últimas tecnologías de compresión de video se encuentra *Scalable Video Coding* (SVC), la cual tiene por objetivo codificar distintas calidades en un único *bitstream* capaz de mostrar cualquiera de las calidades embebidas en éste según se reciba o no toda la información. En el caso de una conexión del tipo *streaming*, en donde es necesaria una fluidez y fidelidad en ambos extremos, la tecnología SVC tiene un potencial muy grande respecto de descartar un mínimo de información para privilegiar la fluidez de la transmisión. El software utilizado para la creación y manipulación de estos *bitstreams* SVC es *Joint Scalable Video Model* (JSVM).

En este contexto, se desarrolla el **algoritmo de deadline** en Matlab, que omite información del video SVC de acuerdo a qué tan crítico sea el escenario de transmisión. En este escenario se considera la percepción de fluidez del usuario como medida clave, por lo cual se prioriza mantener siempre una tasa de 30 fps a costa de una pérdida de calidad mínima. El algoritmo, omite información de acuerdo a qué tan lejos se esté de este *deadline* de 30 fps, si se está muy lejos, se omite información poco relevante, y si se está muy cerca, información más importante.

Los resultados se contrastan con TCP y se evalúan para distintos valores de RTTs, cumpliendo totalmente el objetivo para valores menores a 150 ms que resultan en diferencias de hasta 20 s a favor del algoritmo de deadline al término de la transmisión. Esta mejora en tiempo de arribo no descarta información esencial y sólo degrada ligeramente la calidad del video en pos de mantener la tasa de 30fps.

Por el contrario, en escenarios muy adversos de 300 ms en RTT, las omisiones son de gran envergadura y comprometen *frames* completos, en conjunto con una degradación generalizada del video y la aparición de artefactos en éste. Por tanto la propuesta cumple los objetivos en ambientes no muy adversos.

Para toda la simulación se uso un video en movimiento de 352x288 y 150 *frames* de largo.

A mis padres Ximena, Jorge
y a mi hermano Pablo

Agradecimientos

Sin el incondicional apoyo emocional, las tardes de película, la fruta nocturna o los desayunos de mi madre Ximena, no sería hoy, ni por asomo, la persona que soy hoy. Me hizo ver lo sabio de los pequeños detalles que resultan en cambios importantes para todos nosotros.

Agradezco a mi padre Jorge, por su sabiduría, su constante preocupación por mis estudios y los de mi hermano, además de su peculiar sentido del humor y perseverante responsabilidad.

A mi hermano Pablo, por las incontables horas de ocio que pasamos juntos desde niños. Las risas y peleas que espero se sigan repitiendo conforme avancen los años.

A mis amigos de siempre y para siempre, con quiénes he disfrutado todos estos años de universidad, Carolina, Daniel, Rodrigo, Juampi, Gonzalo, Pedro, Fran, Sebástian, Victor, Nicolás y Camilo.

A Camila, mi compañera, por su cariño, apoyo y ayuda en todo lo necesario durante el transcurso de éste trabajo.

A con quiénes nos desvelamos largas horas estudiando hasta la madrugada, aunque más risas que estudio debo decir y con quiénes terminabamos tareas en las micros. Javier el que siempre fluye, Gonzalo el furioso, y por supuesto el carismático y siempre preocupado Alex.

A mis amigos del 5to piso, por siempre ir a molestarlos, a mi querido laboratorio OWL, con Claudio, Boris, Diego y Liliana por los almuerzos de risas todos con temas interesantes.

Como olvidar también a mis compañeros del coro, siempre dispuestos con un ánimo que parece nunca fatigarse a cantar, reirse, y unirnos en una sóla voz. Jamás olvidaré mi paso por un grupo de personas tan cálido y receptor.

Finalmente a mi profesor guía Claudio por su constante apoyo y paciencia para guiarme y orientarme a través de todos estos años. También a Cesar y Hugo quiénes formaron parte de mi comisión y dieron su apoyo de varias formas para el término de este trabajo.

Todos ustedes, y quienes no caben en esta página, son el recuerdo vivo y latente de estos 6 años, que jamás olvidaré y espero, de todo corazón sigamos en contacto por siempre y para siempre.

Sinceramente, yo estoy hecho... de todos ustedes.

Tabla de Contenido

1. Introducción **1**
- 1.1. Motivación . . . 1
- 1.2. Objetivos del Trabajo de Memoria . . . 2

2. Conceptos Básicos y Revisión Bibliográfica **3**
- 2.1. Modelo TCP/IP y Capa de Transporte . . . 3
- 2.2. Ventana de Congestión . . . 4
- 2.3. Video Streaming . . . 5
- 2.4. Codificación de Video y Scalable Video Coding . . . 5
- 2.5. Group of Pictures (GOP) . . . 7
- 2.6. Peak Signal to Noise Ratio . . . 8
- 2.7. Estado del Arte . . . 9

3. Implementación **11**
- 3.1. Codificación del Video . . . 12
- 3.2. Generación de *Trace File* . . . 14
- 3.3. Generación de *Packet Trace File* . . . 15
- 3.4. Transmisión y Recuperación de Paquetes . . . 17
  - 3.4.1. Recuperación paquetes base-I . . . 18
  - 3.4.2. Recuperación paquetes base-B . . . 19
- 3.5. Algoritmo de Deadline . . . 20
- 3.6. Reconstrucción del Video . . . 22
  - 3.6.1. Eliminación de Calidad . . . 22
  - 3.6.2. Eliminación de Base . . . 23
  - 3.6.3. Percepción de Pérdidas . . . 25

4. Resultados **29**
- 4.1. Escenario Nacional . . . 30
- 4.2. Escenario Continental . . . 33
- 4.3. Escenario Transcontinental . . . 35

5. Conclusiones y Trabajo Futuro **37**
- 5.1. Conclusiones . . . 37
- 5.2. Trabajo Futuro . . . 38

6. Bibliografía **39**

Índice de tablas

3.1. Archivo de Configuración **main** de JSVM 12
3.2. Archivo de Configuración **layer1** de JSVM 12
3.3. Archivo de Configuración **layer0** de JSVM 13
3.4. Información de Codificación del Video 13
3.5. *Trace File* coloreado de 2 capas de calidad 14
3.6. *Packet Trace File* coloreado para 2 capas de calidad 15
3.7. Packet Trace File con pérdidas y Recuperación base-I 18
3.8. Packet Trace File con pérdidas y Recuperación base-B 19
3.9. Eliminación de Calidad 22

Índice de Ilustraciones

2.1. Modelo de capas TCP/IP 4
2.2. Ventana de Congestión TCP Reno 5
2.3. Principio SVC, a partir de un sólo *bitstream* se decodifican distintas calidades para diferentes necesidades y/o equipos. 6
2.4. Distintos tipos de *frames* dentro de un GOP [1] 7
2.5. Esquema de transmisión DASH [12] 9

3.1. Modelo bloques general 11
3.2. Ventana de Congestión Simulada 17
3.3. Niveles de Deadlines 20
3.4. Algoritmo de Deadline 21
3.5. Eliminación de Calidad 23
3.6. Dependencia de Frames 25
3.7. Pérdida de Base 26
3.8. Pérdida Parcial Directa 26
3.9. Pérdida Parcial Indirecta 27
3.10. Pérdida de Calidad 28

4.1. Tiempos de Transmisión de Video por Frame, Loss rate: 0.01, RTT: 0.05s . . 30
4.2. PSNR y Ventana de Congestión Escenario Nacional 31
4.3. Reconstrucción de video, frames del 77 al 81 32
4.4. Tiempos de Transmisión de Video por Frame, Loss rate: 0.01, RTT: 0.15s . . 33
4.5. PSNR y Ventana de Congestión Escenario Continental 34
4.6. Reconstrucción de video, frames del 1 al 5 34
4.7. Tiempos de Transmisión de Video por Frame, Loss rate: 0.01, RTT: 0.3s . . 35
4.8. PSNR y Ventana de Congestión Escenario Transcontinental 36
4.9. Reconstrucción de video, frames del 13 al 18 36

Capítulo 1

Introducción

1.1. Motivación

El tráfico de red del tipo video *streaming* es cada vez más usado por los usuarios alrededor del globo, ya sea por trabajo o entretención. Dentro de esta última categoría cabe mencionar que los servicios de Youtube y Netflix en conjunto ocupan más del 50 % del tráfico de red en EE.UU.

Adicionalmente, existe una tendencia de tener cada vez mejores resoluciones para la reproducción de video, prueba de este hecho es la reciente aparición de los televisores *UltraHD*, de hasta 16 veces la resolución *Full HD* o también la opción de resolución en Youtube. Esto último, sumado a la exigencia de una reproducción de video siempre continua y suave del lado del usuario, han provocado que las nuevas experiencias multimedia sean incapaces de ser disfrutadas en su totalidad.

Frente a este hecho, las actuales tecnologías de transmisión de datos son insuficientes para enfrentar el mencionado escenario, y por este motivo es urgente encontrar nuevas tecnologías que permitan una justa co-existencia entre los contenidos multimedia y la capacidad respectiva para enviarlas a través de una red cada vez más saturada de información.

Es así que es esencial el entendimiento de los últimos algoritmos de compresión de video como MPEG-4 *Advanced Video Coding* (o H.264) en conjunto con la naturaleza misma de la transmisión de datos actual (TCP o UDP) para lograr crear nuevos y mejores estándares para la transmisión de contenido *streaming*.

Dentro de las últimas tecnologías de compresión se encuentra *Scalable Video Coding* (SVC) que permite, en palabras simples, la codificación única de un video en distintas calidades, para luego ser mostrado en pantalla según la calidad que se requiera. SVC tiene un gran potencial para desarrollar nuevos protocolos de transmisión y un primer acercamiento es mostrado en el presente trabajo.

1.2. Objetivos del Trabajo de Memoria

A continuación se presentan los objetivos generales y específicos planteados para este trabajo de memoria.

Objetivo General

Diseñar e implementar una cooperación de tipo *cross-layer* entre las capas de transporte y aplicación del modelo OSI para redes orientadas a contenido *streaming*:

- Reducir el tiempo de espera del usuario para el display del video.
- Reducción a costa de un mínimo de pérdida de calidad.

Objetivos Específicos

- Implementar un protocolo *cross-layer* que permita la coordinación de envío de las calidades de SVC con requerimientos de visualización del usuario (reproducción continua de video).
- Definir en qué circunstancias la nueva propuesta actuará y cómo lo hará.
- Obtención de medidas de error y velocidad para el escenario tradicional y la nueva propuesta.

Capítulo 2

Conceptos Básicos y Revisión Bibliográfica

Este capítulo tiene por objetivo contextualizar al lector en los diversos tópicos que se abordarán durante el resto del trabajo de memoria. Conceptos de comunicaciones como capa de tranporte y ventana de congestión así como también conceptos de compresión de video, referentes a video streaming y codificación. Como punto aparte también se hará una breve descripción de métricas de comparación de video a fin de contrastar resultados en partes posteriores de este trabajo.

2.1. Modelo TCP/IP y Capa de Transporte

Para facilitar el entendimiento de los distintos procesos involucrados en una conexión de red, se establece en los años 70 un modelo jerárquico de encapsulamiento de la información hecho por los señores Vinton Cerf y Robert E.Kahn denominado *Transmission Control Protocol and Internet Protocol* (TCP/IP). Este modelo jerárquico, en términos simples, encapsula la información por capas, dando distintas claves de señalización según la capa en donde se encuentre. De esta manera la interconexión de distintos dispositivos se facilita, y los distintos formatos entre capas quedan estandarizados.

El esquema tradicional del modelo TCP/IP se presenta en la figura 2.1, mostrando el encapsulamiento de la data original (en azul), desde la capa más cercana al usuario (capa 4, de aplicación) hasta la capa física (capa de enlace, conexión por cable, fibra, inalámbrico etc). Es en la capa de transporte en donde se desarrolla el presente trabajo, en esta capa se encuentran los protocolos de control de congestión y de recuperación de paquetes que serán interconectados con la capa de aplicación, específicamente aplicaciones de video *scalable video coding*.

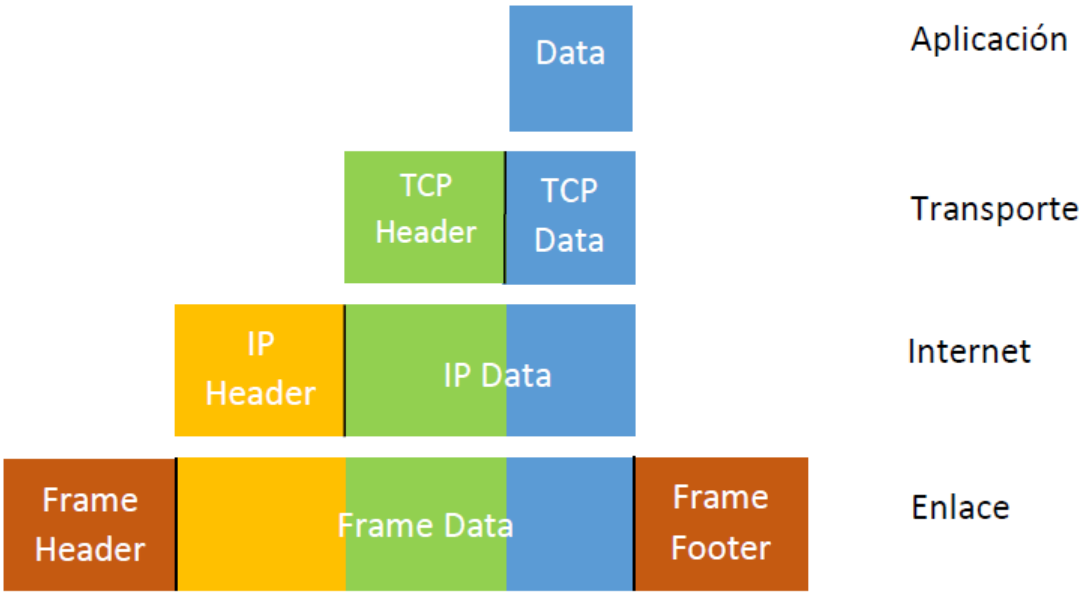

Figura 2.1: Modelo de capas TCP/IP

2.2. Ventana de Congestión

La ventana de congestión es un proceso inherente a la transmisión de datos, propio del protocolo TCP en la capa de transporte y además corresponde a la cantidad de paquetes enviados simultáneamente por unidad de tiempo. La cantidad de paquetes enviados en el tiempo (que es variable) se denomina ventana de congestión. Así una ventana puede consistir en 4,5 ó más paquetes dependiendo de diversos algoritmos de control llamados Control de Congestion de TCP [6], cuyo objetivo es evitar la congestión y la co-existencia de otros protocolos en la red. Entre estos destacan:

1. *Slow Start:* Crecimiento cuadrático de la ventana de congestión hasta alcanzar un umbral determinado (*sstresh*) por la cantidad de tráfico en el canal.
2. *Congestion Avoidance:* Crecimiento lineal de la ventana de congestión hasta que algún paquete se extravíe.

Estos dos mecanismos, en conjunto con *Fast Recovery* y *Fast Retransmit*, constituyen la mayoría del control de tráfico mundial en conexiones basadas en TCP. Una ventana típica de TCP luce como se muestra en la figura 2.2, en donde los colores indican en qué fase del algoritmo se encuentra el control.

Los instantes en que la ventana de congestión reduce su tamaño corresponden a dos eventos:

1. *ACK triple duplicado:* Significa que el receptor no recibió el paquete esperado, por lo cual responde con una señalización especial. La no recepción del paquete implica un tráfico considerable en el enlace, por lo cual se disminuye el tamaño de la ventana.

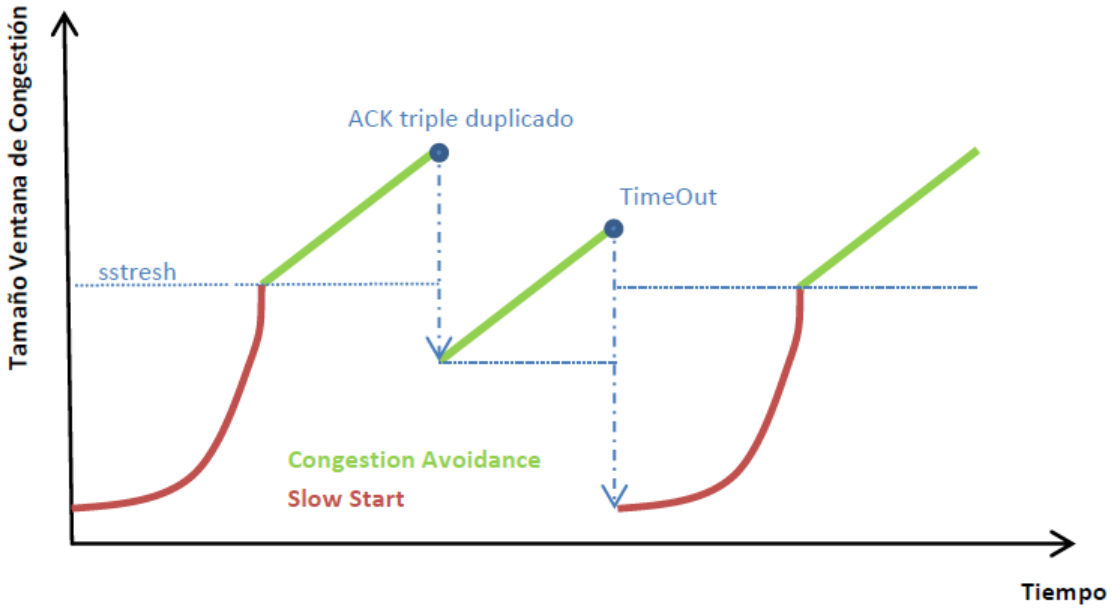

Figura 2.2: Ventana de Congestión TCP Reno

2. *TimeOut:* Indica que tras un tiempo establecido por el router, nunca se obtuvo un ACK triple duplicado. Esto indica que el estado del enlace es de alto tráfico y la ventana de congestión debe reducirse drásticamente.

Ambos esquemas anteriores corresponden a la implementación Reno de control de congestión.

2.3. Video Streaming

El concepto de *video streaming*, consiste en mostrar al usuario la información al mismo tiempo que ésta es obtenida desde un servidor. *Video Streaming* constituye todo un desafío para los protocolos de transporte de información, ya que se requiere que la información sea codificada, enviada y decodificada a una tasa de al menos 25 imágenes por segundo. Más aún, en tiempos modernos, en donde la exigencia ya no sólo trata de la fluidez del video, sino que también de la calidad de éste mismo, en términos de la calidad temporal (imágenes por segundo), espacial (resolución) y de cuantización de la información.

2.4. Codificación de Video y Scalable Video Coding

La codificación tiene como principal objetivo comprimir el video mediante la eliminación de información redundante (similaridad entre frames, *data* imperceptible para el ojo humano, etc.), para luego, convertir el archivo de video en un formato de bits denominado *bitstream* que sea adecuado para el dispositivo en que se quiera reproducir.

Existen muchos formatos de codificación, entre estos *HEVC*, *WMV*, *WEBm* y *H.264*, cada uno con sus respectivas ventajas y desventajas. De estos formatos, H.264 es la tecnología de compresión de video más popular usada para grabar, comprimir y distribuir el contenido de video. Usada en los discos de Blu-ray, Vimeo, Adobe Flash Player y el conocido Youtube. Dicha tecnología fue concebida originalmente en 2003 y posee una serie de características que lo hacen destacar frente a otros estándares. Dentro de las más importantes están:

- *Inter Picture Prediction*
- *Motion Compensation/Estimation*
- ***Scalable Video Coding***

Scalable Video Coding es en realidad una reciente extensión de H.264 que, en palabras simples, permite codificar distintas calidades de un mismo video (c/r a frames por segundo, resolución etc.) en un *bitstream* único [4] [11] para luego ser usado de acuerdo a las necesidades de *display* de los receptores en la red, como se muestra en la ilustración 2.3.

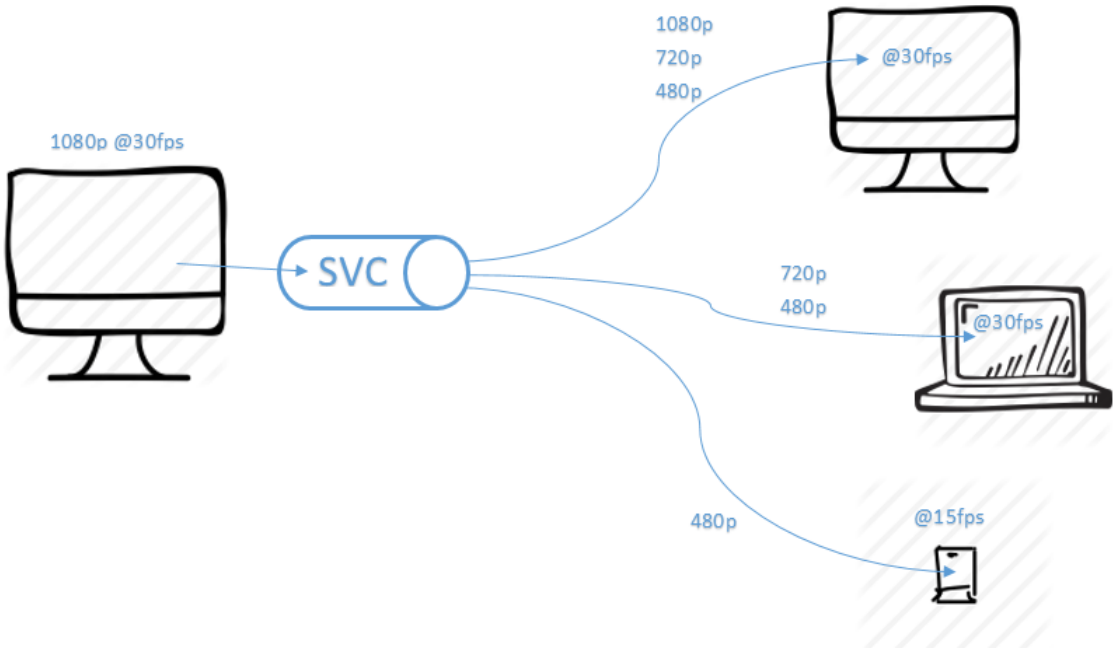

Figura 2.3: Principio SVC, a partir de un sólo *bitstream* se decodifican distintas calidades para diferentes necesidades y/o equipos.

En el esquema tradicional de codificación, se hacen necesarias tantas codificaciones como calidades del video se quieran tener, por lo cual SVC presenta una ventaja comparativa en términos de procesamiento de datos, y un potencial muy alto en lo que se refiere a la transmisión de la información propiamente tal.

2.5. Group of Pictures (GOP)

Group of Pictures es una secuencia de imágenes o *frames* dentro de un video. Éste generalmente se mantiene durante todo el video y su principal característica es que todos los *frames* pueden ser decodificados dentro de este GOP.

Existen distintos tipos de *frames* dentro del GOP, cada uno cumple una función esencial para la correcta estimación del video final.

- I-frame (*Intra frame*): Es el *frame* más importante dentro del GOP, está codificado independientemente de los demás frames y sirve cómo base de predicción del resto de frames en el GOP. Cada GOP empieza y termina con un I-Frame.
- P-frame (*Predictive frame*): Contiene información de diferencia relativa a los *frames* codificados previamente.
- B-frame (*Bipredictive frame*): Contiene información de diferencia relativa a los *frames* codificados previa y posteriormente.

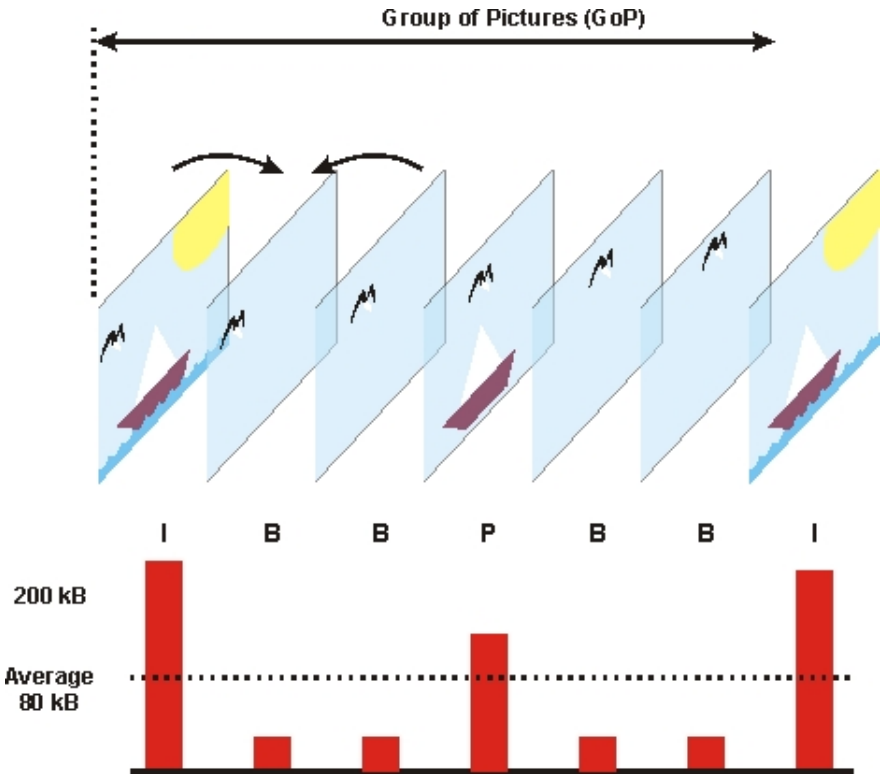

Figura 2.4: Distintos tipos de *frames* dentro de un GOP [1]

El I-*frame* no requiere más información que él mismo para decodificarse, es el *frame* más pesado ya que contiene toda la información de la imagen. Por otro lado los *frames* P y B se complementan con los *frames* I para seguir formando *frames* completos con técnicas como *Motion Compensation/Estimation*.

El GOP *size* se define como la distancia entre *frames* del tipo I dentro de un GOP (6 para el caso de la figura 2.4). Poner I-*frames* a intervalos regulares limita la propagación

de error de estimación más allá del GOP, por lo que muestra ser una técnica robusta de codificación. Se puede apreciar en la figura 2.4 que los I-*frames* son los más cargados de información, mientras que los B-*frames* y P-*frames* se predicen ayudados de sus predecesores y sucesores. Es importante señalar que los B-*frames* y P-*frames* no se muestran directamente en la reproducción del video, sino que se usan como referencia (junto a sus vecinos) para reconstruir el *frame* original sin necesidad de enviarlo por completo (de otra forma, sólo se enviarían *frames* del tipo I).

2.6. Peak Signal to Noise Ratio

Peak Signal to Noise Ratio (PSNR) es una medida usada generalmente en el procesamiento de senãles que corresponde, en términos simples, a una representación de qué tan fuerte es la senãl en relación al ruido distorsionador que afecta la calidad de su representación.

En imágenes, se utiliza con el propósito de implementar una medida que cuantifique la reconstrucción/mejoramiento de la imagen en base a algoritmos (codificaciones, compresiones etc.) respecto de la imagen original. Es en general una medida de que tan distintas son 2 imágenes. Esta medida, no es presentada en forma lineal debido al gran rango dinámico que puede experimentar esta cuantificación en imágenes, por lo cual generalmente se expresa en decibeles. Su representación matemática es:

$$PSNR = 10 \cdot \log_{10} \quad \frac{MaxErr^2 \cdot w \cdot h}{\sum_{j=0}^{h} \sum_{\mathrm{i}=0}^{w} (x_{\mathrm{i}j} - y_{\mathrm{i}j})^2} \qquad \mathrm{d}B, \tag{2.1}$$

donde $MaxErr$ corresponde a el máximo valor de pixel en la imagen (255 para este trabajo), w y h son las dimensiones del video en pixeles, x e y son los valores del pixel en la imagen original y la reconstruida respectivamente.

Además, esta medida se puede tomar para cada componente del video, por ejemplo si se trabaja con un formato de video *.yuv* entonces existirán : Y-PSNR, U-PSNR y V-PSNR. En el contexto del presente trabajo, se usará el Y-PSNR como medida principal de reconstrucción de video ya que es la componente que corresponde a la luminiscencia del video, factor al cual es más sensible el ojo humano.

2.7. Estado del Arte

Muchas son las formas actuales de distribuir el contenido de video por internet, sin embargo una que ha ido adquiriendo particular popularidad es el *Dynamic Adaptative Streaming over HTTP* (DASH), siendo el protocolo adoptado recientemente por Youtube y Netflix, que en conjunto son responsables de aproximadamente un 50 % del tráfico en EE.UU [7].

DASH consiste en la codificación del video en distintas calidades (y por tanto en distintos *bitrates*) del lado del servidor, además cada una de estas codificaciones se separa en segmentos de tiempo iguales. De esta forma, se envía al usuario una calidad pertinente al *throughput* actual de la conexión, formando así un resultado final compuesto de distintas calidades de video.

El esquema general de DASH se puede apreciar en la figura 2.5

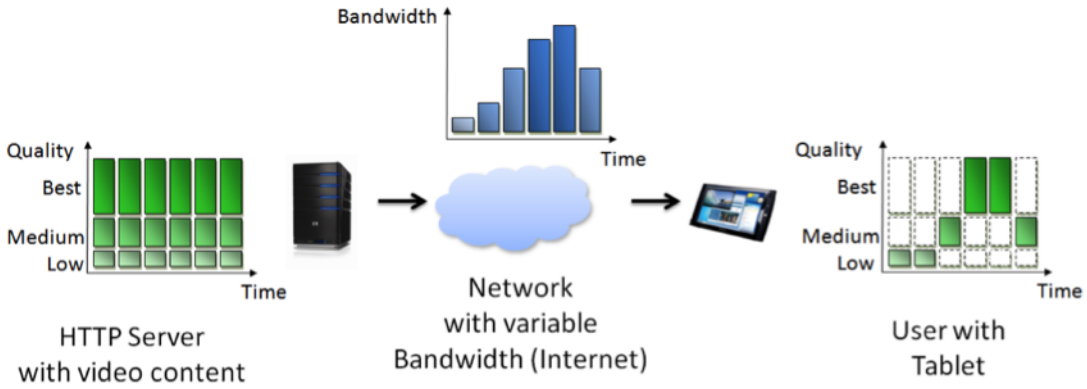

Figura 2.5: Esquema de transmisión DASH [12]

Una desventaja de DASH es el tamaño fijo de los segmentos de video (el valor típico es menor a 10s), forzando el envío de una determinada calidad de video durante el tiempo que dure el segmento. El throughput bien puede variar durante ese intervalo de tiempo, pudiendo ser menor al requerido para enviar la calidad de video seleccionada. De esta forma el usuario percibe el desajuste esperando el tiempo requerido para que los paquetes atrasados lleguen a destino, lo cual no es recomendable si se está ante un escenario streaming.

Otra desventaja es la repercusión que tiene sobre el almacenamiento en los servidores. DASH necesita tener tantas repeticiones de un video como calidades se quiera tener. Por ejemplo, si es necesario tener 3 tipos de calidades en resolución, entonces deben existir 3 copias del video en 1024x768, 800x600 y 640x480 en el servidor. Sin embargo, para abordar este problema, se ha optado por estrategias como almacenar sólo fracciones del video en distintos segmentos de calidad, utilizando el hecho de que muchos videos online no son visualizados hasta el final [2], pero de todas formas usar DASH sigue repercutiendo en la capacidad de almacenamiento del servidor.

En este aspecto, SVC tiene la ventaja al ser capaz de proveer distintos niveles de calidad solamente con una codificación.

Tradicionalmente el protocolo *Real-Time Protocol* (RTP) y su aplicación dedicada *Real Time Streaming Protocol* (RTSP) solucionan el problema del contenido en tiempo real, pero tienen la desventaja de ser muy agresivos con el ancho de banda, impidiendo una justa coexistencia con TCP. Además son bloqueados por diferentes *firewalls* a través de la red [9] y pueden llegar a descargar la totalidad del video sin que el usuario lo visualice por completo, desperdiciando recursos. Estos protocolos están basados en *User Datagram Protocol* (UDP), protocolo no orientado a la confiabilidad, por lo que una pérdida asociada a un I-*frame* del GOP H.264 repercute muy negativamente en el resultado final del video.

Con respecto a soluciones dedicadas a H.264, en [3] se propone adaptar la calidad del video usando la opción *Constant Bit Rate* (CBR) de x264, en donde la información de red a la capa de aplicación es provista por *Datagram Congestion Control Protocol* (DCCP). Otro enfoque basado en H.264/SVC y UDP es estudiado en [8], en donde la estimación del ancho de banda disponible es realizada por el cliente de la aplicación. La limitante de este trabajo es que sólo se implementa la recuperación de paquetes del tipo capa-base, perdiendo la capacidad de recuperar capas de calidad cuando sea posible.

Todos los esquemas anteriores responden a las necesidades de la red, pero independiente de las necesidades del usuario. Para una aplicación streaming, es necesario que el usuario perciba una fluidez en la entrega de datos multimedia (audio y video), por lo que en el presente trabajo se realiza un primer acercamiento a este enfoque.

Para dicho fin, se utilizará un critero de *Deadline* de Visualización, el cual indicará en qué instante es necesario que lleguen los *frames* a destino antes de que el usuario perciba latencia en la conexión y por tanto tiempos de espera para el video. Así es como se optará por no enviar paquetes de calidad SVC de acuerdo a qué tan lejos se esté del deadline. Los paquetes omitidos son seleccionados en base a si son del tipo base I, calidad I, base B o calidad B y a qué tan lejos se está del deadline. Es decir, es una omisión selectiva de paquetes para ajustarse al deadline de visualización.

Capítulo 3

Implementación

Para la implementación se usará **Joint Scalable Video Model** (JSVM) en lo que respecta a codificación/decodificación de video y **Matlab 2013a** para la simulación de transmisión del video. JSVM es el software de referencia de uso de SVC creado por ISO/IEC Moving Pictures Experts Group (MPEG) y la ITU-T Video Coding Experts Group (VCEG). JSVM está escrito en C++ y se distribuye como source code [5]. JSVM es una herramienta versátil, que permite la codificación y decodificación de video para posteriormente generar bitstreams de tipo SVC, además permite la generación de *trace files* que facilitan la eliminación de calidad por frame para el video decodificado.

Los *trace files* generados por JSVM serán utilizados posteriormente por Matlab para la implementación del algoritmo final. Por lo cual su generación, entendimiento y modificación son la piedra angular del presente trabajo.

A continuación se presenta un esquema general de los pasos a seguir en la figura 3.1, a modo de guía para el resto de la lectura. El bloque resaltado es fundamental en la propuesta.

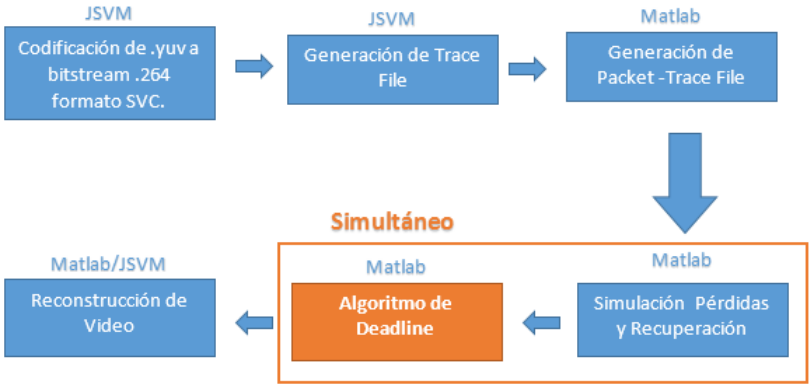

Figura 3.1: Modelo bloques general

3.1. Codificación del Video

JSVM usa archivos *.txt* como archivos de configuración para codificar el video. Toda codificación necesita el archivo principal *main.txt* que da los lineamientos generales de la ésta, y además tantos archivos de texto como capas se quieran obtener de SVC.

Se usará un solo video en formato *.yuv* de 150 frames en 352x288 para toda la simulación el cual será codificado en dos capas de calidad, mediante un archivo de texto *main* y otros dos de capa SVC. El archivo main puede visualizarse en la tabla 3.1:

```
# JSVM Main Configuration File (snr_mgs_main.cfg)

OutputFile              encoded/mgs_bus.264     # Bitstream file
FrameRate                       30              # Maximum frame rate [Hz]
FramesToBeEncoded               150             # Number of frames (at input frame rate)
GOPSize                         8               # GOP Size (at maximum frame rate)
IntraPeriod                     8               # Intra Period
BaseLayerMode                   0               # Base layer mode (0,1: AVC compatible,
                                                #                  2: AVC w subseq SEI)
CgsSnrRefinement                1               # SNR refinement as 1: MGS; 0: CGS
EncodeKeyPictures               1               # Key pics at T=0 (0:one, 1:MGS, 2:all)
MGSControl                      2               # ME/MC for non-key pictures in MGS layers
                                                # (0:std, 1:ME with EL, 2:ME+MC with EL)
SearchMode                      4               # Search mode (0:BlockSearch, 4:FastSearch)
SearchRange                     2               # Search range (Full Pel)
NumLayers                       2               # Number of layers
LayerCfg                snr_mgs_layer0.cfg      # Layer configuration file
LayerCfg                snr_mgs_layer1.cfg      # Layer configuration file
```

Tabla 3.1: Archivo de Configuración **main** de JSVM

Dentro de las opciones de configuración más importantes para la posterior transmisión del video se encuentra el tamaño del GOP (fijado en 8), y el FrameRate (30) que da lugar a curva de *deadline* de visualización de la cual se hablará mas adelante.

A su vez, las capas de calidad tienen la configuración mostrada en las tablas 3.2 y 3.3:

```
# JSVM Layer Configuration File 1 (snr_mgs_layer1.cfg)

InputFile       videoref/bus150.yuv     # Input file
SourceWidth             352             # Input frame width
SourceHeight            288             # Input frame height
FrameRateIn             30              # Input frame rate [Hz]
FrameRateOut            30              # Output frame rate [Hz]
InterLayerPred          1               # Inter-layer Predict (0: no, 1: yes, 2: adaptive)
MGSVectorMode           0               # MGS vector usage selection
#MGSVector0 4                           # Specifies 0th position of the vector
#MGSVector1 4                           # Specifies 1th position of the vector
#MGSVector2 8                           # Specifies 2th position of the vector
```

Tabla 3.2: Archivo de Configuración **layer1** de JSVM

```
# JSVM Layer Configuration File 0 (snr_mgs_layer0.cfg)

InputFile              videoref/bus150.yuv      # Input file
SourceWidth            352                      # Input frame width
SourceHeight           288                      # Input frame height
FrameRateIn            30                       # Input frame rate [Hz]
FrameRateOut           30                       # Output frame rate [Hz]
MGSVectorMode          0                        # MGS vector usage selection
```

Tabla 3.3: Archivo de Configuración **layer0** de JSVM

Una vez definidas las opciones de codificación de SVC, se procede a codificar el video con:

```
H264AVCEncoderLibTestStatic -pf snr_mgs_main.cfg -lqp 0 40 -lqp 1 20
```

En donde las opciones —lqp indican el nivel de cuantización de la capa respectiva.

Ahora que el video ha sido codificado, éste se almacena como mgs_bus.264 que es el bitstream SVC del video original. La consola mostrará los resultados que incluyen el orden temporal de envío de frames (y el tipo) además de información de resumen de bitrate. El ejemplo correspondiente se muestra en la tabla 3.4

```
Profile & Level info:
=====================
DQ= 0:  Main @ Level 2.1
DQ= 1:  Scalable High @ Level 2.1

AU  0: I  T0 L0 Q0  QP 36  Y 30.0878 U 38.4191 V 38.1775    54488 bit
    0: I  T0 L0 Q1  QP 16  Y 51.0685 U 54.9023 V 55.3139   298192 bit
AU  8: I  T0 L0 Q0  QP 36  Y 30.1239 U 38.3431 V 38.9553    56320 bit
    8: I  T0 L0 Q1  QP 16  Y 50.9047 U 54.7691 V 55.1720    295800 bit
AU  4: B  T1 L0 Q0  QP 39  Y 29.1586 U 38.7321 V 38.822       7488 bit
    4: B  T1 L0 Q1  QP 19  Y 46.9734 U 52.9402 V 53.8445    128688 bit
AU  2: B  T2 L0 Q0  QP 41  Y 28.7092 U 38.9818 V 38.9459      4808 bit
    2: B  T2 L0 Q1  QP 21  Y 45.7648 U 52.8738 V 53.8541     97120 bit

(...)

SUMMARY:
                        bitrate    Min-bitr   Y-PSNR   U-PSNR   V-PSNR
                       --------- ---------- -------- -------- --------
   352x288 @  3.7500  1334.3529   220.4357  51.0845  55.1298  55.4057
   352x288 @  7.5000  1922.1600   265.5138  49.1711  54.0720  54.6569
   352x288 @ 15.0000  2748.8592   311.4240  47.4913  53.3634  54.1353
   352x288 @ 30.0000  3919.1902   359.1722  46.0759  52.7866  53.7535

Encoding speed: 1134.367 ms/frame, Time: 55584.000 ms, Frames: 49
```

Tabla 3.4: Información de Codificación del Video

3.2. Generación de *Trace File*

Así, una vez obtenido el bitstream SVC (denominado mgs_bus.264), se genera su *trace file* con el comando:

```
BitStreamExtractorStatic —ptbus150.txt mgs_bus.264
```

El *trace file* es un archivo .txt que contiene información sobre las calidades embebidas en cada *frame*, indica en qué parte del *bitstream* empiezan y terminan, además de si son descartables o no (si son *frames* del tipo base I u otro diferente) así como parámetros de calidad de las capas. Este archivo puede usarse para eliminar la capa de calidad que sea requerida y ver el impacto en la visualización del video modificado, y es con este fin, que se utlizará en el presente trabajo.

Un *trace file* con anotaciones de color es mostrado en la tabla 3.5

Start-Pos.	Length	LId	TId	QId	Packet-Type	Discardable	Truncatable
0x00000000	203	0	0	0	StreamHeader	No	No
0x000000cb	14	0	0	0	ParameterSet	No	No
0x000000d9	16	0	0	0	ParameterSet	No	No
0x000000e9	9	0	0	0	ParameterSet	No	No
0x000000f2	9	0	0	0	ParameterSet	No	No
0x000000fb	9	0	0	0	SliceData	No	No
0x00000104	6811	0	0	0	SliceData	No	No
0x00001b9f	37274	0	0	1	SliceData	Yes	No
0x0000ad39	9	0	0	0	SliceData	No	No
0x0000ad42	7040	0	0	0	SliceData	No	No
0x0000c8c2	36975	0	0	1	SliceData	Yes	No
0x00015931	9	0	1	0	SliceData	Yes	No
0x0001593a	936	0	1	0	SliceData	Yes	No
0x00015ce2	16086	0	1	1	SliceData	Yes	No
0x00019bb8	9	0	2	0	SliceData	Yes	No
0x00019bc1	601	0	2	0	SliceData	Yes	No
0x00019e1a	12140	0	2	1	SliceData	Yes	No
0x0001cd86	8	0	3	0	SliceData	Yes	No
0x0001cd8e	329	0	3	0	SliceData	Yes	No
0x0001ced7	9202	0	3	1	SliceData	Yes	No
0x0001f2c9	8	0	3	0	SliceData	Yes	No

Base - I Enhancement – I Base – B Enhancement - B

Tabla 3.5: *Trace File* coloreado de 2 capas de calidad

Start-Pos nos da el inicio de la capa en el archivo bitsteam mgs_bus.264 generado en formato hexadecimal, Length indica el largo de la capa en bytes y las leyendas Lid, Tid, Qid se corresponden con su significado dependency_id, temporal_id y quality_id. Las primeras 5 líneas del *trace file* son de ajuste de parámetros y son indispensables para la posterior decodificación del video. Por último se incluyen 2 flags que informan si la capa es descartable o si es truncable.

Para facilitar el entendimiento, cada línea del *trace file* ha sido coloreada según el tipo frame al cual corresponda. Como es de esperar, los frames Base-I (en verde) son no-descartables, por lo que sin éstos no podría generarse el resto de los frames (más aún, JSVM no decodificará el video si alguno de éstos llegase a faltar). Esta simbología se usará a lo largo de todo el presente trabajo. También el *trace file* es coherente con la codificación vista en la tabla 3.4 en donde primero se transmiten 2 frames del tipo I (el 0 y el 8) para posteriormente enviarse frames del tipo B. Ahora que el *trace file* ha sido obtenido, se debe proceder con la obtención de un *trace file* similar, pero a nivel de paquetes, no de frames completos. Este nuevo perfil se llamará *packet trace file*.

3.3. Generación de *Packet Trace File*

Para la generación de este nuevo *packet trace file* se desglosa su predecesor a nivel de paquetes con un tamaño de 1460 bytes (máximo permitido para paquetes TCP [10]), para este fin se usará Matlab para procesar el archivo *trace.txt* generado por JSVM. Adicionalmente se añadirá información de control para cada paquete a fin de manipularlos con mayor facilidad. Un ejemplo de este procedimiento se puede visualizar en la tabla 3.6:

#TRACE	LENGTH	DISC.	#TEMP	START	END	DEADL	CWND	V
1	203	0	0	0	250	20,05	0,05	1
2	14	0	0	0	250	20,05	0,05	1
3	16	0	0	0	250	20,05	0,15	1
4	9	0	0	0	250	20,05	0,15	1
5	9	0	0	0	250	20,05	0,15	1
6	9	0	1	251	44344	20,05	0,25	1
7	1460	0	1	251	44344	20,05	0,25	1
7	1460	0	1	251	44344	20,05	0,25	1
7	1460	0	1	251	44344	20,05	0,25	1
7	1460	0	1	251	44344	20,05	0,35	1
7	971	0	1	251	44344	20,05	0,35	1
8	1460	1	1	251	44344	20,05	0,35	1
8	1460	1	1	251	44344	20,05	0,35	1
8	1460	1	1	251	44344	20,05	0,35	1
8	1460	1	1	251	44344	20,05	0,45	1
(...)	(...)	(...)	(...)	(...)	(...)	(...)	(...)	(...)
12	9	1	3	88369	105399	20,183	1,65	1
13	936	1	3	88369	105399	20,183	1,65	1
14	1460	1	3	88369	105399	20,183	1,75	1
14	1460	1	3	88369	105399	20,183	1,75	1

Tabla 3.6: *Packet Trace File* coloreado para 2 capas de calidad

Cada columna tiene los siguientes significados:

- #**Trace**: Indica la fila a la que corresponde el paquete en el *trace file* original.

- **Lenght**: Indica el tamaño del paquete, para este trabajo se toma un máximo de Maximum Segment Size de 1460 bytes.

- **DISC**: Indica si el paquete es descartable o no, dependiendo de si es un paquete perteneciente a un I-*frame* o B-*frame*.

- **#Temp**: Indica el orden temporal de transmisión de paquete, que no es el mismo de display en pantalla, así por ejemplo:

  Temp 1 → Display 0
  Temp 2 → Display 8
  Temp 3 → Display 4
  Temp 4 → Display 2

  Ver tabla 3.4 para un mejor entendimiento.

- **Start**: Indica la posición de inicio de la capa a la cual pertenece el paquete en bytes.

- **End**: Indica la posición de término de la capa a la cual pertenece el paquete en bytes.

- **DeadL**: Indica el límite temporal (en segundos) dentro del cual el paquete debe llegar a destino antes de que sea necesario mostrar el frame correspondiente en pantalla.

  Este límite es el *deadline* de visualización y viene según la calidad temporal del video, es decir, si se requiere mostrar un video a 30 fps, entonces cada frame debe mostrarse a 1/30 [s] de su predecesor, por lo cual es fundamental que todos los paquetes que corresponden al siguiente frame lleguen antes de este límite temporal, de otra forma mermarán la fluidez del video.

  Como medida adicional se suele incluir un tiempo de *buffer* antes de reproducir el video, a fin de que se tenga cierto respaldo temporal para las posibles pérdidas que puedan ocurrir en la transmisión. Por ahora el *buffer* está fijado en 20 s, un valor bastante grande para cualquier usuario, pero en la simulaciones éste valor es reducido a 3 segundos.

  Este deadline no toma en cuenta los tiempos de decodificación en el receptor, ya que se asumen despreciables respecto del tiempo de transmisión por red.

- **CWND**: Es la Ventana de Congestión (ver sección 2.2) expresada en segundos. A modo de ejemplo las primeras 2 filas de la tabla 3.6 llegan en un tiempo de 0.05 segundos, luego la ventana de congestión se agranda en 1, por lo que ahora se envían las 3 siguientes filas simultáneamente que llegan en 0.15 segundos, para luego volver a agrandarse la ventana en una unidad más. El round trip time está fijo en rtt = 0.1 s pero será un parámetro de evaluación en las siguientes secciones.

Cabe destacar que la CWND mostrada en la 3.6 no sufre ninguna pérdida, por lo que su incremento siempre es lineal, situación que posteriormente se modificará al pasar por un canal de pérdidas. En el caso de que se produzcan pérdidas, entonces la actual ventana de congestión se reducirá a la mitad.

- **V**: Es el dígito verificador, su valor es 1 cuando CWND <DEADL, y 0 en caso contrario. Es decir este dígito indica si los paquetes llegaron a tiempo o no.

De esta forma los únicos elementos requeridos para generar el *packet trace file* , es el *round trip time*, *buffer time* y el *trace file* original.

Una vez que el packet *trace file* ha sido obtenido, ahora se debe transmitir la información por un canal con pérdidas.

3.4. Transmisión y Recuperación de Paquetes

Para la transmisión de video se genera una distribución uniforme de pérdidas en el intervalo de duración del video. El *loss rate* es un parámetro de evaluación que será usado en las siguientes secciones.

La transmisión de información a través de TCP se realiza a través de ventanas de congestión (ver sección 2.2) para transmitir cierta cantidad de paquetes de forma simultánea. Estas ventanas en presencia de pérdidas disminuyen su tamanõ de forma de no ser agresivas con el resto de paquetes que se envían a través de un canal común.

Un ejemplo de una ventana que ya ha pasado por la etapa de *slow start*, y se encuentra en el estado de *congestion avoidance* se muestra en la figura 3.2

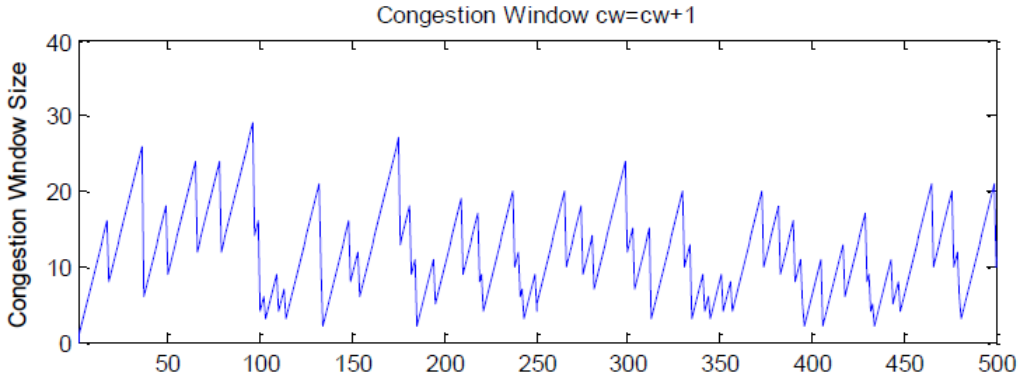

Figura 3.2: Ventana de Congestión Simulada

De esta forma, la ventana de congestión adquiere una forma conocida cómo diente de sierra, en donde las caídas corresponden a repentinas pérdidas y las subidas corresponden a intervalos de tiempo en que no ocurren pérdidas.

En TCP los paquetes siempre se intentan recuperar, de aquí se desprende que el protocolo es confiable para la transmisión de archivos. Pero SVC no requiere que todos los paquetes sean recibidos para poder reconstruir el video, es esta la característica que se explotará en el presente trabajo.

3.4.1. Recuperación paquetes base-I

El envío total de un frame base-I generalmente necesita de 6 u 7 paquetes, si alguno de estos se perdió durante la transmisión, entonces este será reenviado en la ventana de congestión subsiguiente al envío de los frames que corresponden al frame base-I, tal como se muestra en la tabla 3.7:

#TRACE	LENGTH	DISC	#TEMP	START	END	DEADL	CWND	V	REC
33	9	0	10	168025	212086	20,583	5,15	1	141
34	1460	0	10	168025	212086	20,583	5,15	1	0
34	1460	0	10	168025	212086	20,583	5,25	1	0
34	1460	0	10	168025	212086	20,583	5,25	1	144
34	1460	0	10	168025	212086	20,583	5,35	1	0
34	1460	0	10	168025	212086	20,583	5,35	1	0
34	53	0	10	168025	212086	20,583	5,45	1	147
35	1460	1	10	168025	212086	20,583	5,45	1	0
33	9	0	10	168025	212086	20,583	5,55	1	141
34	1460	0	10	168025	212086	20,583	5,55	1	144
34	53	0	10	168025	212086	20,583	5,65	1	147
35	1460	1	10	168025	212086	20,583	5,65	1	0
35	1460	1	10	168025	212086	20,583	5,75	1	0

Pérdida Recuperación

Tabla 3.7: Packet Trace File con pérdidas y Recuperación base-I

Esta vez adicionalmente se introdujo una nueva columna REC (recuperación) que indica el momento de la pérdida y la posterior recuperación. El número en su interior no es más que el índice global de la matriz, el número no es importante, sólo su correspondencia con la recuperación.

La recuperación ocurre a partir del segundo cuadro enmarcado en la columna CWND, observar que la recuperación ocurre en el primer cambio de ventana de congestión luego de terminar de enviar el frame base-I. También es posible apreciar la interrupción del envío de la capa de enhancement-I (amarillo) para la recuperación de paquetes base I (en verde).

3.4.2. Recuperación paquetes base-B

Esta vez, el frame base-B requiere sólo 2 paquetes para su transmisión. La recuperación se efectúa en la ventana de congestión inmediatamente después de la pérdida. Un ejemplo se presenta en la tabla 3.8

#TRACE	LENGTH	DISC	#TEMP	START	END	DEADL	CWND	V	REC
92	218	1	29	489302	498635	20,88333	10,95	1	0
93	8	1	30	498636	507989	20,95	10,95	1	0
94	354	1	30	498636	507989	20,95	10,95	1	414
95	1460	1	30	498636	507989	20,95	10,95	1	0
94	354	1	30	498636	507989	20,95	11,05	1	414
95	1460	1	30	498636	507989	20,95	11,05	1	0
95	1460	1	30	498636	507989	20,95	11,15	1	0

Pérdida Recuperación

Tabla 3.8: Packet Trace File con pérdidas y Recuperación base-B

Cómo se puede apreciar, la recuperación ocurre casi inmediatamente una vez perdido el paquete base-B, el paquete (al igual que en la recuperación base-I) se reenvía al inicio de la siguiente ventana de congestión. Adicionalmente se puede visualizar la disminución de la ventana de congestión al momento de la pérdida, de 4 paquetes a 2 paquetes. La recuperación de paquetes de calidad o enhancement es análoga a la recuperación de paquetes de base B.

3.5. Algoritmo de Deadline

El algoritmo de deadline tiene como objetivo evitar que el usuario perciba un retraso en la transmisión del video, es decir, mantener la tasa de 30 *frames* por segundo en todo momento. Para lograr esto, se usa el hecho de que el decodificador SVC no requiere que todos los paquetes lleguen a destino para mostrar el video en pantalla. De esta forma, se puede optar por omitir ciertos paquetes cuando se esté cerca del límite de visualización (o deadline, simplemente). Más aún, se pueden fijar distintos niveles de aproximación al deadline, ya que se dispone de 4 tipos de paquetes diferentes.

Entonces, si se fija la separación de los intervalos, por ejemplo cada 5 RTT, se obtiene el esquema mostrado en la figura 3.3:

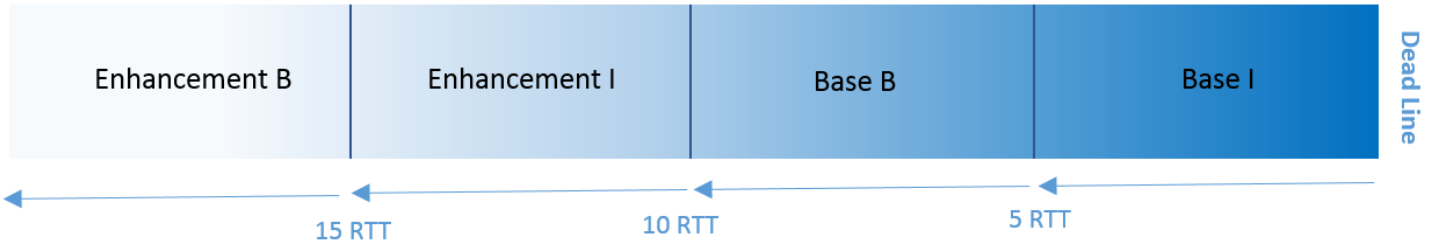

Figura 3.3: Niveles de Deadlines

Desde la perspectiva del emisor, si los paquetes próximos a enviar llegaran antes que el primer deadline (se puede estimar este tiempo de acuerdo al RTT), entonces de todos los paquetes a enviar, sólo se enviarán aquellos que corresponden a los paquetes base I, ya que se está por debajo del primer límite de 5 RTT. De igual forma, si los próximos paquetes a enviar llegaran a 12 RTT del deadline, entonces se omiten los paquetes de enhancement B y se envían enhancement I, base B y base I.

Con propósitos de clarificación, se presenta un ejemplo en la figura 3.4 en la cual todos los paquetes de la ventana de congestión (de 7 unidades) llegarán en menos de 10 RTT del deadline de visualización. Al ocurrir esto, se eliminan sucesivamente los paquetes de enhancement I o B (ya que ambos corresponden a un tiempo menor a 10 RTT del deadline) hasta tener una ventana repleta sólo de paquetes de base I o B, que corresponden a un intervalo menor a 10 RTT del deadline.

Es de importancia señalar que la siguiente ventana de paquetes a enviar no se reduce a la mitad, ya que no hubo una pérdida que indique congestión, sino que hubo una omisión voluntaria de parte del emisor. Así, la nueva ventana tendrá un tamaño de 8 unidades y tratará nuevamente de enviar todos los paquetes en cola (paquetes 15,16,17 etc.), independiente de si son del tipo base o enhancement. Finalmente la nueva ventana se somete al proceso del algoritmo de deadline explicado para reiniciar el ciclo.

Con este tipo de implementación, se usa el espacio en tiempo reservado para paquetes enhancement para enviar paquetes del tipo base (por ejemplo), garantizando de esta forma la visualización fluida por parte del usuario (a expensas de calidad, claro está).

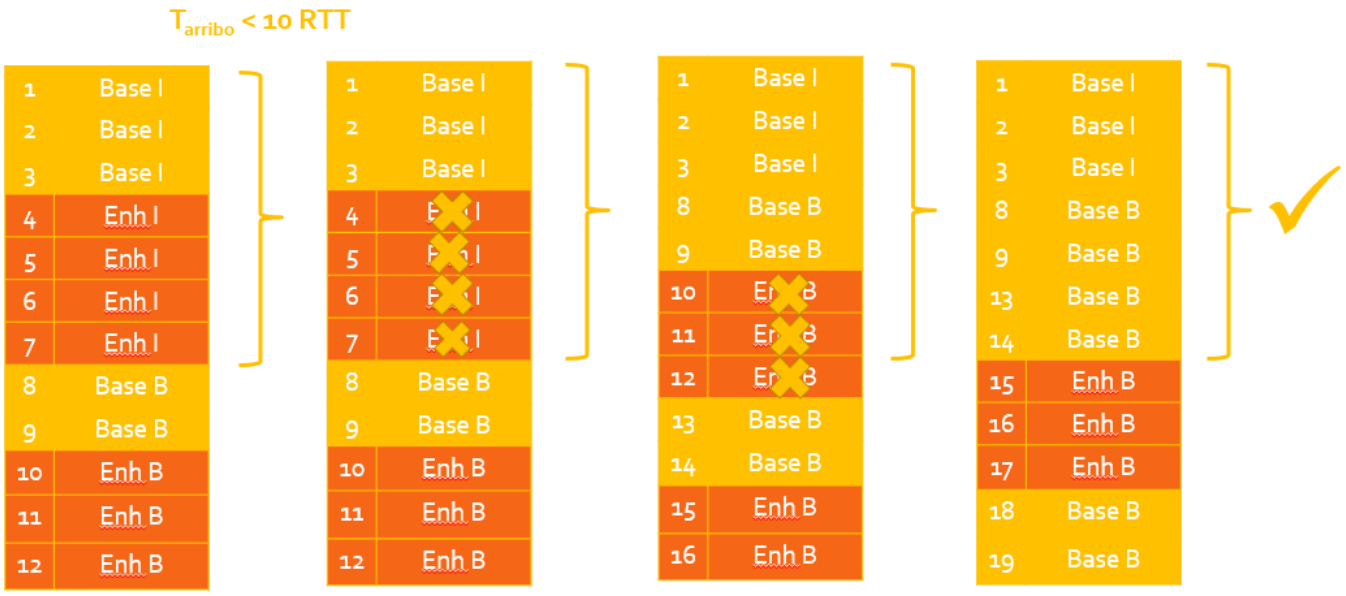

Figura 3.4: Algoritmo de Deadline

Una de las ventajas del algoritmo es que permite una resolución de control a nivel de la ventana de congestión, a diferencia de DASH, el cual ejecuta cambios de calidad en tiempos del segmento (entre 2 y 10 segundos generalmente), quedando bastante grande respecto a los cambios en la ventana de congestión, que son del orden de los milisegundos.

Por último, es vital la separación de los intervalos de aproximación al deadline (ver figura 3.3). Si la separación es muy pequeña y densa, el algoritmo de deadline no alcanzará a reaccionar a tiempo para eliminar capas de calidad antes que eliminar capas base, resultando en la eliminación de frames completos. Por el contrario, si la separación es muy grande, entonces el algoritmo actúa muy anticipadamente, eliminando calidad cuando aún existía tiempo para enviar los paquetes asociados a ésta, y por tanto, desperdiciando recursos. Esta separación, es propuesta como una de las aristas para un trabajo futuro en la sección 5.2.

Las separaciones de 5,10 y 15 RTT funcionan bastante bien en los resultados finales, por lo que se mantendrán a lo largo del presente trabajo.

3.6. Reconstrucción del Video

Para la reconstrucción del video, una vez que el algoritmo de deadline ha sido aplicado sobre los paquetes enviados en Matlab, se debe usar el *trace file* de JSVM para eliminar las filas correspondientes en el archivo de texto. La correspondencia del *packet trace file* con el *trace file* está dada por la columna #**Trace** (ver sección 3.3), de esta forma es fácil saber qué paquetes corresponden a qué fila del *trace file* de JSVM.

3.6.1. Eliminación de Calidad

La eliminación de calidad es bastante fácil en JSVM, basta con eliminar la fila correspondiente al segmento de calidad en el archivo de texto del *trace file*. Para mayor clarificación se muestra un ejemplo en la tabla 3.9

Start-Pos.	Length	LId	TId	QId	Packet-Type	Discardable	Truncatable
0x00000000	203	0	0	0	StreamHeader	No	No
0x000000cb	14	0	0	0	ParameterSet	No	No
0x000000d9	16	0	0	0	ParameterSet	No	No
0x000000e9	9	0	0	0	ParameterSet	No	No
0x000000f2	9	0	0	0	ParameterSet	No	No
0x000000fb	9	0	0	0	SliceData	No	No
0x00000104	6811	0	0	0	SliceData	No	No
0x00001b9f	37274	0	0	1	SliceData	Yes	No
0x0000ad39	9	0	0	0	SliceData	No	No
0x0000ad42	7040	0	0	0	SliceData	No	No
0x0000c8c2	36975	0	0	1	SliceData	Yes	No
0x00015931	9	0	1	0	SliceData	Yes	No
0x0001593a	936	0	1	0	SliceData	Yes	No
0x00015ce2	16006	0	1	1	SliceData	Yes	No
0x00019bb8	9	0	2	0	SliceData	Yes	No
0x00019bc1	601	0	2	0	SliceData	Yes	No
0x00019e1a	12140	0	2	1	SliceData	Yes	No
0x0001cd86	8	0	3	0	SliceData	Yes	No
0x0001cd8e	329	0	3	0	SliceData	Yes	No
0x0001ced7	9202	0	3	1	SliceData	Yes	No
0x0001f2c9	8	0	3	0	SliceData	Yes	No

Base - I Enhancement – I Base – B Enhancement - B

Tabla 3.9: Eliminación de Calidad

Así, una vez se obtenga el *trace file* con las eliminaciones de calidad (por ahora trace_test.txt), se debe reconstruir el video con el comando en la *cmd*:

```
BitStreamExtractorStatic mgs_bus.264 mezclaMGS2.264 -et trace_test.txt
```

En el cual mgs_bus.264 corresponde al bitstream del video original y mezclaMGS2.264 al bitstream final.

3.6.2. Eliminación de Base

La eliminación de información base de JSVM puede realizarse de forma análoga a la eliminación de calidad, pero al efectuarlo de esta forma los frames son eliminados por completo y no aparecen en el video, complicando la tarea de usar PSNR por frame reconstruido. En otras palabras, si se omite tan sólo 1 paquete de calidad base, entonces se tendrán 149 frames en vez de los 150 originales.

Para abordar este problema se crean frames artificiales llenados con ceros. El uso de estos frames en blanco tiene la ventaja de que pueden observarse los artefactos producidos en los frames vecinos por su dependencia al frame omitido (ver sección 2.5).

El reemplazo del frame puede efectuase en Matlab tomando cada imagen como una matriz de 352x288 para cada componente YUV, es decir, cada imagen es una matriz de 352x288x3 que debe ser llenada completamente de ceros. Una vez que el video modificado es obtenido, se debe codificar en un bitstream para su posterior manipulación.

Para resumir, los pasos a seguir son:

- Llenar con ceros los frames omitidos y crear un nuevo video con frames en blanco, para así crear el bitstream *mgs_bus_blank.264* y su respectivo *trace file bus150_blank.txt* a partir de éste.
- Se usa la codificación original *mgs_bus.264* y su contraparte *mgs_bus_blank.264* para mezclarlos en la codificación *mezclaMGS.264*, como se muestra en la figura 3.5:

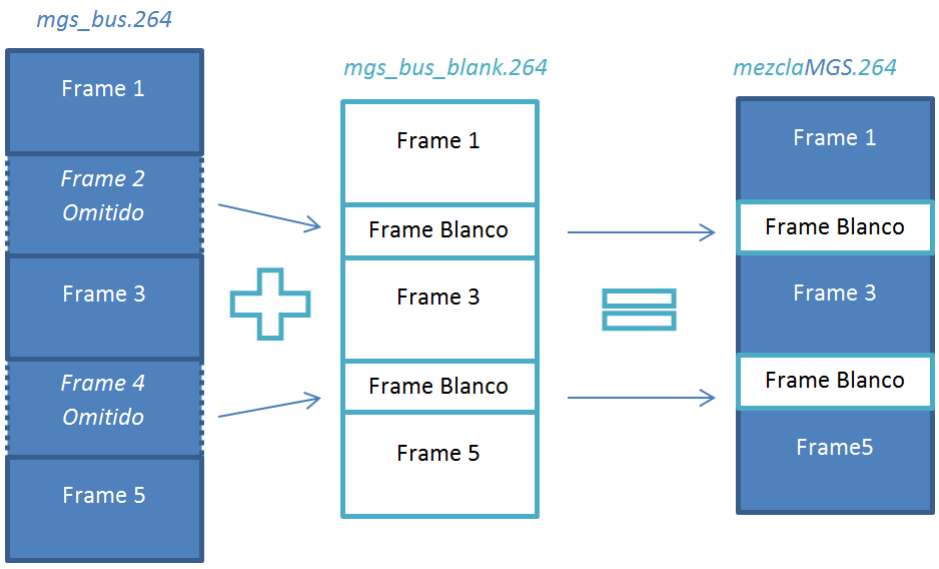

Figura 3.5: Eliminación de Calidad

Para conocer en qué lugares del bitstream hacer el reemplazo de los datos, se recurre a las columnas **Start** y **End** del *packet trace file*.

Se puede visualizar el video en busRECONSTRUIDO.yuv si se usa:

```
H264AVCDecoderLibTestStatic mezclaMGS.264 busRECONSTRUIDO.yuv
```

- Ahora, falta agregar las pérdidas de calidad, para lo cual se usa el procedimiento explicado en la sección 3.6.1 sobre el *trace file* del bitstream *mezclaMGS.264*.

  Para saber qué filas eliminar, se debe usar la columna #**TEMP** del *packet trace file* original que facilita la correspondencia entre paquetes de calidad omitidos originalmente y el orden temporal de envío de los frames, para así identificarlos en el nuevo *trace file* de la *mezclaMGS.264*.
- Por último, se eliminan las filas correspondientes a calidad en el archivo *trace_test.txt*, y se obtiene el bitstream final *mezclaMGS2.264* con:

```
BitStreamExtractorStatic mezclaMGS.264 mezclaMGS2.264 trace_test.txt
```

mezclaMGS2.264 se usa para reconstruir el video final con:

```
H264AVCDecoderLibTestStatic mezclaMGS2.264 bus_final.yuv
```

bus_final.yuv contiene el video con todas las pérdidas de base, artefactos en frames vecinos y pérdidas de calidad que serán explicadas en la siguiente sección.

3.6.3. Percepción de Pérdidas

Como se dijo anteriormente, existen 3 tipos de percepción de las pérdidas y cada una amerita una explicación por separado. Estas pérdidas se denominarán:

- Pérdida de Base
- Pérdida Parcial
- Pérdida de Calidad

El tipo de percepción está fuertemente ligado a la dependencia entre frames del GOP. Para facilitar el entendimiento se usará como referencia la figura 3.6, que complementa lo ya explicado sobre GOP en la sección 2.5.

Para ejemplificar la dependencia, se puede ver en la figura 3.6 que el frame 3 depende de la información contenida en 2 y 4 para su correcta decodificación. También puede apreciarse el hecho de que 0 y 8 no tienen dependencia alguna ya que corrresponden a I-frames, pero el resto tiene dependencia bi-direccional y por tanto son B-frames.

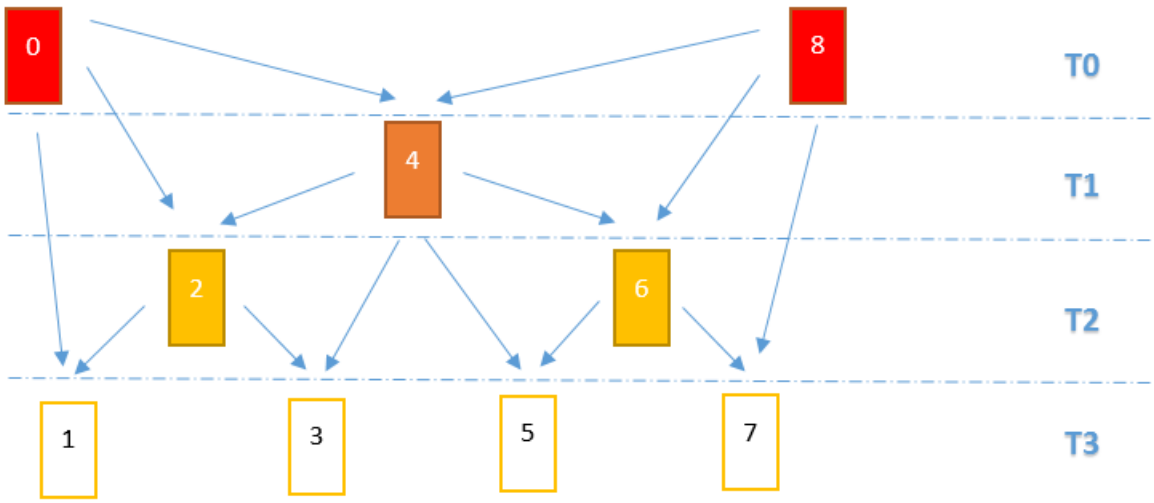

Figura 3.6: Dependencia de Frames

Los niveles temporales T0,T1,T2 Y T3 son consecuencia del hecho de que si se mostrasen, por ejemplo, todos los niveles hasta T3, se obtendría una tasa de 30fps. Si se mostrasen hasta T2, de 15 fps; hasta T1, de 7.5 fps; y hasta T0, de 3.25 fps. Como observación adicional se hace notar el hecho de que un 50 % del GOP está constituido de frames del nivel T3, y por tanto la mitad del video está compuesto de estos frames.

A continuación se presentan las causas que llevan a las distintas percepciones de pérdidas, así como ejemplos de éstas. Es importante tomar la figura 3.6 cada vez que se hable de los frames.

Pérdida de Base

Corresponde a una pérdida de base en cualquier nivel temporal y tiene como implicancia la total ausencia del frame en cuestión, además de la posible repercusión en forma de artefactos a los frames que dependen de él.

La pérdida total se ilustra en la figura 3.7

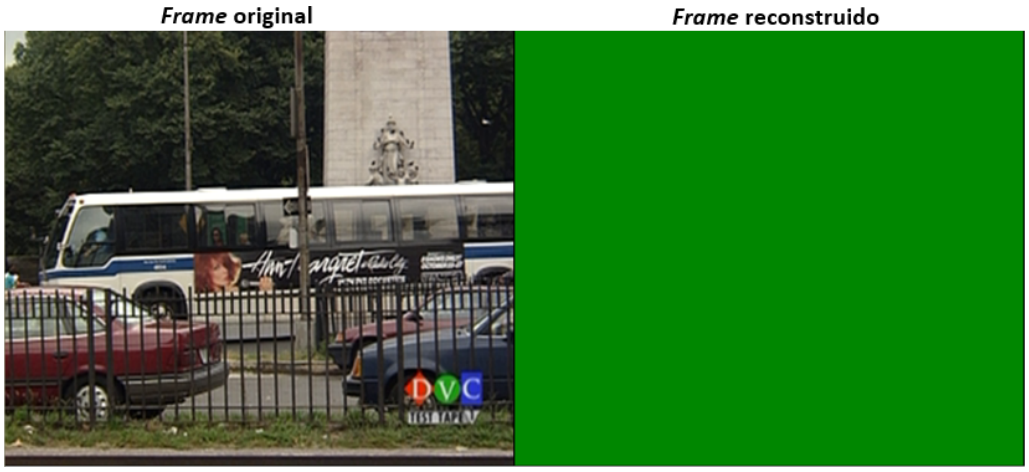

Figura 3.7: Pérdida de Base

Pérdida Parcial

Corresponde a un frame afectado por la omisión de un frame del cual era directamente dependiente, por ejemplo la pérdida de 2 y el correcto arribo de 4 conllevaría una pérdida parcial en 3.

Un ejemplo de esto se aprecia en la figura 3.8:

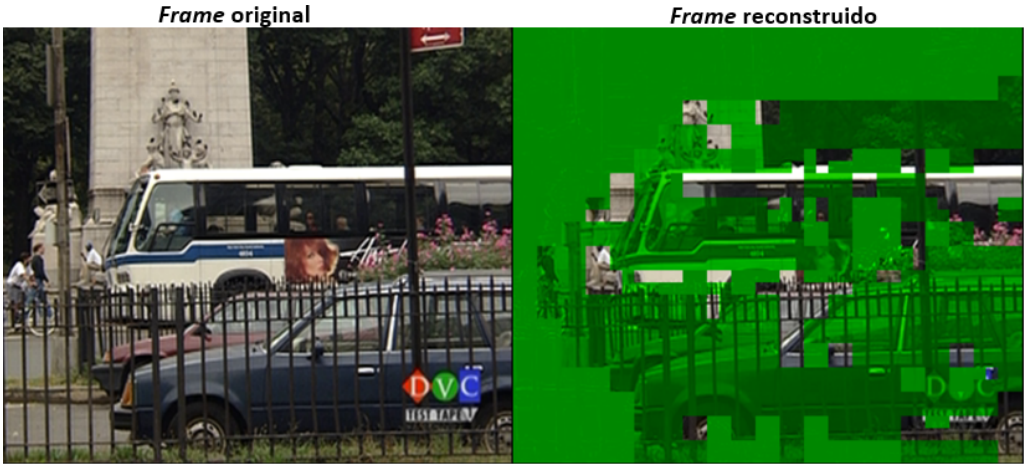

Figura 3.8: Pérdida Parcial Directa

Es importante notar que mientras más jerarquía posea una pérdida de base, más efecto tendrá sobre el resto del GOP. Por ejemplo, si se perdiese el frame 0, todo el GOP resultaría con artefactos, ya que todos los frames dependen de 0. Por otro lado, si se perdiese 2, entonces sólo se visualizarían artefactos en 1 y 3 (además de la pérdida total de 2). Además, mientras menos directa sea la dependencia de una pérdida de base, menos se verán afectados los frames involucrados. Por ejemplo, la pérdida de 0 afecta notablemente a 1,2 y 4 ya que poseen dependencia directa de él, pero los frames 6 y 7 no son tan severamente afectados, ya que la información de 4 y 8 ha llegado correctamente. Este contraste es evidenciado en las figuras 3.8 (artefactos en 1) y 3.9 (artefactos en 7).

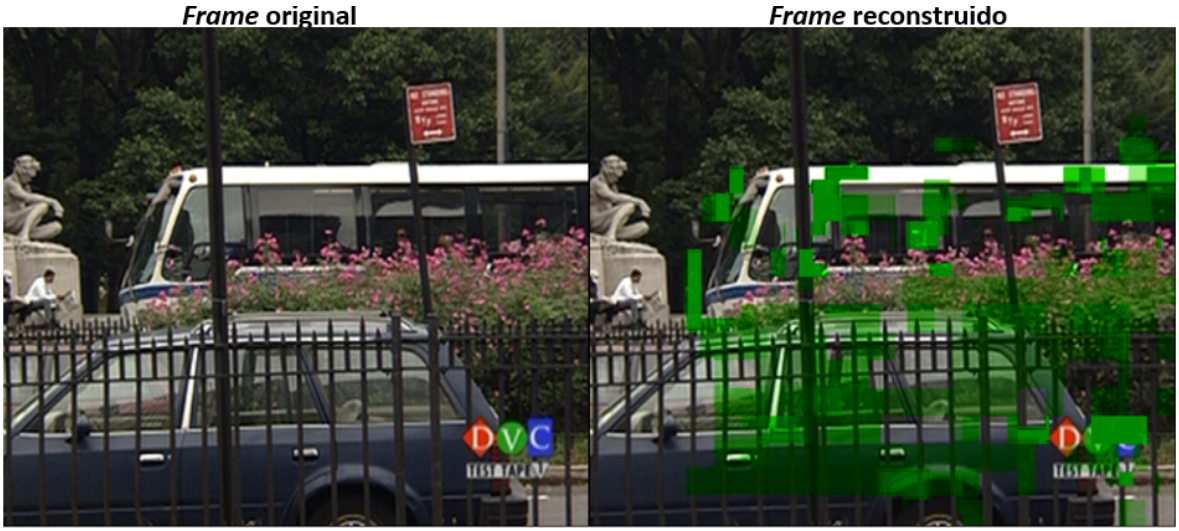

Figura 3.9: Pérdida Parcial Indirecta

Pérdida de Calidad

Corresponde al hecho de que exista sólo pérdida de calidad de alguno de los frames. Un ejemplo de esta percepción puede verse en la figura 3.10, en donde se ha realizado un acercamiento al video además de la inclusión del PSNR por pixel para ambas imágenes. El PSNR indica el nivel de la reconstrucción, más azul indica mala reconstrucción y más rojo indica lo contrario.

El hecho de que un frame pierda calidad también tiene implicancia para los frames dependientes. La pérdida de calidad de un I-frame es la más grave de todas, ya que repercute sobre todo el GOP (al igual que en el caso de pérdida de base). Las pérdidas parciales y las pérdidas de calidad pueden afectar simultáneamente a un frame, es decir, se puede tener pérdida de calidad y artefactos al mismo tiempo.

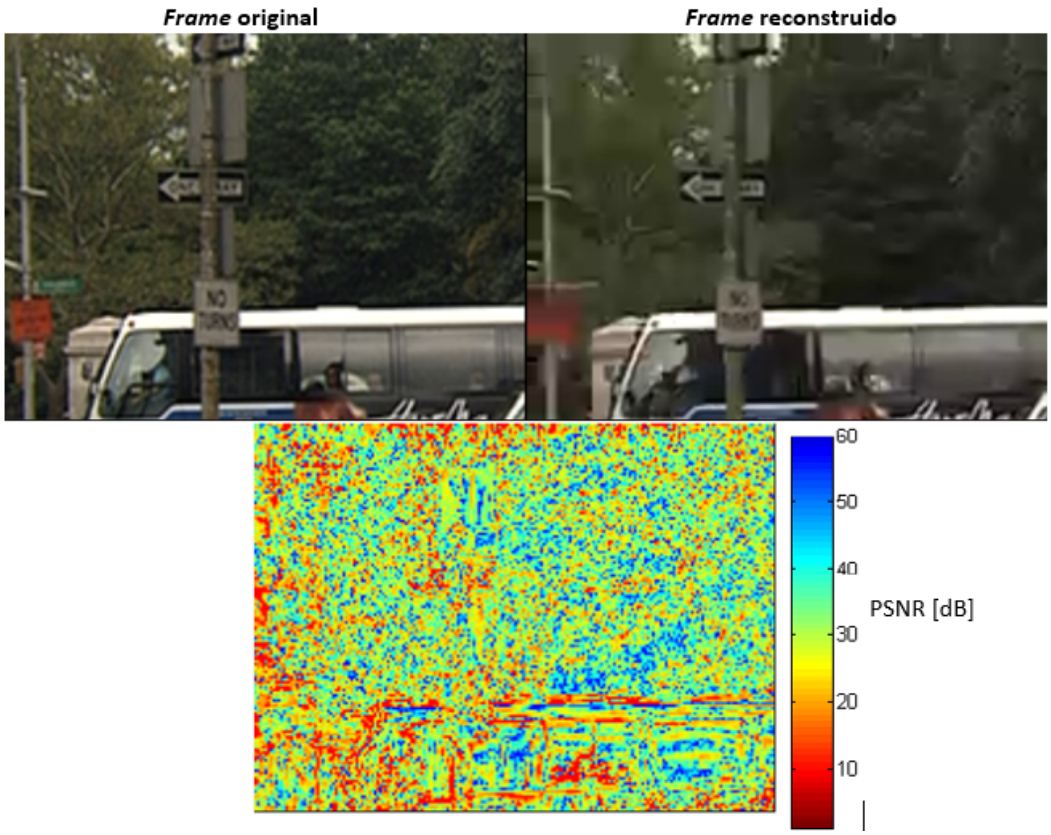

Figura 3.10: Pérdida de Calidad

La visualización de las figuras 3.7, 3.8, 3.9 y 3.10, además del computo del PSNR por frame, es realizado gracias al software de reproducción y análisis de videos YUV, **YuvToolKit** [14] de libre acceso y código *open source*.

Capítulo 4

Resultados

Para mostrar la mejora en la fluidez del video del nuevo algoritmo, se procederá a contrastar la propuesta con un envío tradicional de TCP bajo diferentes condiciones de RTT y un *loss rate* fijo. La razón de dejar un *loss rate* fijo se explica en el hecho de que éste valor es muy variable dependiendo del destino/origen, horario, condiciones del servidor y otros factores, además un incremento de *loss rate* equivale (en términos del tiempo de llegada) a un incremento en el RTT. Estas figuras serán acompañadas de la ventana de congestión correspondiente y una representación PSNR de la componente Y del video (Y-PSNR).

El cómputo de PSNR es logrado gracias al software de video **MSU Video Quality Measurement Tool** [13], del *MSU Video Team* de Rusia. El software es de distribución gratuita en su versión estándard y ofrece distintas métricas para evaluar la reconstrucción de video.

Los escenarios a evaluar son 3:

- Nacional, RTT de 50ms, *loss rate* de 0.01
- Continental, RTT de 150ms, *loss rate* de 0.01
- Transcontinental, RTT de 300ms, *loss rate* de 0.01

4.1. Escenario Nacional

Considerando Chile, se puede asumir razonablemente un RTT cercano a los 50 ms, en el cual se visualiza el comportamiento del algoritmo en la figura 4.1. La línea roja continua indica el deadline de visualización de usuario a 30fps, y comienza a partir de los 3s ya que éste es el *buffer time* usado para toda la simulación. La línea roja punteada corresponde al 3er deadline de visualización (ver figura 3.3). La línea azul corresponde a un envío TCP de la información, mientras que la línea magenta es el envío con el algoritmo de deadline implementado.

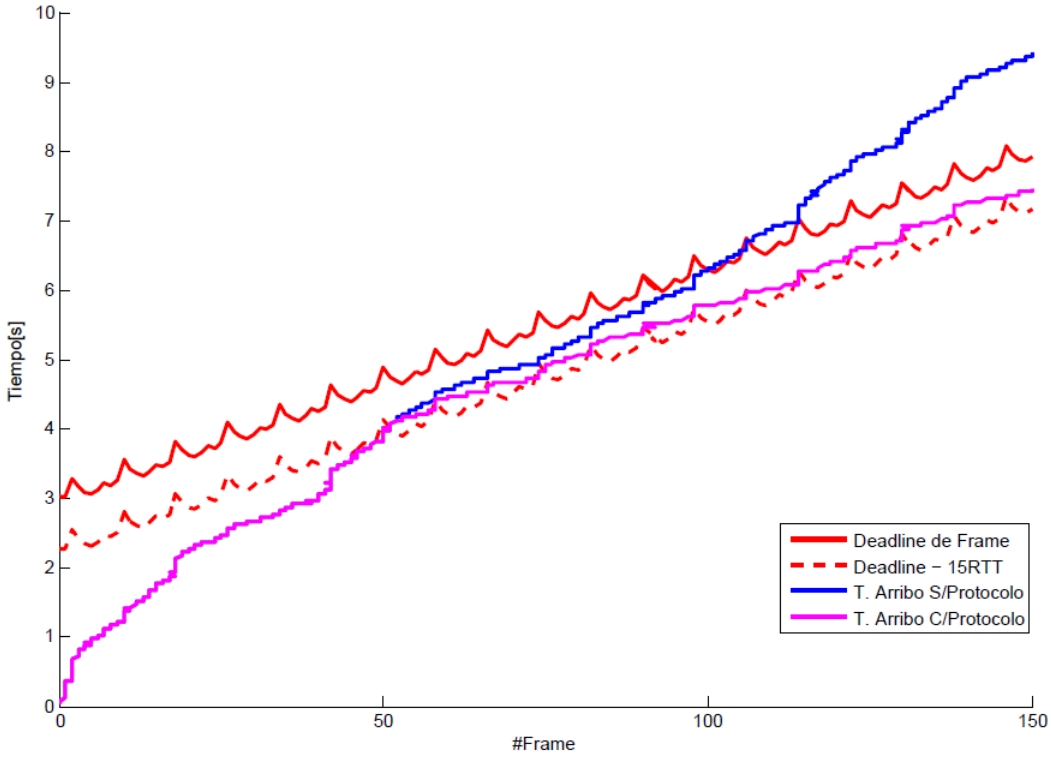

Figura 4.1: Tiempos de Transmisión de Video por Frame, Loss rate: 0.01, RTT: 0.05s

Se puede ver que aproximadamente en el frame #50 el deadline de seguridad de 15 RTT es alcanzado, y en consecuencia el protocolo comienza a actuar según el algoritmo de deadline (ver sección 3.5), y por tanto las curvas de transmisión comienzan a separarse hasta tener una diferencia de 2.5 segundos al final del video en los 150 frames. Otro factor que llama la atención es el hecho de que el nuevo protocolo, al omitir información, se mantiene fiel al deadline de 15 RTT, evitando así mayor proximidad con el deadline de frame y la omisión de paquetes del tipo base. Dicho esto, es evidente que el nuevo protocolo es efectivo a la hora de prevenir la interrupción de fluidez en el video. Sin embargo, la fluidez del video del algoritmo tiene un costo de calidad asociado, costo que se ve reflejado en la figura 4.2

La curva roja de la figura 4.2.a es la medida de Y-PSNR con respecto al video original, mientras que los triángulos de colores indican el tipo de omisión que efectuó el algoritmo de deadline. Es de observarse que, en concordancia con la figura 4.1 a partir de los 50 frames, el

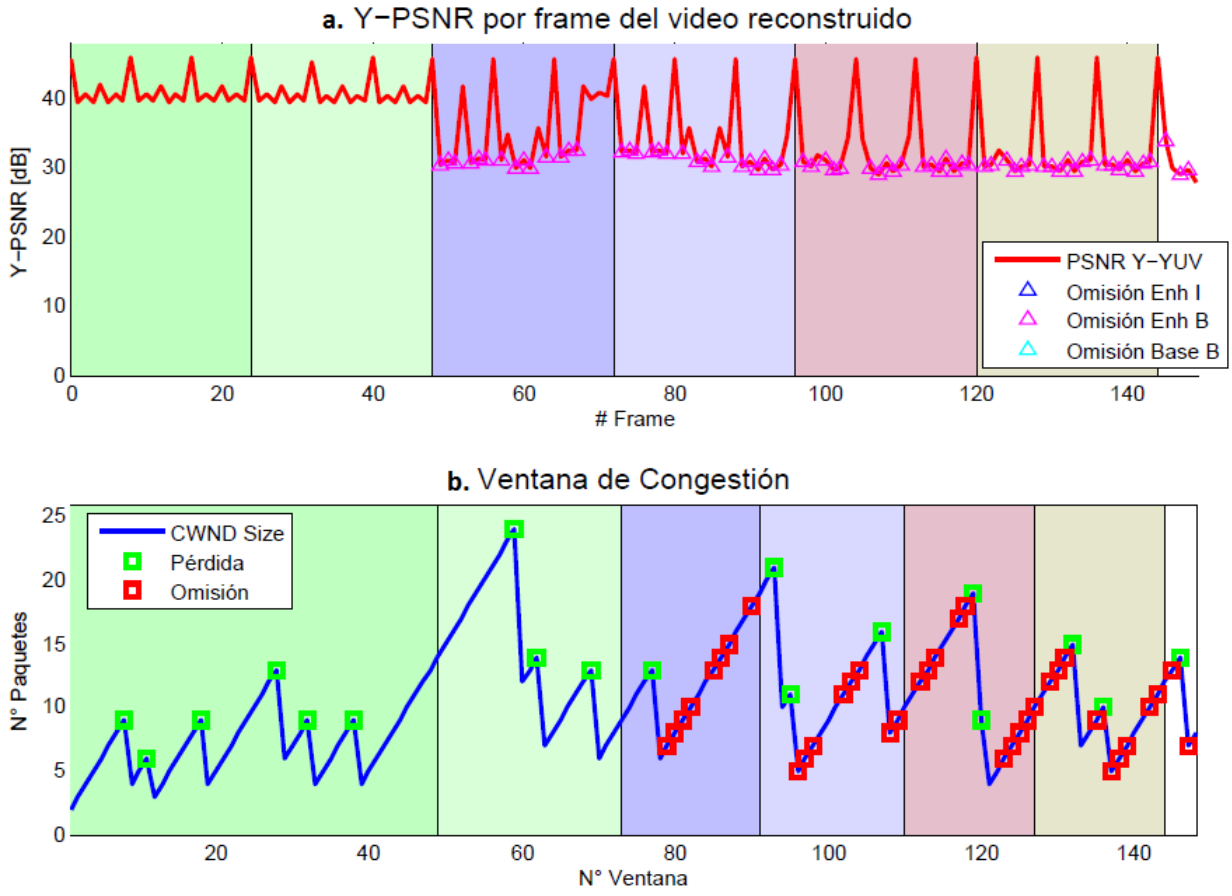

Figura 4.2: PSNR y Ventana de Congestión Escenario Nacional

algoritmo omite calidad, y esto se refleja en la curva de Y-PSNR a partir del frame mencionado. Sin embargo, sólo la omisión de información de enhancement B es omitida, cediendo espacio en el tiempo para que lleguen paquetes más importantes, como los de enhancement I o base I (estos últimos se reflejan en los picos de la curva Y-PSNR).

La curva azul de la figura 4.2.b es el tamaño de la ventana de congestión conforme avanza la transmisión del video. Los cuadrados verdes indican pérdida (y la reducción de la ventana a la mitad) y los cuadrados rojos omisión de paquetes. Las sombras detrás de las curvas muestran la correspondencia entre la cantidad de frames enviados y el número de ventanas de congestión que fueron necesarias para tal propósito. Por ejemplo, los 24 primeros frames (o 3 GOPs) se enviaron luego de que 49 ventanas de congestión fueran computadas. La separación de 3 GOPs por color se mantiene a lo largo de todo el resto del trabajo.

Observando que en la figura 4.2.a sólo hay pérdidas de enhancement B, se infiere el video estará repleto de percepciones de pérdida del tipo de calidad (ver sección 3.6.3). Una muestra del video reconstruido se muestra en la figura 4.3

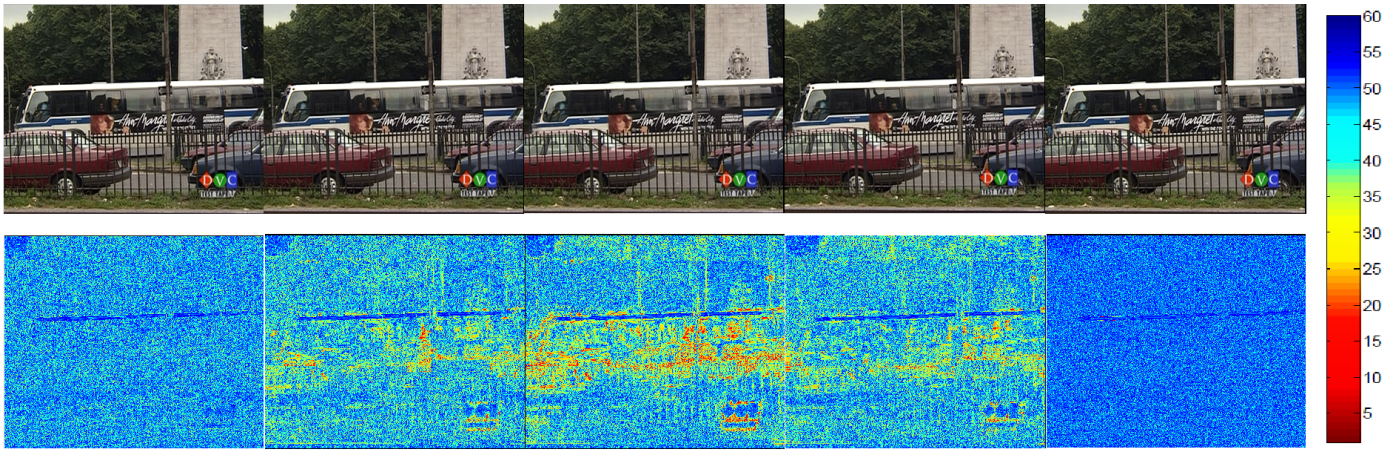

Figura 4.3: Reconstrucción de video, frames del 77 al 81

Se distinguen los frames que tienen falta de *enhancement B* y aquellos que no por la diferencia en los valores de PSNR. Además los valores más negativos de PSNR corresponden la parte más dinámica del video, que en este caso corresponde al bus que se desplaza, fenómeno propio de la naturaleza predictiva de H.264.

4.2. Escenario Continental

Para el continente de américa, sería prudente considerar un tiempo RTT de 150 ms para las simulaciones.

Dicho esto, se puede ver el comportamiento del algoritmo en la figura 4.4

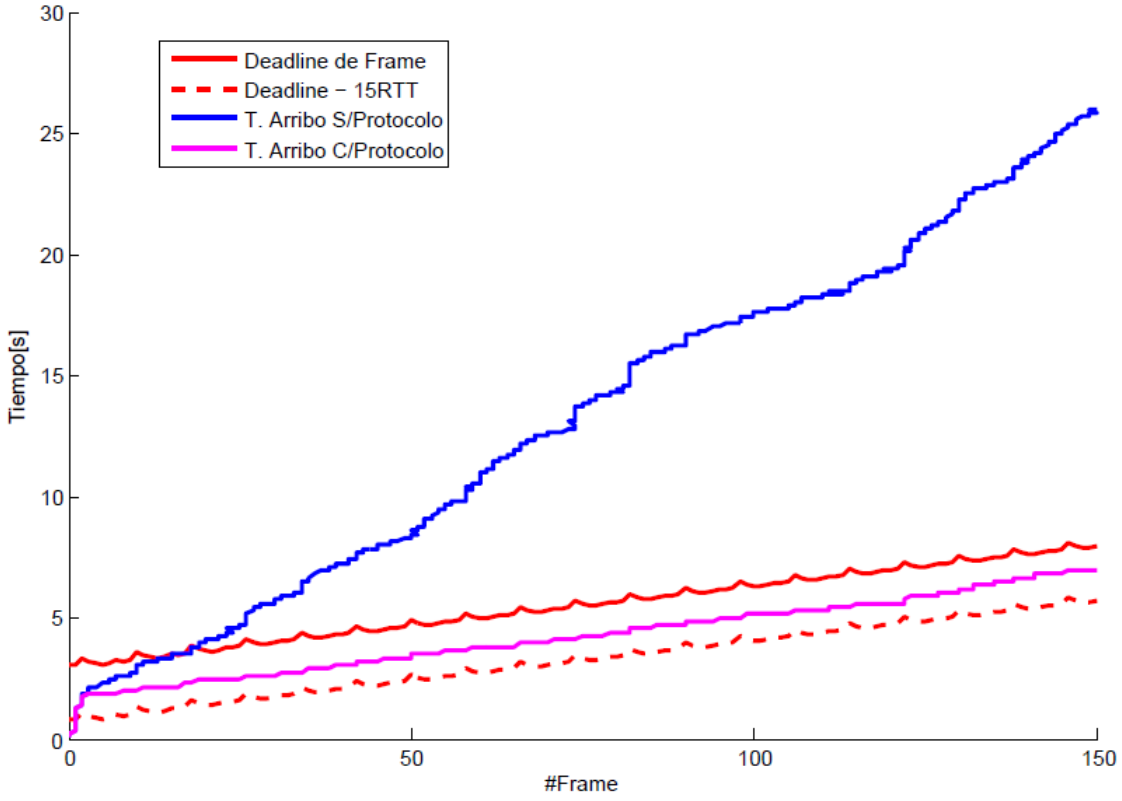

Figura 4.4: Tiempos de Transmisión de Video por Frame, Loss rate: 0.01, RTT: 0.15s

Esta vez a diferencia del escenario nacional, el deadline de 15 RTT es alcanzado muy tempranamente, provocando una separación de las curvas de transmisión alrededor del frame #5 y una diferencia abismante 20 segundos hacia el final del video, valor muy por sobre los 2.5s del escenario nacional.

Cabe destacar que esta vez el nuevo protocolo se mantiene ligeramente alejado del deadline de 15 RTT, en contraste con el escenario nacional, en donde la proximidad era mayor. Este evento desencadena que el protocolo sobrepase el 2do deadline de 10 RTT (no mostrado en la figura, por claridad) y por tanto se empiecen a descartar paquetes del tipo *enhancement I*.

La pérdida de calidad asociada a la omisión de paquetes del escenario continental se puede observar en la figura 4.5.a. Esta vez no toda la información de *enhancement I* logra ser enviada (esto solamente es logrado en el frame #0), provocando una severa degradación en los frames I y B (efecto visualizado en la ausencia de picos del Y-PSNR).

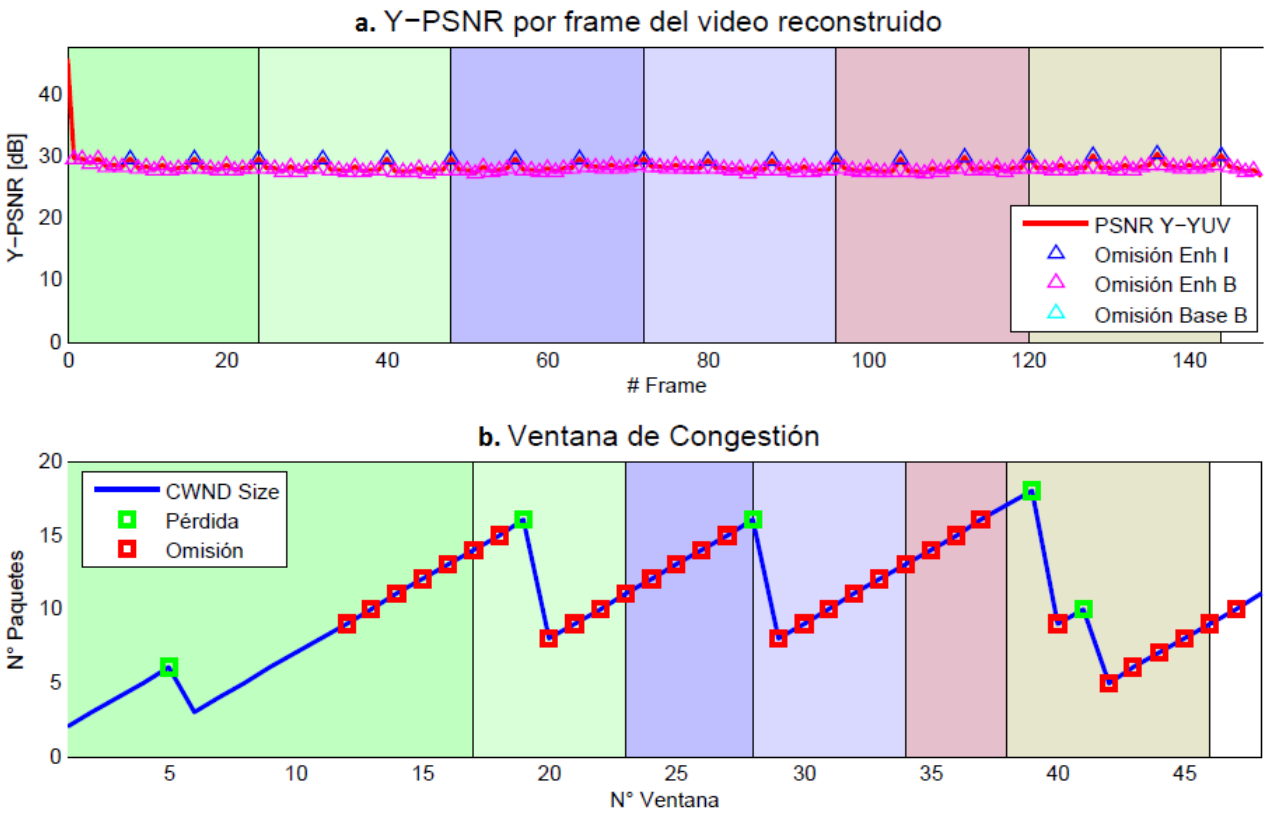

Figura 4.5: PSNR y Ventana de Congestión Escenario Continental

Sin embargo, al igual que en el caso nacional, las pérdidas (si bien más graves que antes) sólo son de calidad, por lo cual en el video final sólo se visualiza una degradación generalizada del video, sin frames faltantes ni artefactos.

La reconstrucción asociada se visualiza en la figura 4.6, en donde es claro como sólo el primer *frame* recibio la capa en *enhancement*-I respecto de quiénes le siguen.

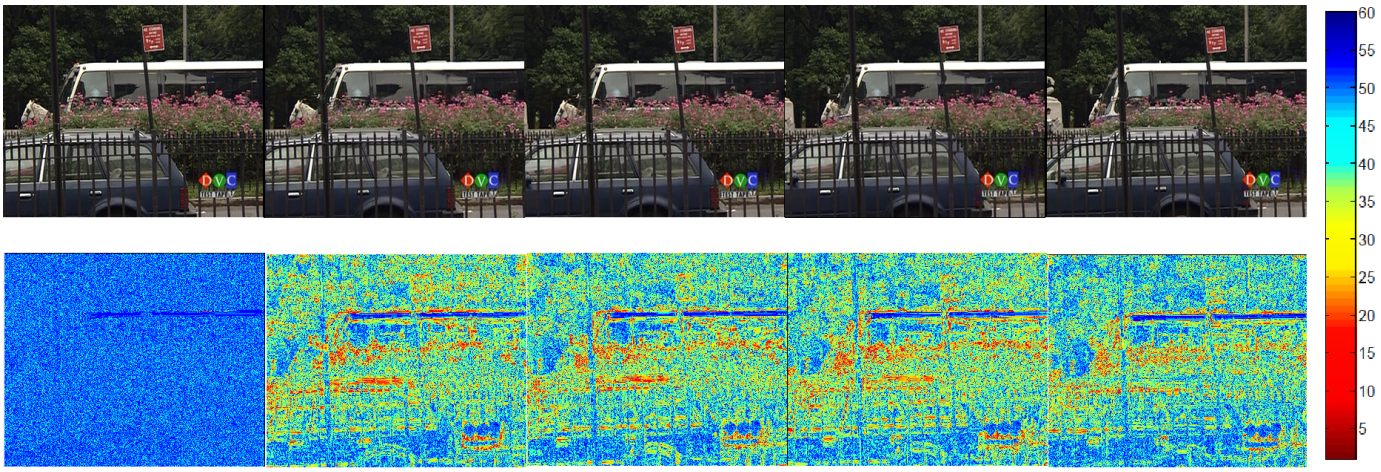

Figura 4.6: Reconstrucción de video, frames del 1 al 5

4.3. Escenario Transcontinental

Para un escenario de este tipo, se puede considerar unos 300ms como el valor de RTT para transmisiones.

El comportamiento del algoritmo puede visualizarse en la figura 4.7

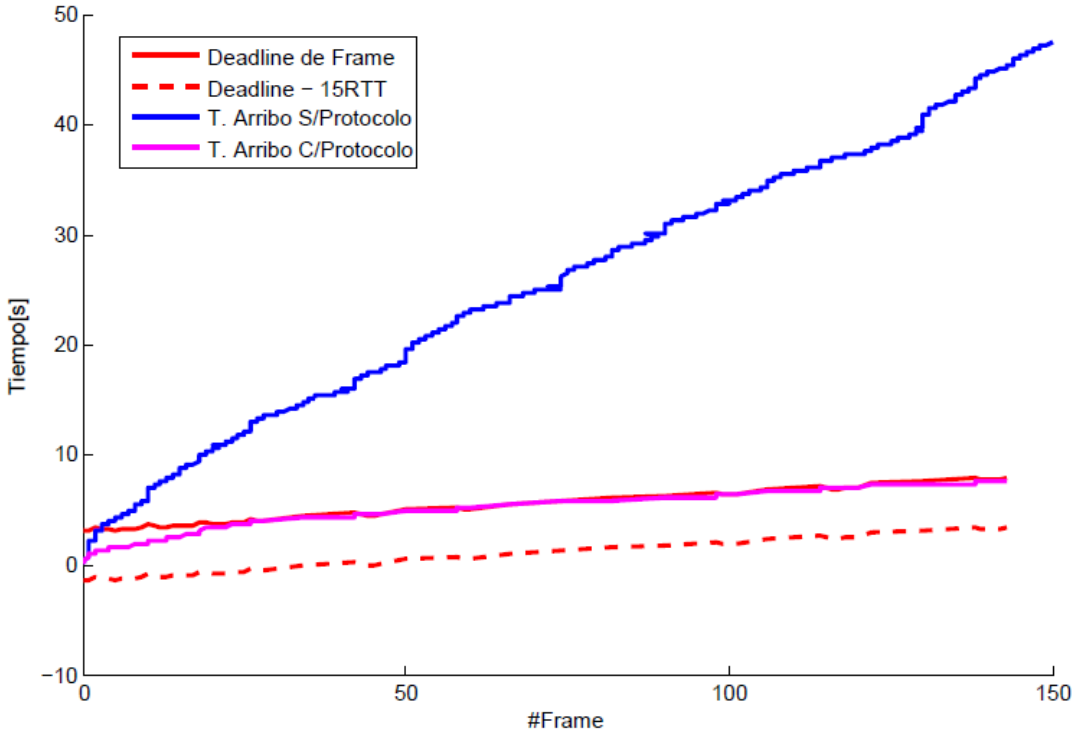

Figura 4.7: Tiempos de Transmisión de Video por Frame, Loss rate: 0.01, RTT: 0.3s

Esta vez, se puede notar un escenario mucho más agresivo, ya que la curva magenta del protocolo es prácticamente co-lineal al deadline de frame. El escenario es tan crítico que la mayor parte del tiempo se está por debajo del deadline de 5 RTT (no mostrado en la figura, por claridad), resultando en el envío sólo de frames base I. La diferencia en tiempo al final de los 150 frames es de unos abismales 50 segundos aproximadamente.

Las pérdidas asociadas a este escenario se muestran en la figura 4.8.a, en donde lo primero a observar, es que esta vez se cuenta con pérdidas de base B, que ocasionan una degradación generalizada del video, además de frames en blanco y artefactos en éste, convirtiéndose en un video no deseable para el usuario (a pesar de que el video llegue a tiempo para poder ser visualizado).

Otro punto notable es la cantidad de paquetes omitidos, esto se aprecia en la cantidad de ventanas que fueron enviadas para la completitud del video en la figura 4.8.b, en donde se requirieron 27 ventanas de congestión para el propósito, en contraste con las casi 150 del escenario nacional (figura 4.2.b).

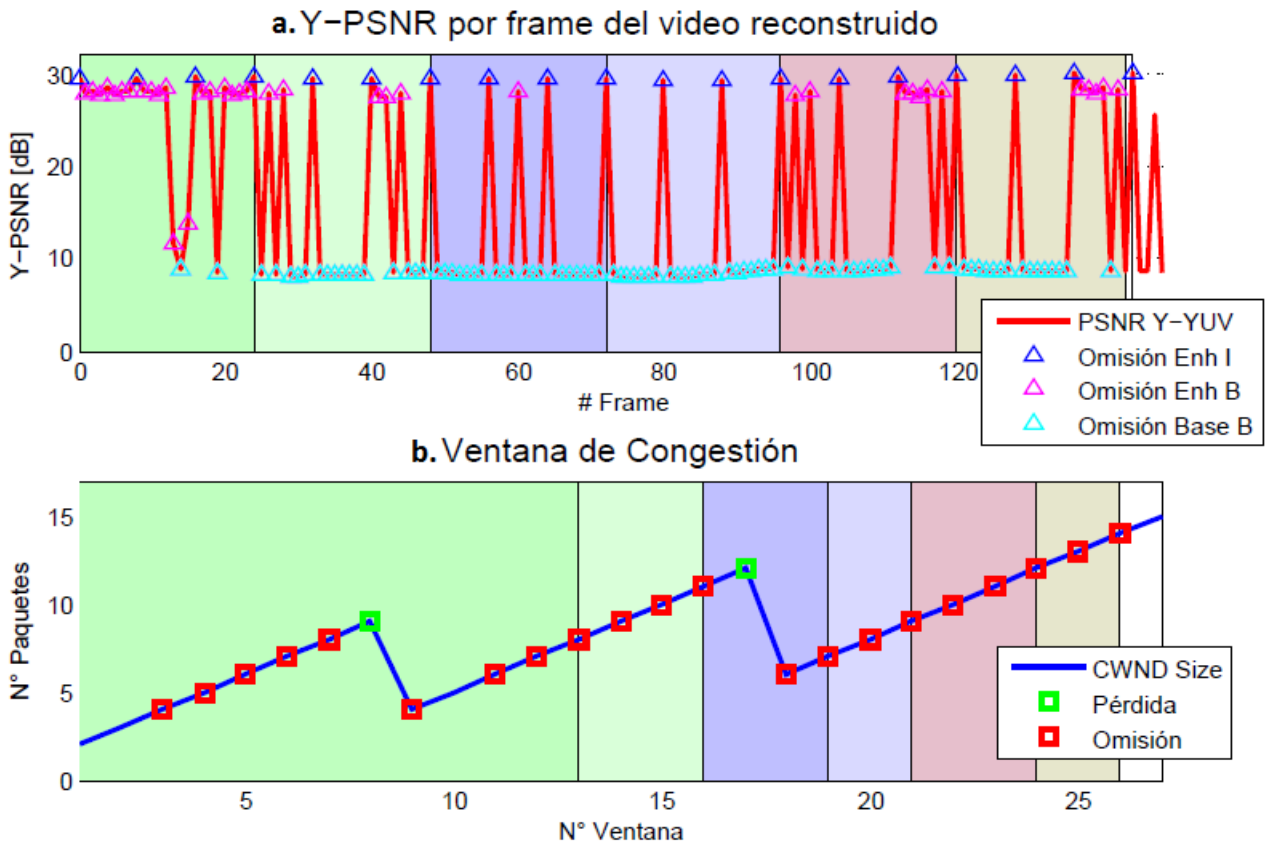

Figura 4.8: PSNR y Ventana de Congestión Escenario Transcontinental

Bajo todo punto de vista, el escenario transcontinental es crítico y produce un video final lleno de degradaciones, frames en blanco y artefactos que resulta no deseable para cualquier usuario. Una fracción de la reconstrucción puede verse en la figura 4.9

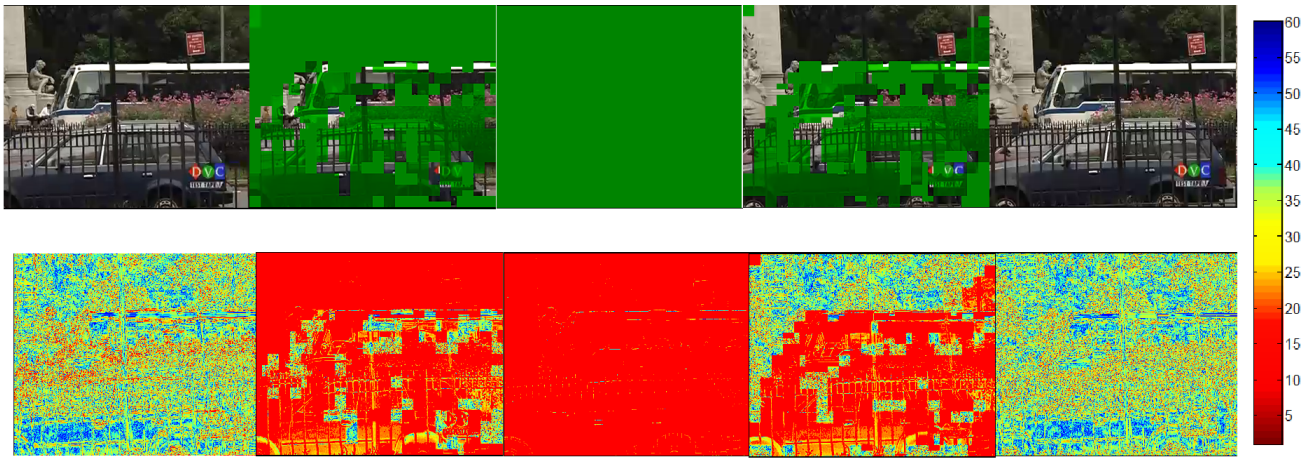

Figura 4.9: Reconstrucción de video, frames del 13 al 18

Para enfrentar esto, se puede comprimir aún más el video, reduciendo el nivel de cuantización de la capa base. Si bien el video perdería calidad, al menos llegará sin artefactos y/o frames en blanco.

Capítulo 5

Conclusiones y Trabajo Futuro

A continuación se presentan las conclusiones a partir de lo mostrado en la sección de resultados y además se establecen las líneas de lo que podría hacerse en un trabajo futuro.

5.1. Conclusiones

A partir del algoritmo de deadline implementado y los escenarios propuestos para que éste mostrase su potencial, se puede afirmar que se cumple el objetivo propuesto en un principio, de facilitar, desde una perspectiva de usuario, la reproducción de archivos de video a costa de un mínimo de pérdidas de calidad. Esta afirmación se respalda en los resultados mostrados en el escenario nacional y continental, en donde, gracias a la omisión selectiva del algoritmo de deadline, sólo existen pérdidas selectivas, evitando frames en blanco u artefactos en el video final. Esta situación se da generalmente para tiempos menores a 150 ms de RTT.

El algoritmo de deadline muestra importantes resultados frente a un escenario TCP continental, en el cual logra obtener una ventaja de 20 segundos al término de los 150 frames, cifra que puede aumentar considerablemente si se toma en cuenta una transmisión *streaming* de mayor duración. Otro punto a favor del algoritmo de deadline, es que es fácilmente escalable a más capas de calidad SVC, pudiendo establecer más *deadlines* de seguridad para un video de calidad HD, por ejemplo. Finalmente, es de destacar el hecho de que el algoritmo de deadline funciona única y exclusivamente por parte del servidor, no introduciendo paquetes de control adicionales sobre la red que podrían perjudicar el desempeño de la transmisión.

En contraparte, si el escenario es muy agresivo, como en el caso transcontinental, el algoritmo de deadline sólo logra enviar un video pobremente reconstruido, repleto de degradaciones y artefactos que resulta, ante todo criterio, desagradable para el usuario. En tal caso, es deseable que el usuario tenga la opción de usar o no el algoritmo, de acuerdo a sus necesidades. Todos los resultados finales pueden ser mejorados si usamos un video no dinámico como el usado en este trabajo. La información enviada a través de la red disminuye por la información redundante presente en el video (fondo fijo, pocos movimientos etc.) y la fluidez del video en el receptor es algo más fácil de alcanzar en términos generales.

Como conclusión final, el algoritmo de deadline resulta ser útil para escenarios de video-conferencia *streaming*, en donde la calidad del video no es algo primordial, y puede sacrificarse en pos de una mayor fluidez del video. No obstante, este algoritmo también puede ser usado en el contexto del entretenimiento para plataformas como YouTube o Netflix, siempre y cuando el usuario lo desee.

5.2. Trabajo Futuro

Primero que todo, el trabajo actual se hizo sobre un video de resolución de 352x288 de 5 segundos de duración codificado sobre 2 capas SVC, escenario bastante acotado. Por tanto se hace natural efectuar un análisis de sensibilidad frente a distintos parámetros de codificación SVC, respecto al tamaño del GOP, el nivel de cuantización de las capas, y la cantidad de éstas para videos de distinta resolución y duración. Esta información es importante ya que podría dar las primeras luces sobre cual es la parametrización óptima de codificación SVC que permite un mayor aprovechamiento de la tecnología para la transmisión de datos de video *streaming*.

Dentro de lo que respecta al algoritmo, también es importante el tipo de *deadlines* a usar, ya que en el presente trabajo se utilizó una distribución uniforme de *deadlines* para la selección de la información a omitir. Si la cantidad de capas aumenta (para un video de mayor resolución, por ejemplo) sería interesante ver el desempeño de una separación no-uniforme de *deadlines*, como por ejemplo una cuadrática. También dentro del algoritmo de deadline, se propone una selección de paquetes a omitir más elaborada. En el escenario actual, sólo es posible la omisión de los paquetes dentro de la ventana de congestión en cuestión, independiente de la jerarquía de éstos (ver figura 3.6), por lo que podría ser deseable descartar paquetes futuros (fuera de la ventana actual) de menor jerarquía por sobre paquetes de mayor importancia dentro de la misma ventana.

Respecto al contraste de resultados con el protocolo TCP, con el cual se comparó el algoritmo de deadline, se propone contrastar el algoritmo con estándares más ad-hoc del formato *streaming* como RTSP, UDP o DASH. Este contraste debe efectuarse una vez que el algoritmo de deadline esté completamente optimizado sobre los puntos ya expuestos para lograr hacer frente a los esquemas de transmisión uni-direccional como lo son estos protocolos.

Por último, el camino natural a seguir es la implementación sobre un sistema real de transmisión, con el propósito de evaluar el desempeño de la propuesta en condiciones reales, como RTT, *loss rates* variables, consideración de tiempos de procesamiento de los extremos etc. Como se puede ver, son muchas las líneas de trabajo que se pueden abordar para un mejor desarrollo de la propuesta, evidenciando el gran potencial de ésta. Así, es posible constituir fácilmente una nueva línea de investigación en el campo de la transmisión de contenido *streaming* alrededor del globo.

Bibliografía

[1] Lawrence Harte David Price. Imagen gop. Disponible en: http://www.iptvdictionary.com/iptv_dictionary_MPEG_GOP_definition.html. Accessed: 2015-06-15.

[2] Divyashri Bhat Dilip Kumar Krishnappa and Michael Zink. DASHing YouTube: An Analysis of Using DASH in YouTube Video Service. Technical report, Octubre 2013.

[3] R. Boreli G. Sarwar and E. Lochin. Xstream-X264: Real-time H.264 streaming with cross-layer integration. RFC 5681, In Multimedia and Expo (ICME), 2011 IEEE International Conference on, Julio 2011.

[4] Detlev Marpe Heiko Schwarz and Thomas Wiegand. Overview of the Scalable Video Coding Extension of the H.264/AVC Standard . Journal, Fraunhofer Institute for Telecommunications, Heinrich Hertz Institute (HHI), Julio 2007.

[5] ITU-T Video Coding Experts Group (VCEG) Joint Video Team (JVT). Joint scalable video model 2011. Disponible en: http://www.hhi.fraunhofer.de/de/abteilungen/video-coding-analytics/research-groups/image-video-coding/research-topics/svc-extension-of-h264avc/jsvm-reference-software.html. Accessed: 2015-07-11.

[6] V. Paxson M. Allman and E. Blanton. TCP Congestion Control. RFC 5681, International Computer Science Instite (ICSI), September 2009.

[7] Sandvine: Inteligent Broadband Networks. Global internet phenomena report. Disponible en: https://www.sandvine.com/downloads/general/global-internet-phenomena/2014/1h-2014-global-internet-phenomena-report.pdf. Accessed: 2015-07-08.

[8] D. T. Nguyen and J. Ostermann. Congestion Control for Scalable Video Streaming Using the Scalability Extension of H.264/AVC. RFC 5681, Selected Topics in Signal Processing, IEEE Journal of, Agosto 2007.

[9] Inc.-Mountain View CA USA Chatterjee S. Paulsamy, V. Zapex Technol. Network convergence and the NAT/Firewall problems . Technical report, School of Information Science Claremont Graduate University , Enero 2003.

[10] J. Postel. The TCP Maximum Segment Size and Related Topics. RFC 879, International Computer Science Instite (ICSI), November 1983.

[11] Lain E. Richardson. *The H.264 Advanced Video Compression standard.* Wiley, 2nd edition, 2010.

[12] Christian Timmerer. Mpeg dash tutorials. Disponible en: `http://multimediacommunication.blogspot.co.at/2013/09/mpeg-dash-tutorials.html`. Accessed: 2015-07-08.

[13] Dmitriy Vatolin. Msu video quality measurement. Disponible en: `http://compression.ru/video/quality_measure/info_en.html`. Accessed: 2015-07-31.

[14] David Yuheng Zhao. Yuv video tool kit. Disponible en: `http://www.yuvtoolkit.com/`. Accessed: 2015-07-30.