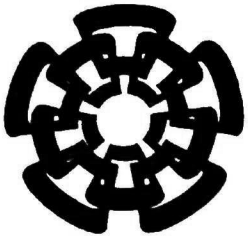
XX (178592.1)



Centro de Investigación y de Estudios Avanzados
Unidad Guadalajara

Trans-codificación y Transmisión de Video en Redes Convencionales y Activas

Tesis presentada por:
Joel Isaac Ramírez Barba

para obtener el grado de:
Maestro en Ciencias

en la especialidad:
Ingeniería Eléctrica

Director de Tesis:
Dr. Mario Angel Siller González Pico

Guadalajara, Jalisco, Diciembre de 2008.

ID 159233-1001

Trans-codificación y Transmisión de Video en Redes Convencionales y Activas

Tesis de Maestría en Ciencias
Ingeniería Eléctrica

Por:
Joel Isaac Ramírez Barba
Ingeniero en Computación
Universidad de Guadalajara 2002-2006

Becario de CONACYT, expediente no. 203130

Director de Tesis:
Dr. Mario Ángel Siller González Pico

CINVESTAV Unidad Guadalajara, Diciembre de 2008.

Resumen

En esta tesis se presenta una arquitectura que aprovecha las características de las redes activas para desplegar un servicio de transcodificación en los nodos internos de la red, disminuyendo el tráfico en la red generado por una transmisión de video multicast y resolviendo los problemas de la heterogeneidad existentes en las transmisiones de video, tales como la capacidad de procesamiento en los nodos finales e intermedios y el ancho de banda de los enlaces de la red.

Se utiliza una arquitectura distribuida que permite a los usuarios conectarse y desconectarse en cualquier momento de una transmisión de video, además, se elimina el retardo que ocasiona el obtener un mapa global de la red necesario en las arquitecturas centralizadas. En la arquitectura, los nodos activos se comportan como agentes reactivos que reaccionan a las señales de entrada o mensajes de control de acuerdo al estado en el que se encuentren y sus algoritmos. En esta tesis se estudian los estados en los que un nodo activo puede estar a la llegada de un mensaje y se describen las acciones que debe realizar el nodo activo a la llegada de un mensaje.

Se incorporan mecanismos de adaptación a la falta de recursos tanto en los nodos como en los enlaces de la red, y mecanismos de restablecimiento ante la liberación de recursos en los nodos o en los enlaces. Los mecanismos de adaptación propuestos aumentan la probabilidad de conexión y disminuyen la pérdida de paquetes en la red, mientras que los métodos de restablecimiento tratan de convertir el árbol multicast de la sesión de video en un árbol multicast óptimo que de otra manera se generaría si no se utilizasen los mecanismos de adaptación.

Finalmente, se presenta un modelo de redes de Petri para la arquitectura propuesta donde se estudian las propiedades de libertad de bloqueos, acotamiento y vivacidad. Así mismo, se estudia el desempeño de la arquitectura propuesta mediante varias simulaciones en NS2.

Abstract

This thesis presents a video transcoding architecture based in active networks. This architecture reduces the network traffic in a multicast video transmission in relation to conventional methods. It also proposes a solution to the heterogeneity problem derived from the processing power at both end and intermediate nodes and the network links available bandwidth.

The use of a distributed architecture allows users to join and leave a video session, and avoids the associated delay for centralised approaches. The architecture active nodes behave as reactive agent. These react to input signals or to the arriving control messages according to its internal state and algorithms. This thesis studies the node internal states originated from an arriving message and the actions that must be taken accordingly.

Active nodes are adaptable to both the nodes and links resources constraints. This adaptability capacity proved to increase the connection probability and reduce the packet loss across the network. In the other hand it tries to convert the multicast video session tree into an optimal multicast tree, otherwise being formed if the adaptability methods hadn't been employed.

Finally, presents a Petri nets model of the architecture is proposed to study properties such as deadlock, boundedness and liveness. A performance study of the proposed architecture is presented through several simulation scenarios using NS2.

Agradecimientos

A Dios por regalarme la vida...

A mis papás Joel y Celina por su esfuerzo, apoyo y cuidado incondicional en cada etapa de mi vida...

A mis hermanos Jesel, Nayeli y Nezlim por su compañía, amistad y apoyo...

A mis compañeros y amigos porque con su amistad aligeraron la carga en tiempos difíciles...

A mi asesor Mario Siller por sus consejos y apoyo en el desarrollo de la tesis...

A mis profesores por su dedicación y empeño al compartir sus conocimientos...

Al CINVESTAV por seguir formando ciudadanos preparados para afrontar la vida...

Al CONACYT porque cree en el talento que existe en México y lo apoya otorgando becas como la que recibí...

El hombre encuentra a Dios
detrás de cada puerta que la
ciencia logra abrir.
(A. Einstein).

Índice de contenidos

Índice de Figuras

Índice de Tablas

CAPITULO 1

1 Introducción

Resumen: En este capítulo se muestra el planteamiento del problema de la heterogeneidad en las redes. Además, se expone la motivación para atacar dicho problema y el objetivo general de esta tesis.

1.1 Planteamiento del problema

Debido a la actual tendencia hacia el cómputo ubicuo (omnipresente) la diversidad de dispositivos que se conectan a las redes varían desde sensores, teléfonos celulares, PDAs, computadoras portátiles y computadoras personales hasta consolas de video juegos y televisores de alta definición. Esta discrepancia en capacidades de procesamiento, almacenamiento y visualización es conocida como heterogeneidad de los dispositivos finales. Dentro de la red también existe cierta diferencia entre las capacidades de procesamiento, almacenamiento y transmisión de los dispositivos internos de red (routers, switches, gateways, etc.), así como en la capacidad de los enlaces que los interconectan que varían en ancho de banda, retardo, etc. La variedad de dispositivos finales e internos y la diversidad de tecnologías en la red conforman el problema conocido como heterogeneidad de la red.

Al unirse a una transmisión, ya sea de video, voz o audio, los clientes desean tener la mejor calidad posible, tomando en cuenta sus limitaciones debidas a su conexión y dispositivo final. En una transmisión multicast esto no es posible por sí sólo, los clientes con mayores recursos es razonable suponer que demandan una calidad de video mayor a la demandada por los clientes con pocos recursos. Si se transmite un solo flujo pueden darse dos casos: en el primero se hace la transmisión a la tasa de bits requerida por el cliente con mayores recursos, en este caso los clientes con menos recursos se verán afectados por una alta pérdida de paquetes si su enlace no es suficiente o el nodo no tiene los recursos necesarios para reproducir el contenido. En el segundo caso se hace la transmisión a la tasa de bits requerida por el cliente con menores recursos asegurando que todos puedan recibir y reproducir el contenido, en este caso los nodos con más recursos se encontrarán limitados a una transmisión con menor calidad que la deseada.

Este problema se puede atacar adaptando el video al formato requerido por los usuarios mediante el proceso de transcodificación y envío del flujo de video en varias resoluciones. En las redes convencionales la transcodificación se hace en los nodos finales de la red, ya sea que se realice en el servidor que envía el flujo o en servidores especiales que proveen este servicio. El problema es la cantidad de tráfico que se genera al enviar varios flujos a través de la red. Para disminuir el tráfico se puede optar por realizar la transcodificación dentro de la red. Para llevar a cabo esto se pueden incorporar *gateways* especiales a la red ó proporcionar cierta capacidad de procesamiento en la capa de aplicación a los dispositivos internos de la red (red activa).

Al desarrollar un servicio de transcodificación en redes activas surgen problemas tales como: la creación del árbol multicast y la localización de los transcodificadores para optimizar los recursos de la red; la conexión y desconexión dinámica de usuarios; la adaptación a los recursos de los dispositivos de los usuarios; la adaptación a la falta de recursos de los nodos para realizar la transcodificación y la readaptación a la liberación de recursos; la adaptación a la falta de recursos de los enlaces para enviar el video y la readaptación a la liberación de recursos; las herramientas de redes activas que pueden ser aprovechadas.

1.2 Motivación

El crecimiento de la transmisión de video en Internet, ya sea en tiempo real o sobre demanda (VoD), impulsa la investigación de nuevas formas de transmisión de video que optimicen los recursos de la red y mejoren la calidad de servicio y de experiencia para los usuarios finales.

Es razonable suponer que los usuarios buscan tener la mejor calidad posible, tomando en cuenta sus limitaciones de procesamiento, almacenamiento, transmisión y visualización que tienen. Debido a que cada usuario cuenta con capacidades diferentes es necesario realizar una adaptación del video que sea lo más cercana posible a las necesidades del usuario. Además, con el crecimiento de las transmisiones multimedia en las redes actuales y el problema de tráfico de red que crea una transmisión de video, es necesario tomar en cuenta la adaptación a los recursos de la red, castigando lo menos posible a los usuarios finales.

Debido a la poca flexibilidad de las redes actuales para atacar el problema de la heterogeneidad se ha pensado en utilizar redes activas como una herramienta para hacer más eficiente la transmisión de video.

1.3 Objetivo de la tesis.

El objetivo de esta tesis es investigar acerca de cómo pueden implementarse los esquemas de transcodificación de video en redes activas y auto-reconfigurase de acuerdo a las condiciones de transmisión y de la red de comunicaciones. Como resultado de la investigación proponer una arquitectura de nodo y red activa para el servicio de transcodificación en redes activas que mejore el rendimiento de la red.

1.4 Organización de la tesis.

En el capítulo 2 se presentan algunas definiciones fundamentales para entender el tema, además se muestran los métodos existentes para la transmisión de video. En el capítulo 3 se define la arquitectura propuesta para el servicio de transcodificación en redes activas. En el capítulo 4 se muestra un modelo de redes de Petri utilizado para probar algunas propiedades de la arquitectura propuesta. En el capítulo 5 se muestra una simulación de la arquitectura propuesta, se hace una validación del funcionamiento de la simulación y finalmente se analizan algunas métricas de desempeño. Por último en el capítulo 6 se muestran las conclusiones de la tesis y las limitantes de la arquitectura propuesta que pueden ser atacadas como trabajo a futuro.

CAPITULO 2

2 Marco Teórico.

Resumen: En esta sección se realiza una breve introducción a los principales conceptos relacionados con la tesis. Se definen conceptos como redes activas, codificación de video, transcodificación de video, calidad de servicio y calidad de experiencia. Además, se muestran y analizan algunos de los enfoques utilizados para la transmisión de video y los principales trabajos relacionados con la transcodificación en redes activas.

2.1 Redes Activas

Las redes activas aparecieron como un programa fundado por DARPA (*Defense Advanced Research Projects Agency*) en el año de 1996. En donde se buscaba crear la próxima generación de redes. Una Red Activa, o también llamada red programable, es aquella en la que los usuarios pueden incorporar código, ya sea en los paquetes o en los nodos, para que sea ejecutado por los nodos internos de la red (conocidos también como nodos activos). De esta manera los nodos activos además de realizar sus funciones comunes como el establecimiento de rutas, el encaminamiento de paquetes, el control de tráfico, el filtrado de paquetes, etc. tienen la capacidad de leer los datos de usuario y manipularlos.

Un nodo activo con el fin de ejecutar código cargado por los usuarios provee uno o varios entornos de ejecución (EE). Cada EE define una maquina virtual y una interfaz de programación (API) para que los usuarios desarrollen sus aplicaciones. Las aplicaciones que corren dentro del EE de un nodo activo son llamadas aplicaciones activas (AA). Con el propósito de administrar y proveer de recursos de procesamiento, memoria y transmisión a los EE el nodo activo utiliza un sistema operativo llamado NodeOS. La estructura básica de un nodo activo es mostrada en la figura 2.1.

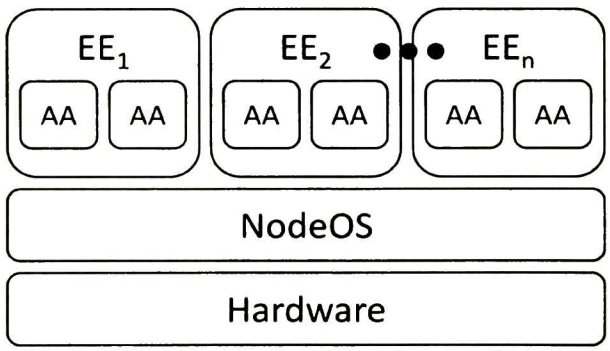


Figura 2.1 Arquitectura básica de un nodo activo

Cuando un paquete llega a un nodo activo, primero el nodo debe verificar si se trata de un paquete común o de un paquete activo. Si se trata de un paquete común, simplemente es encaminado hacia la interfaz correspondiente. Y si se trata de un paquete activo, éste es entregado al EE en donde se ejecuta la AA indicada por el paquete o para que se ejecute el código contenido en él. Si la ejecución del programa demanda el envío de otro paquete activo desde este nodo, la información del nuevo paquete se encapsula de nuevo y se regresa a los procesos de ruteo para su transmisión.

Según Psounis [Konstantinos99] existen tres enfoques para clasificar las arquitecturas de redes activas, paquetes activos, nodos activos y paquetes y nodos activos. Los trabajos sobre redes activas son explicados a fondo en [Galis04], [Bush01] y [Konstantinos99].

Enfoque de paquetes activos. En este enfoque se encapsula código ejecutable dentro de los paquetes de red. Los nodos simplemente ofrecen un entorno de ejecución para los códigos que viajan dentro de los paquetes activos.

Enfoque de nodos activos. En este enfoque los nodos contienen el código activo. Los paquetes únicamente transportan los identificadores de las funciones que se deberán ejecutar y los datos para su ejecución.

Enfoque de paquetes y nodos activos. En este enfoque los paquetes activos transportan código ligero, mientras que en los nodos reside código más complejo permitiendo mezclar las características de los dos enfoques anteriores.

2.2 Codificación de video.

La codificación de video es el proceso en el cuál se convierte un video análogo a digital para su procesamiento en dispositivos digitales, en esta sección se muestra una breve introducción a la codificación de video.

2.2.1 Digitalización de video.

El video por naturaleza es captado como una señal análoga, para poder procesar el video es necesario convertirlo a un formato digital. Para esto se debe realizar un muestreo del video análogo al digital, llamado digitalización.

El video digital consiste de una serie de imágenes, también llamadas cuadros (*frames*), consecutivas mostradas una tras otra creando así el efecto de movimiento. El número de imágenes por segundo de un video depende del estándar que se utilice, por ejemplo, el estándar NTSC define 30 cuadros por segundo mientras que el estándar PAL define 25 cuadros por segundo.

Cada cuadro es conformado por varios puntos llamados pixeles (*picture elements*). Para representar 256 niveles de gris en un video blanco y negro, es necesario tener 8 bits por cada pixel. En el caso del video a color es necesario tener 8 bits por cada color elemental (Rojo, Verde y Azul, *RGB*) siendo así 24 bits por pixel. Los pixeles en ocasiones son agrupados en bloques y macrobloques para su codificación.

2.2.2 Resolución espacial y temporal.

El término resolución espacial es utilizado para referirse al número de pixeles que contiene una imagen, expresado normalmente en el número de pixeles a lo ancho por el número de pixeles a lo largo, ej. 1024 x 768. La resolución temporal es el número de cuadros por segundo que son capturados en un video. Esta es normalmente expresada en fps (*frames* o cuadros por segundo).

2.2.3 Compresión de video.

Supongamos que queremos digitalizar un video a color, con una resolución espacial de 1024 x 768 pixeles, utilizando el estándar norteamericano para televisión, el cual define una resolución temporal de 50fps. En este caso tendríamos una tasa de bits de 900Mbps. (1024 x 768 x 24 x 50 = 900 Mbps).

Para transmitir a esta tasa de bits es necesario contar con tecnologías de red de alta velocidad como SONET que se sólo se encuentran al alcance de pocos usuarios. Para transmitir el video sobre redes de baja velocidad incluso para no saturar la redes de alta velocidad es necesario comprimir el video antes de enviarlo.

La compresión de video puede ser de dos maneras: sin pérdidas (*lossless*) o con pérdidas (*lossy*). En la primera el video puede ser recuperado exactamente igual al original, mientras que en las segunda, durante la compresión se pierde cierta información provocando que el video al ser decodificado no sea exactamente igual al original. Las técnicas de compresión con pérdidas alcanzan una mayor reducción en la tasa de bits, pero tienen una degradación en la calidad del video que la compresión sin pérdidas no tiene.

La compresión de video también puede ser clasificada de acuerdo a las técnicas de codificación que son basadas en predicción o en transformación. En la codificación predictiva, se codifican algunos valores y la diferencia de estos con los demás de tal forma que sólo sea necesario enviar un valor y las diferencias con los demás valores en lugar de mandar todos los valores completos. Para decodificar se hace una predicción de los demás valores utilizando las diferencias obtenidas. En la figura 2.2 se muestran dos imágenes (a) y (b) y la diferencia q existe entre ellas (c). En la codificación por transformación, primero se hace una transformación del dominio espacial a algún otro dominio bien conocido como DCT ó DWT, los valores transformados (coeficientes) son cuantificados para después ser codificados.

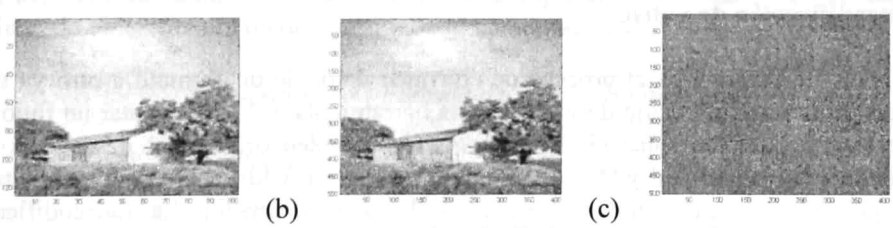


Figura 2.2 (a) y (b) Imágenes originales. (c) Diferencia entre las imágenes.

Al realizar la cuantificación, la cantidad de coeficientes obtenidos y la calidad de la imagen son proporcionales al número de niveles de cuantificación que se utilicen. De esta manera, si se reduce el número de niveles se reducirá el número de coeficientes y por ende el tamaño de almacenamiento de la imagen, pero esto tiene como consecuencia la disminución de la calidad de la imagen. La figura 2.3 muestra la imagen original y la imagen cuantificada con menos niveles, donde se puede apreciar el efecto de la pérdida de la calidad de la imagen al disminuir los niveles de cuantificación.

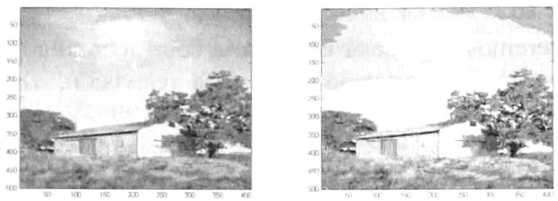


Figura 2.3 Comparación de imágenes con distintos niveles de cuantificación.

La figura 2.4 muestra el proceso para comprimir una imagen con un formato JPEG, donde se inicia transformando la imagen al dominio de la frecuencia (DCT), después se cuantifican los coeficientes (Q) que finalmente son pasados por un codificador de longitud variable (VLC) obteniendo una secuencia de bits que representa la imagen.

Mezclando la codificación predictiva y la codificación por transformación se obtienen los marcos I, P y B. Los marcos I se obtienen a través de una codificación por transformación, los marcos P se obtienen a través de una codificación por predicción utilizando como referencia un marco I y los marcos B sirven para mejorar la resolución temporal del video utilizando como referencia los marcos sucesor y predecesor a él.

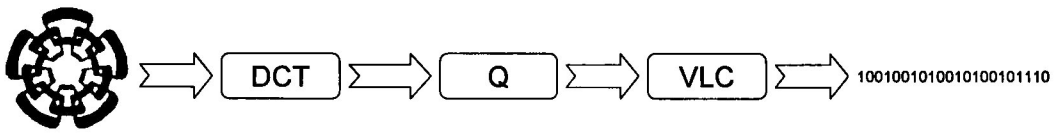


Figura 2.4 Proceso para comprimir una imagen en formato JPEG.

2.3 Transcodificación de video.

La Transcodificación es el proceso de convertir datos de un formato a otro ya sea audio, voz, texto o video. La transcodificación de video es una herramienta útil para adaptar un flujo de video a los recursos de la red y de los usuarios. Transformando el video original al deseado por el usuario ó simplemente bajando su calidad y tasa de bits. Cuando la transcodificación simplemente disminuye la tasa de bits del video sin cambiarlo de formato es llamada *transrating*. La transcodificación de video puede ser realizada en tres dominios: dominio de los pixeles, dominio de la frecuencia y dominio completamente comprimido.

Dominio de los pixeles, la transcodificación en este dominio implica la decodificación y codificación completa del video. Como lo muestra la figura 2.5, en T1 se realizan operaciones directamente a los pixeles para disminuir el tamaño del video. Ejemplos de operaciones en este dominio son bajar los niveles de los colores, disminuir la luminosidad y cambiar el tamaño de la imagen.

Dominio de la frecuencia, en este dominio la transcodificación se realiza manipulando los coeficientes resultantes de alguna transformación al dominio de la frecuencia (DCT ó DWT). Esto implica una decodificación y codificación parcial del video. Las operaciones en este dominio son representadas en la figura 2.5 por T2.

Dominio completamente comprimido, la transcodificación en este dominio se realiza tirando directamente los cuadros I, P y B del video. Existen técnicas de tirado inteligente que disminuyen la pérdida de calidad de video que implica el tirar todo un cuadro. El tirado de paquetes es mostrado en la figura 2.5 como T3.

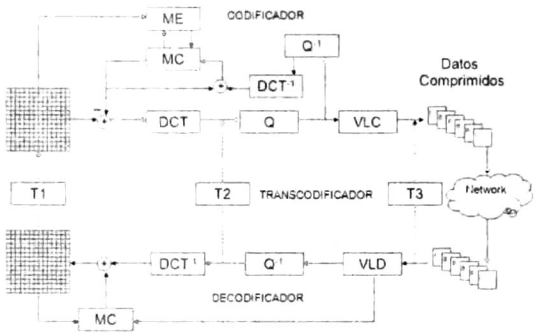


Figura 2.5 Dominios para la transcodificación de video.

De acuerdo con Vetro en [Vetro03] y Khan en [Khan02] la transcodificación de video es un trueque (*tradeoff*) entre la calidad de video, la reducción de la tasa de bits y la complejidad computacional.

En la tabla 2.1 se muestra la relación entre la pérdida de calidad de video y la complejidad computacional cuando se busca disminuir la tasa de bits. En el dominio de los pixeles se tiene la menor pérdida de calidad de video pero la mayor complejidad computacional. En el dominio de la frecuencia para disminuir la pérdida en la calidad del video se deben utilizar técnicas que aumentan la complejidad computacional, teniendo una pérdida de calidad de video y complejidad computacional media en comparación a los otros dos dominios. Y en el dominio completamente comprimido se tiene la mayor pérdida de calidad de video pero la menor complejidad computacional.

Dominio	***Pérdida de calidad***	***Complejidad computacional***
Pixel	Baja	Alta
Frecuencia	Media	Media
Comprimido	Alta	Baja

Tabla 2.1 Trade-off entre Calidad de video perdido y complejidad computacional

2.4 Transmisión de video.

Existen varios enfoques para transmitir el video sobre una red, aquí se muestran los que se consideraron como los principales. La tabla 2.2 muestra los enfoques para la transmisión de video y los clasifica de acuerdo al número de tasas de transmisión que soportan, ya sea que soporte una (*tasa simple*) o varias tasas (*Multi-tasa*) y de acuerdo al lugar donde se realiza la adaptación de los flujos, ya sea en alguno de los extremos de la conexión (*fin a fin*) ó en algún lugar dentro de la red (*activa*).

Enfoque	Tasa de video	Adaptación del flujo
Adaptación a una tasa simple.	*Tasa simple*	*Fin a fin*
Simulcasting.	*Multi-tasa*	*Fin a fin*
Adaptación a flujos de video por capas.	*Multi-tasa*	*Fin a fin*
Gateway multimedia a nivel aplicación	*Multi-tasa*	*Activa*
Video por capas en redes activas.	*Multi-tasa*	*Activa*
Transcodificación en redes activas.	*Multi-tasa*	*Activa*

Tabla 2.2 Enfoques para el envío multicast de video.

Adaptación a una tasa simple.

En este enfoque el servidor de video envía un solo flujo de video a todos los usuarios, ocasionando que los usuarios finales deban adaptarse a dicho flujo. Existe dos problemas con este enfoque, si la calidad del video es mayor a las capacidades del dispositivo el usuario experimentará una pérdida en la calidad del video debido a la falta de recursos ó incluso la pérdida total del video al no poder reproducirlo. Si por el contrario se recibe una calidad menor a la soportada por el dispositivo, no existirá pérdida en la calidad de video pero el usuario estará limitado a la calidad que el servidor este transmitiendo.

Simulcasting.

En este enfoque los usuarios son agrupados de acuerdo a las capacidades de sus dispositivos. El servidor, ya sea que contenga varios formatos del video almacenados o que los obtenga transcodificando el formato original, crea un árbol multicast para cada grupo de usuarios. Este enfoque se adapta a las necesidades de los usuarios pero envía información redundante a la red (distintos flujos del mismo video). La figura 2.6 muestra un ejemplo de envío de video utilizando simulcasting, donde se puede observar el tráfico que se genera en algunos de los enlaces.

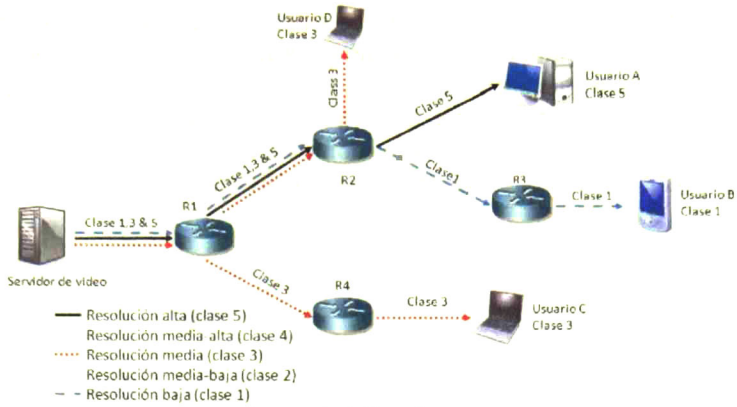


Figura 2.6 Simulcasting.

Adaptación a flujos de video por capas.

En este enfoque se utilizan formatos de video codificados por capas que fueron introducidos por Ghanbari en [Ghanbari99]. El video por capas se compone por una capa base y varias capas de mejoramiento. Los usuarios se subscriben al número de capas que pueden recibir de acuerdo a sus capacidades. Todos los usuarios se subscriben a la capa base que cuenta con la información necesaria para decodificar el video con una calidad baja, la calidad de video es aumentada subscribiéndose a las capas de mejoramiento. Esta aproximación está limitada sólo a los formatos de codificación por capas. La figura 2.7 muestra un ejemplo de transmisión utilizando video por capas, donde los usuarios se subscriben directamente con el servidor de video.

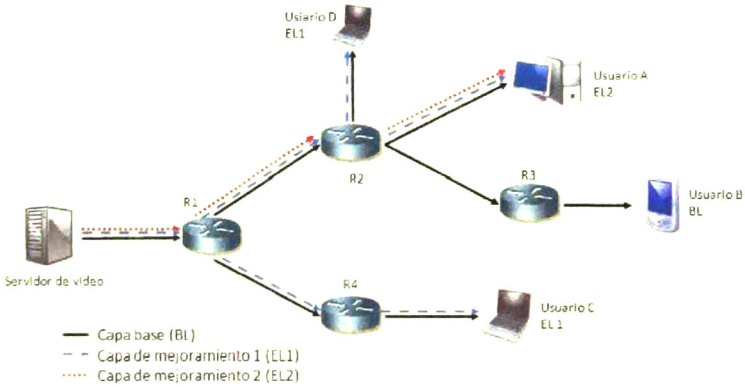


Figura 2.7 Video por capas.

Gateway multimedia a nivel aplicación.

En este enfoque se utilizan gateways especializados en transcodificación de video que son colocados a través de la red para adaptar los flujos originales de video a los flujos requeridos por los usuarios. Los usuarios que necesitan adaptar el video original piden el servicio de transcodificación a los gateways. El hecho de que los gateways se encuentren bien localizados dentro de la red puede

reducir el tráfico en comparación con el enfoque de simulcasting. El problema es que se necesitan varios gateways para hacer más óptimo el desempeño de la red pero el costo de los equipos actuales es elevado. Además de que las actualizaciones de los transcodificadores sólo las pueden realizas los administradores de red que tengan acceso directo al equipo. La figura 2.8 muestra una red donde existe un gateway multimedia que realiza la transcodificación para los usuarios que la necesitan. Se observa que existe tráfico redundante en el enlace del router R1 al Gateway.

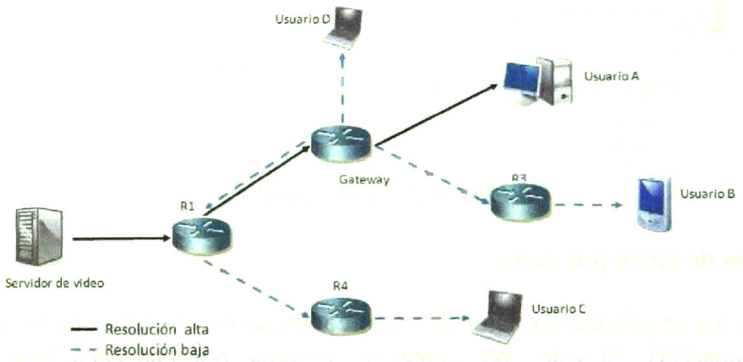


Figura 2.8 Gateway Multimedia a nivel aplicación.

Video por capas en redes activas.

En este enfoque se emplean los mismos formatos de video por capas mencionados anteriormente, con la diferencia de que los usuarios se subscriben con los nodos activos y no directamente con el servidor. Esto libera al servidor de almacenar los datos de todos los usuarios, además hace que los usuarios se conecten más rápido y que se disminuya el tráfico de control en la red. Los nodos activos son los encargados del filtrado de la capas de video de acuerdo con las peticiones de los usuarios. El enfoque sigue teniendo el problema de poca flexibilidad de los formatos de video por capas.

Transcodificación en redes activas.

En este enfoque se colocan transcodificadores dentro de los nodos activos. El servidor de video únicamente envía el formato más grande pedido por un usuario y los nodos activos se encargan de adaptar dicho formato al formato pedido por cada usuario por medio de la transcodificación. Este enfoque disminuye el tráfico de red al sólo enviar un solo flujo de video y soporta cualquier formato de video que sea instalado en los nodos activos. La desventaja del uso de la transcodificación es la cantidad de recursos que se necesitan para llevarla a cabo y el retardo que el proceso de la transcodificación agrega a los paquetes, esta última desventaja también la sufren los demás enfoques que utilizan la transcodificación como técnica de adaptación. Sin embargo, las arquitecturas de procesadores tales como Dual core, entre otras, pueden mitigar dicho tiempo. La figura 2.9 muestra como la adaptación del flujo se realiza en los nodos activos de la red.

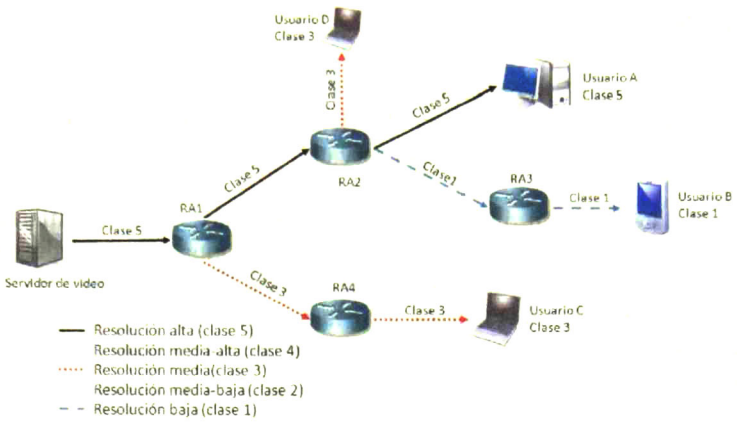


Figura 2.9 Transcodificación en redes activas.

Ramírez-Barba y Siller en [ramirezb08] muestran un análisis de los trabajos relacionados con la transcodificación de video en redes activas. Los trabajos son citados y detallados en el capítulo 3 sirviendo como fundamento para la arquitectura propuesta. La tabla 2.3 muestra un resumen del trabajo relacionado donde se analiza: la técnica de distribución de código (*code distribution method*), el retardo de conexión (*set-up delay*), la adaptación a los recursos de los nodos y de los enlaces (*network and node adaptation*), el formato multimedia soportado (*media format*) y el dominio en el que se realiza la adaptación ó transcodificación (*transcoding domain*). El retardo de conexión se refiere al tiempo necesario para localizar e instalar el código de la transcodificación para iniciar la transmisión. Es definido como bajo (*low*) si el código se encuentra pre-instalado en los nodos o si ya ha sido cargado con anterioridad, Cuando el código es transportado dentro de los paquetes activos se considera que se tiene un retardo medio (*medium*) y alto (*high*) cuando es necesario cargarlo desde una base de código ó si el sistema utiliza un algoritmo centralizado para su localización. La adaptación a la red y a los nodos se refiere a la capacidad de ajustarse a los recursos de procesamiento, almacenamiento y transmisión disponibles en la red.

Approach	Code distribution method	Set-up delay	Network resources adaptation	Node resources adaptation	Media format	Transcoding Domain
Active Video[16]	Active packets	Medium	NS	NS	MPEG4	Fully Compressed
Self-organising Video [17]	Active packets	High, (Centralised)	YES (triggered by coding standard and video content awareness).	YES (computation optimisation and distribution)	MPEG-2	Pixel
ARTES 1 [18]	Active nodes (Code Base)	High (initially loaded) / Low (previously loaded or preinstalled)	NS	YES (computation distribution)	JPEG2000	DWT (equivalent to DCT)
MeGa [20]	Active nodes (NS)	Low	NS	NS	NS	NS
Duysburgh active framework [22]	Active nodes (ANTS-Code Based)	High (initially loaded) / Low (previously loaded or preinstalled)	NS	YES (computation distribution)	PCM, G.711, G 726, G.728, G.729	Voice (Fully decoding required – equivalent to Pixel)

Tabla 2.3 Trabajo relacionado con la transcodificación en redes activas. NS = No especificado.

2.5 Calidad de Servicio QoS

La ITU-T y la ETSI en [ITU-TE.800] y [ETSI-ETR003] definen la calidad de servicio ó QoS (por sus siglas en ingles "*Quality of Service*") como "El efecto colectivo del desempeño del servicio el cuál determina el grado de satisfacción del usuario del servicio"

Los mecanismos de QoS son divididos en dos categorías: servicios diferenciados (*diffserv*) y servicios integrados (*intserv*). Para asegurar la calidad los servicios diferenciados se dan privilegios al tráfico de acuerdo a los servicios prestados a cada usuario, mientras que los servicios integrados utilizan protocolos de reservación de recursos a lo largo del camino que seguirán los paquetes del usuario. Bai en [Bai04] menciona a las redes activas como mecanismo para asegurar la calidad de servicio en las redes.

2.6 Calidad de experiencia (QoE)

Siller en [Siller06] define calidad de experiencia o QoE (por sus siglas en ingles "Quality of Experience"), como: *"the user's perceived experience presented by the application layer, where the application layer acts as a user interface to the overall result of the individual quality of services"*.

2.7 Conclusiones

Todos los enfoques para la transmisión de video aquí presentados tienen sus ventajas y desventajas. El uso de redes activas puede ayudar a optimizar el desempeño de las técnicas utilizadas en las redes convencionales, pero tienen la desventaja de que los proyectos para la implementación de las redes activas comerciales se encuentren aún en investigación.

El uso de la transcodificación permite adaptar los flujos de video a las necesidades de los usuarios de manera flexible, pero demanda una complejidad computacional alta. Existen actualmente muchas investigaciones en el ámbito de la transcodificación multimedia que buscan reducir la complejidad computacional y aumentar la calidad del video generando la menor tasa de bits posible.

En cuanto a los trabajos relacionados con la transcodificación de video en redes activas, todos proveen adaptación a la heterogeneidad de los usuarios. Sólo algunos proveen adaptación a los recursos de los nodos internos utilizando técnicas como la difusión de la computación y la optimización de la computación. Y sólo uno provee adaptación a los enlaces de la red mediante la disminución de la tasa de bits en los puntos de constricción, pero lo hace de manera centralizada y estática. ¿Se puede mejorar el desempeño de la red utilizando técnicas de adaptación a los recursos disponibles?.

CAPITULO 3

3 Arquitectura para la transcodificación en redes activas

Resumen: Este capítulo detalla la arquitectura propuesta para la transcodificación de video en redes activas. Se hace un análisis de los estados que un nodo activo puede tener a la llegada de un mensaje de control y las acciones que este debe realizar. En cada una de la secciones se hace una introducción con el trabajo relacionado en el área o estado del arte, se plantea una solución y se fundamenta lo propuesto.

3.1 Objetivos de la arquitectura.

Desafíos de los sistemas distribuidos.

Una arquitectura de red puede ser vista como un sistema distribuido, en el caso de las redes activas los nodos internos son parte fundamental del sistema distribuido. De acuerdo con esto, se han considerado algunos de los desafíos sugeridos por Coulouris en [Coulouris01] para el diseño de un sistema distribuido.

Heterogeneidad. El sistema debe considerar la variedad y la diferencia entre las tecnologías de red, los dispositivos hardware, los sistemas operativos, los lenguajes de programación, etc. que conforman una red.
Extensibilidad. El sistema debe de proveer la posibilidad de ser extendido o re-implementado.
Seguridad. El sistema debe de proveer cierto grado de confidencialidad (protección contra el abuso de individuos no autorizados), integridad (protección contra la alteración o corrupción de información) y disponibilidad (protección contra la interferencia al acceso a los recursos).
Escalabilidad. El sistema debe de mantener su efectividad y estabilidad ante el incremento en el número de recursos y el número de usuarios.
Tratamiento de fallos. El sistema debe proveer mecanismos para el tratamiento de fallos.
Concurrencia. Cada recurso compartido del sistema debe garantizar que opera correctamente en un entorno concurrente.
Transparencia. El sistema debe de ser percibido como un todo más que como una colección de componentes por el usuario.

Objetivos de la arquitectura.

Los objetivos que se buscan cubrir en la arquitectura propuesta son:

1) Proveer un mecanismo de adaptación para la transmisión de video de acuerdo a los requerimientos de los usuarios.

2) Aumentar la probabilidad de que un usuario obtenga un flujo de video mediante la adaptación dinámica a los recursos de los nodos activos basándose en la capacidad de procesamiento, memoria y transmisión de los nodos activos. Combinando los métodos de difusión y optimización de la computación.

3) Ofrecer un mecanismo para restablecer la difusión de la transcodificación y la optimización de la computación ante la liberación de recursos en los nodos activos.

4) Disminuir la pérdida de paquetes de video debida a la congestión de los enlaces de la red mediante la adaptación dinámica a los recursos de la red utilizando la degradación de la tasa de bits para disminuir el tráfico de la red, basada en métricas como el ancho de banda disponible y la pérdida de paquetes. Utilizando una adaptación distribuida para optimizar los retardos.

5) Ofrecer un mecanismo para restaurar la tasa de bits degradada por el mecanismo de adaptación ante la liberación de recursos en los enlaces.

6) Proporcionar un tiempo de conexión menor a los 200 milisegundos que es el tiempo medido en el trabajo relacionado [Duysburgh04], mediante el uso de un algoritmo distribuido estable para el establecimiento y localización de los transcodificadores.

3.2 Definición de la arquitectura propuesta.

La arquitectura propuesta funciona como una aplicación activa (AA) que es montada en la capa de aplicación de un nodo activo sobre una arquitectura de red activa. En cada sección de este capítulo se muestra el estado del arte relacionado al problema a tratar y se hace una crítica al mismo, finalmente se muestra la descripción de la arquitectura propuesta.

3.2.1 Arquitectura de red activa.

Para hacer la arquitectura propuesta menos compleja se supone que la arquitectura de red activa subyacente a la nuestra se encarga de varios aspectos como: el protocolo para la distribución del código; la asignación de los recursos de procesamiento, memoria y transmisión, entre los procesos activos (servicios activos) y los procesos no activos (Ruteo, control de flujo, etc.); el almacenamiento de código; la administración de versiones de código; la seguridad; la tolerancia a fallos.

En [Konstantinos99] las arquitecturas de red activa son agrupadas de acuerdo al enfoque básico con el que se realiza el procesamiento activo. A continuación se presentan los enfoques, una breve descripción de cada uno y las principales arquitecturas relacionadas. Además se hace un análisis sobre la arquitectura más conveniente para montar la arquitectura propuesta para la transcodificación.

Paquetes activos.

Este enfoque sufre de poca seguridad, debido a que códigos malintencionados podrían ser fácilmente ejecutados en los nodos activos ya que todos los paquetes pueden contener código. La forma de aumentar la seguridad es restringiendo la funcionalidad de los programas que son transportados en los paquetes ó implementando un sistema de autentificación por paquete, lo que provoca pérdida en el desempeño de los sistemas. Algunos trabajos que utilizan el enfoque de paquetes activos son *Smart Packets [Schwartz98], Opción de IP activa [Wetherall96] y M0 [Banchs98].*

Nodos activos.

Este enfoque utiliza una base de código que se encuentra en un nodo "bien conocido" para cargar los servicios a los nodos activos además utiliza mecanismos de autentificación de los nodos aumentando la seguridad, además permite que los códigos sean más robustos pues permite que un programa viaje en varios paquetes. Una desventaja de este enfoque es la poca extensibilidad que tiene debido a que el código es pre-cargado en cada nodo activo, ya sea, manualmente (por el administrador de la red) ó por medio de la base de código. Algunos trabajos que utilizan el enfoque de nodos activos son *An architecture for active networks [Bhattacharjee96], Distributed Code Caching for Active Networks (DAN) [Decasper98] y Active Node Transfer System (ANTS) [Wetherall98].*

Paquetes y nodos activos

Este enfoque combina los dos enfoques anteriores tratando de eliminar las desventajas de uno con las ventajas del otro. Algunos trabajos que utilizan el enfoque de paquetes y nodos activos son *SwitchWare [Gunter98] y Netscript [Yemini96].*

Arquitecturas utilizadas para la transcodificación

Najafi et al en [Najafi98] proponen un método basado en el enfoque de paquetes activos llamado *Active Video,* en el cual las operaciones que se van a realizar sobre el video y el video mismo viajan dentro de los paquetes activos, no se proporciona una descripción detallada acerca de la arquitectura. Duysburgh et al en [Duysburgh04] montan una aplicación de transcodificación sobre la arquitectura de red activa ANTS para probar los algoritmos de conexión y difusión que proponen.

Análisis y propuesta de la arquitectura de red activa.

El enfoque de nodos activos provee mayor seguridad, debido a que tiene mayor control en el código que se ejecuta en los nodos activos. Mientras que el enfoque de paquetes activos es menos seguro debido a que todos los paquetes pueden transportar código, esto hace que la posibilidad de ejecutar algún código malicioso en un nodo sea mayor. Para aumentar la seguridad en un enfoque de paquetes activos se pueden implementar mecanismos de autentificación de paquetes, pero esto aumenta la complejidad de los sistemas.

Otro punto a considerar es el tamaño de los programas y las restricciones que estos tienen en cada uno de los enfoques. En el caso del enfoque de paquetes activos los programas son pequeños y tienen restricciones en cuanto a su funcionalidad, esto para aumentar la seguridad y eficiencia. En cuanto a la seguridad los programas pequeños restringidos en funcionalidad pueden causar estragos menores que programas grandes y no restringidos; en cuanto a eficiencia el hecho de transportar un programa en un solo paquete disminuye el tráfico en la red, además el código está disponible en todo momento sin necesidad de cargarlo desde una base de código generando tráfico y un retardo para la aplicación. Por otro lado el enfoque de nodos activos permite programas más grandes y menos restringidos, ya que los códigos residen en los nodos activos o son descargados desde una base de código bien conocida que utiliza mecanismos de autentificación para cuidar la seguridad.

Hablando de disponibilidad en los servicios activos, el enfoque de nodos activos tiene la desventaja de que si no se dispone del código para correr la aplicación en el nodo activo, éste se debe obtener de un servidor de código ó base de códigos, produciendo un retardo en el tiempo de respuesta del servicio. Además, si el código no existe en la base de códigos, es imposible que el usuario obtenga el servicio deseado.

Ya que el proceso de transcodificación demanda una alta complejidad computacional debido a la completa decodificación y re-codificación del video, se opta por utilizar un enfoque de nodos activos. Además, el enfoque de nodos activos ofrece mayor seguridad que el enfoque de paquetes activos.

3.2.2 Estructuras de datos.

En esta sección se detallan las estructuras de datos que permiten que los nodos activos guarden sus estados. Además se definen los tipos de mensaje con los que se comunican las aplicaciones activas que corren en los distintos nodos activos.

Tabla de estados.

De acuerdo con Duysburgh et al en [Duysburgh05] los nodos activos guardan información de los usuarios y de las sesiones que se encuentran disponibles actualmente en el nodo. Es decir sus tablas guardan el estado actual de las sesiones que soporta el nodo. Duysburgh et al utilizan los siguientes campos para guardar la información de las sesiones y usuarios: *Session*, guarda el nombre de la sesión de video. *To*, guarda un puntero al siguiente nodo a quien va a ser enviado el video. *Class*, guarda el formato de video enviado al siguiente nodo. *Tc*, guarda "yes" si el formato requerido es transcodificado a partir del formato de entrada y "no" si el formato se recibe desde otro nodo. *Push*, guarda "yes" si el transcodificador fue movido a este nodo.

La arquitectura que se propone utiliza una tabla de estados. Esta tabla almacena los nodos hijos del árbol multicast, además del formato que se le envía a cada uno y otros campos que determinan el estado del nodo. En la tabla 3.1 se describe brevemente cada uno de los campos de la tabla de estados.

Nombre	Abrev.	Descripción
Session	*s*	Guarda el nombre de la sesión de video.
Destination	*dest*	Guarda un puntero al nodo al cual va dirigido el flujo.
Requested class	*rc*	Guarda la clase pedida por el nodo.
Degraded link class	*dlc*	Guarda la clase degradada por los recursos de red.
Degraded node class	*dnc*	Guarda la clase degradada por los recursos del nodo.
Transcoding	*tc*	Indica si la clase pedida debe ser transcodificada desde el flujo de entrada (*up_class*).
Push	*psh*	Se usa para indicar que la entrada en la tabla no puede ser modificada debido a que pertenece a una difusión de la transcodificación.
Transcoding node	*tn*	Guarda la dirección del nodo donde se realiza la transcodificación si es necesaria.
Degraded node	*dn*	Guarda la dirección del nodo responsable de la degradación del video.
Degraded link	*dl*	Guarda "yes" si el enlace hacia ese nodo esta degradado.

Tabla 3.1 Descripción de los campos de la tabla de estados.

Variables.

Duysburgh et al en [Duysburgh05] hace uso de variables para conocer los siguientes saltos en el árbol. Además guarda un puntero al siguiente salto hacia el servidor de video y el formato de video que se recibe por ese enlace (*C-uplink*).

En la tabla 3.2 se muestran las variables que se utilizan en la arquitectura propuesta y se da una breve descripción de cada una de ellas.

Nombre	Descripción
Top_limit	Límite tolerable para la pedida de paquetes.
Seq_num	Número de secuencia para medir la pérdida de paquetes de datos.
Send_seq	Número de secuencia para el envío de los paquetes de datos.
Packet_loss	Tasa de pérdida de paquetes.
Candidate	Mejor candidato para la difusión.
Req_node	Datos del nodo que pide conectarse.
Leave_node	Datos del nodo que pide desconectarse.
Node_tc	Datos del nodo que necesita transcodificación cuando existe difusión.
Ack	Contador del número de ack recibidos al hacer una petición de difusión.
Up_node	Dirección del nodo padre en el árbol multicast.
Up_class	Formato de video recibido.
Neighbours	Lista de los nodos activos vecinos.

Tabla 3.2 Variables utilizadas por la arquitectura propuesta.

Mensajes

Los mensajes de la arquitectura propuesta, como se puede ver en la figura 3.1, constan de un identificador para el tipo de mensaje (*type*), la sesión de video a la que pertenecen (*session*), un identificador para un formato de video (*vclass*), un número de secuencia (*seq*) y un campo de datos que puede omitirse para algunos mensajes (*dat*). En la tabla 3.3 se muestran los tipos de mensaje y una breve descripción de cada uno de ellos.

Figura 3.1 Estructura de un paquete.

Direccionamiento

Se puede utilizar cualquier tipo de direccionamiento que distinga a los nodos, como pueden ser direcciones IP o MAC. Para efectos de generalizar el direccionamiento y simplificar la explicación, se utiliza un número entero único para cada nodo en la topología (*addr*). Donde el nodo 0 siempre será el servidor de video, los nodos activos y los usuarios pueden tener cualquier otro número que no se repita entre ellos.

Tipo	Descripción	Datos
join	Petición de conexión a la sesión.	Información de la QoE.
join_ack	Acuse de conexión a la sesión.	
join_error	Error en la conexión.	
leave	Petición de desconexión.	
ctrl-down_class	Petición para degradar la clase.	
ctrl-down_classack	Acuse de degradación de clase.	
ctrl-rep	Reporte de tráfico de red.	Tipo de reporte.
ctrl-link_deg	Degradación debido un enlace.	
ctrl-link_res	Restablecimiento debido un enlace.	
ctrl-node_deg	Degradación debido a un nodo.	
ctrl-node_res	Restablecimiento debido a un nodo.	
ctrl-difreq	Petición de difusión.	
ctrl-dify	Acuse positivo de difusión.	Cantidad de recursos.
ctrl-difn	Acuse negativo de difusión.	
ctrl-difack	Confirmación de difusión.	Sentido y bandera para ack.
ctrl-tc_res	Restablecimiento de difusión.	
data	Video.	Bandera de difusión y video.

Tabla 3.3 Tipos de mensajes para la arquitectura propuesta.

3.2.3 Transcodificación y jerarquización de video.

Jerarquización de video

En [Duysburgh00] se clasifican los nodos de acuerdo al tipo de versión del video que están solicitando y ordenan los formatos de mayor a menor tomando en cuenta la tasa de bits. Esto para restringir la transcodificación solo a clases menores.

Como se muestra en la figura 3.2 los formatos en la arquitectura propuesta son llamados clases (*class*) y se les asigna un número *i* de acuerdo a su tasa de bits (*classi*), así que no importa el tipo de formato sino sólo su tasa de bits para clasificarlos. Para agregar un nuevo formato a la arquitectura se debe conocer su tasa de bits para ser clasificado, además, se deben de proveer transcodificadores desde las clases mayores y hacia las clases menores.

Clase	Formato	Bitrate
Class1	MPEG2	64kbps
Class2	H.263	128kbps
Class3	H.264	512kbps
Class4	MPEG4	1Mbps
Class5	H.264	2Mbps

Figura 3.2 Ejemplo de la clasificación de los formatos de video.

Transcodificación de video

Existen muchos trabajos en el área de la codificación de video que hacen que la transcodificación sea un proceso cada vez más eficiente. Vetro et al en [Vetro03] hacen un recopilado de información acerca de las arquitecturas existentes para la transcodificación. Para usos prácticos se deja abierto el tipo de transcodificador que se usa, esto hace que la arquitectura provea flexibilidad y extensibilidad al no restringir a un tipo de transcodificador.

Nomenclatura: En el resto de la tesis para referirnos a un transcodificador de una clase x a una clase y, usaremos el término "x>y" donde "x" es la clase mayor u origen y "y" es la clase menor o destino.

3.2.4 Conexión y ubicación de la transcodificación.

Creación del árbol multicast

El problema relacionado con la creación de árboles multicast óptimos es conocido como árbol de Steiner (*Steiner Tree Problem*). Este problema es NP-completo ya que la obtención de un árbol óptimo es posible pero cuando aumenta el número de nodos la complejidad computacional del algoritmo aumenta de forma muy rápida. Para disminuir la complejidad de los algoritmos que solucionan problemas NP-completos se utilizan heurísticas. Las heurísticas disminuyen considerablemente la complejidad computacional pero se paga el costo de obtener un resultado sub-óptimo. El uso de una heurística ad hoc para cada problema, tiene resultados muy cercanos a los óptimos.

En [Duysburgh00] la creación de los árboles se hace de forma centralizada, es decir, un nodo obtiene información de la red y de los nodos para después generar el árbol multicast y obtener la localización más óptima posible de los transcodificadores. Los árboles multicast son creados de acuerdo a dos heurísticas SST y BA.

Skinned Steiner Tree Heuristic (SST): El algoritmo es dividido en dos pasos, cálculo del árbol y *Skinning the tree*.

Cálculo del árbol. Primero se calcula el árbol con el mínimo costo que conecte la fuente con todos los nodos finales, esto se hace mediante el algoritmo del camino más corto en un árbol (*shortest path tree)*. Después, se eliminan los enlaces que no tienen el suficiente ancho de banda para transportar la clase de video más alta. Si todos los nodos finales se encuentran en el árbol se hace el paso 2 (*skinning the tree*), sino se crea una lista con los destinos que quedaron fuera del árbol, la lista se ordena en base al ancho de banda solicitado. Una vez ordenada, se buscan caminos con recursos suficientes para soportar a los nodos en la lista. Los nodos que no son alcanzados en este paso son marcados como 'inalcanzables' y son eliminados para el resto de los cálculos, en este caso los usuarios finales son notificados.

Skinning the tree. En este paso se realiza la optimización del ancho de banda y la localización de los transcodificadores. Esto se hace recorriendo el árbol desde las hojas hasta la raíz.

Branch Attach Heuristic (BA): El algoritmo también se divide en dos pasos inicialización y agregar ramas.

Inicialización. Primero se calcula el árbol con el mínimo costo hacia todos los destinos que requieren la clase de video más alta, esto se hace utilizando el algoritmo *shortest path tree*. De esta forma se eliminan los enlaces con ancho de banda insuficiente para transportar la clase de video más alta. Después se reserva el ancho de banda necesario en los enlaces que se encuentran en el árbol.

Agregar ramas. En el segundo paso, los costos de las ramas que ya se encuentran en el árbol multicast son cambiados a cero para que tengan prioridad en el siguiente cálculo. Después se calcula el árbol de la siguiente clase de video inferior. Esto se repite hasta que se haya hecho para todas las clases pedidas por los usuarios. La localización de los transcodificadores se hace considerando los flujos de entrada y salida en los nodos. Si los flujos de entradas son distintos a los de salida entonces se instala un transcodificador en el nodo.

En [Lambrecht05] se muestra una implementación distribuida para el algoritmo BA. Un usuario que desea unirse a una sesión de video existente, envía una petición hacia el servidor de video. Si en el camino se encuentra con un nodo activo que ya pertenezca al árbol multicast de la sesión deseada se tienen tres situaciones:

a) La clase de video requerida es igual a alguna clase presente en el nodo. En este caso el nodo sólo replica el flujo hacia el nuevo usuario.
b) La clase requerida es menor que la clase de entrada. En este caso, el nodo realiza una transcodificación para adaptarse al nuevo flujo y lo envía al usuario.
c) La clase requerida es mayor que todas las clases con las que cuenta el nodo. En este caso, el nodo hace una petición hacia arriba en el árbol para pedir el nuevo flujo y poder enviarlo hacia el usuario.

Si en el camino hacia el servidor la petición de conexión no encuentra ningún nodo que contenga la sesión deseada, el usuario recibirá el flujo de video directamente desde fuente.

En [Duysburgh04] se muestra una implementación de un servicio de transcodificación en redes activas donde la conexión se realiza utilizando el algoritmo BA distribuido. El nodo final utiliza un mensaje llamado JOIN donde especifica el nombre de la sesión a la cual desea conectarse y la versión del códec que desea recibir. Esta petición es enviada hacia el servidor multicast situado en la raíz del árbol. Se examina la conexión de los nodos en dos sentidos (*downstream* y *upstream*).

Downstream set-up, en este caso el mensaje JOIN viaja desde el usuario final hasta la fuente. La fuente regresa un mensaje llamado JOIN_ACK, que es el que va formando el camino hacia el usuario final, además, en su trayecto se colocan los transcodificadores necesarios para obtener la tasa requerida.

Upstream set-up, en este caso el mensaje JOIN es quien forma el camino y coloca los transcodificadores necesarios del servidor hacia el usuario.

En [Duysburgh06] se estudia la estabilidad de los algoritmos para el establecimiento de los árboles multicast que se mencionaron anteriormente, se muestra que existe un problema cuando un nuevo usuario se une a una rama del árbol multicast que transporta una clase menor a la requerida, el

nodo activo que recibe la petición del usuario debe de pedir una clase mayor a su nodo padre y actualizar sus tablas para recibir la nueva clase de video, el problema es que si se eliminan o actualizan las entradas en la tabla de estados relacionadas con la antigua clase, los paquetes de la antigua clase que lleguen después de la actualización de la tabla no serán reenviados a sus destinos, haciendo que el usuario experimente un corte en la transmisión. Para resolver este problema en lugar de actualizar los registros en la tabla se agregan nuevos registros a la tabla con los cambios y se marca el campo tc de los registros antiguos con 'n' para evitar la pérdida momentánea de paquetes. Siendo borrados en un futuro mediante paquetes enviados periódicamente llamados J-CLN.

En [Khan02] el problema de la localización de los transcodificadores es centralizado, este es dividido en dos etapas. Primero se determinan los puntos de constricción o cuellos de botella, donde serán localizados los transcodificadores. Y en la segunda etapa, se realiza una búsqueda y selección para localizar los transcodificadores entre los vecinos de los puntos de constricción. Para realizar esto necesita de una vista general de la topología de la red así como de las descripciones de la calidad de la red en los enlaces y nodos involucrados. Una vez que se calculan los puntos de constricción y se localizan los transcodificadores obtienen un mapa en donde se especifica cómo los componentes deben estar conectados, a este mapa le llaman *component connection map*. Para el proceso de coordinación utilizan un componente centralizado llamado *Meta-Controller* (MC), el cuál es el encargado de la interacción con los usuarios, inicialización y desmantelamiento del sistema y de establecer la configuración del estado inicial.

Los métodos centralizados necesitan recolectar información de los enlaces y de los nodos en la red para formar un panorama global en uno de los nodos, aumentando el tiempo de conexión y localización. Además es necesario hacer el cálculo de los árboles multicast y la localización de los transcodificadores cada vez que un nodo desee incorporarse a la red, o restringir el servicio únicamente para los nodos conectados desde el inicio de la sesión. Los métodos distribuidos prescinden de centralizar la información, ya que, cada nodo guarda únicamente su estado y el estado de sus vecinos, la base de estos algoritmos se encuentra en la cooperación que existe entre los nodos para lograr un objetivo, haciéndolos más veloces y dinámicos. Un problema que tienen los métodos distribuidos es la inestabilidad al no tener un control total del sistema. La arquitectura propuesta utiliza un método distribuido para la creación del árbol multicast y para la localización de los transcodificadores. Al utilizar un método distribuido se busca ofrecer a los usuarios un menor tiempo de respuesta con relación a los métodos centralizados. Además la arquitectura propuesta utiliza el algoritmo distribuido BA (*Branch Attach*), por ser un algoritmo cuya estabilidad ya ha sido estudiada por otros autores.

En cuanto al sentido hacia el cual se localizan los transcodificadores en el árbol multicast, Duysburgh et al. [Duysburgh04] analizan las dos sentidos (*downstream* y *upstream*) y concluyen que los métodos varían debido a la asimetría de la red. Además señalan que *upstream set-up* es mejor en términos de tiempo de respuesta y complejidad, mientras que *downstream set-up* obtiene un árbol más parecido al óptimo, pero no difiere mucho con los obtenidos mediante *upstream set-up*. Basado en estos resultados, se propone el uso de un algoritmo *upstream set-up* para optimizar el tiempo de respuesta y la complejidad.

Conexión y localización de transcodificadores en la arquitectura propuesta.

El algoritmo de conexión utiliza los mensajes *join, join_ack* y *join_error.* Además de la variable *req_node* donde se almacenan los datos del usuario que desea conectarse.

Cuando un nuevo usuario desea conectarse a una sesión de video, manda un paquete de conexión (*join*) dirigido hacia el servidor de video, con el nombre de la sesión a la que desea conectarse en el campo *session* del paquete, y la clase de video que desea obtener en el campo *vclass* del paquete, esta clase es llamada *rclass* (*requested class*). En su camino hacia la fuente el paquete *join* visita varios nodos activos. El nodo activo procesa cada petición de forma diferente dependiendo del estado en que se encuentre. En la tabla 3.4 se muestra un resumen de los estados en los que un nodo puede estar a la llegada de un mensaje *join,* estos son definidos de acuerdo a los valores de las variables y de la tabla de estados. Enseguida se detallan los estados de la conexión.

Edo.	Existe *session*	Degrade node	Existe *rclass*	*rclass* < *up_class*	*rclass* > *up_class*	Transcodificador	Recursos
1	No	NA	NA	NA	NA	NA	NA
2	Si	Si	NA	NA	NA	NA	NA
3	Si	No	Si	NA	NA	NA	NA
4	Si	No	No	Si	NA	Si	Si
5	Si	No	No	Si	NA	Si	No
6	Si	No	No	No	Si	No	NA
7	Si	No	No	No	Si	Si	Si
8	Si	No	No	No	Si	Si	No

Tabla 3.4 Estados de la conexión. NA=No Aplica.

Nomenclatura: Para simplificar la nomenclatura de las entradas en la tabla de estados los valores de la entrada se escribirán de la siguiente manera y en el siguiente orden {s|dest|rc|dlc|dnc|tc|psh|tn|dn|dl} (vea la tabla 3.1).

Nomenclatura: En este trabajo se utilizan figuras para ejemplificar los estados del nodo. Las figuras muestran solamente la parte de la red necesaria para generar el estado. La red es mostrada en forma de árbol suponiendo que el servidor de video o nodo raíz se encuentra en la parte superior de la figura. Los nodos de la red son representados por círculos que contienen dentro la dirección del nodo. Los arcos punteados representan las conexiones físicas que no son parte del árbol multicast. Los arcos dirigidos representan los flujos de video que son enviados por el enlace y la dirección de la misma, el peso de los arcos es la clase del flujo de video que viaja por el enlace. Los mensajes son representados por el nombre del mensaje y el valor del campo *vclass* entre paréntesis (Ej. *join*(3)) y una flecha que indica el sentido en el que viaja el mensaje. Cuando un nodo está llevando a cabo una transcodificación esta es mostrada a un lado del nodo (Ej. 5>3).

Estado 1: El nodo activo no es parte del árbol multicast de la sesión deseada, por lo tanto no existe la sesión. Como muestra la figura 3.3 el nodo 2 al recibir el mensaje *join* del nodo 3 no tiene la sesión de video deseada, entonces el nodo 2 intenta unirse a la sesión enviando un nuevo paquete *join* al nodo 1 solicitando la clase de video que necesita el nodo 3. Los datos del nodo 3 son almacenados en la variable *req_node* y se espera la llegada de un paquete *join_ack* ó *join_error* cuyas acciones son

detalladas más adelante en esta misma sección. Cuando llega un mensaje *join_ack* se agrega la entrada {1|3|3|3|3|y|n|-1|-1|n} a la tabla del nodo 2 y se envía un mensaje *join_ack* al nodo 3.

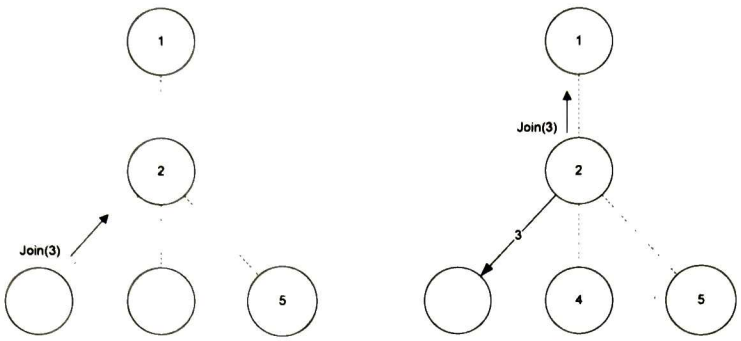


Figura 3.3 Llegada de un mensaje join. Estado 1.

Estado 2: El nodo activo es parte del árbol multicast de la sesión deseada y el enlace del nodo se encuentra degradado. En este caso el nodo seguirá recibiendo la clase de video degradada. Si la nueva clase pedida *rclass* es mayor a la clase antes pedida por el nodo en la tabla de estados *rc* se cambia el valor de *rc* por el de *rclass* en la tabla de estados. Si la clase nueva es igual a la clase de entrada *up_class* se actualiza el campo *tn* con un valor nulo representado por "-1". Finalmente se envía un mensaje *join_ack* para confirmar la conexión con la clase degradada en el campo *vclass* del paquete.

Estado 3: El nodo activo es parte del árbol multicast de la sesión deseada, el enlace no se encuentra degradado y el nodo contiene la clase pedida por el usuario. Como muestra la figura 3.4 el nodo 2 agrega al nodo 3 en su tabla de estados para que los paquetes de la clase de video que pidió sean replicados hacia su interfaz con los valores {1|3|3|3|3|y|n|-1|-1|n}. El nodo 2 envía un mensaje *join_ack* al nodo 3 para confirmar la conexión.

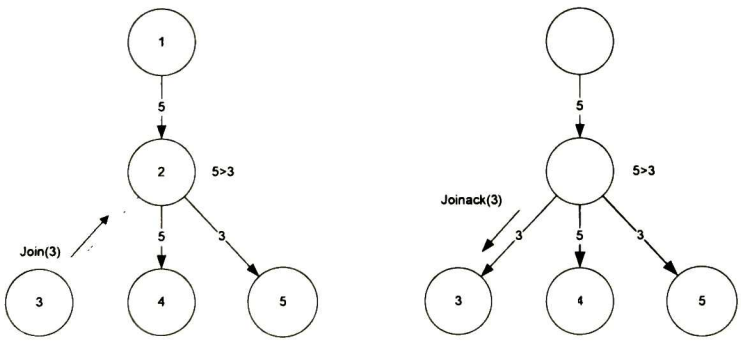


Figura 3.4 Llegada de un mensaje join. Estado 2.

Estado 4: El nodo activo es parte del árbol multicast de la sesión deseada, el nodo no cuenta con la clase pedida por el usuario, la clase pedida es menor a la clase de entrada *up_class* y el nodo cuenta con los recursos suficientes para llevar a cabo la transcodificación. Como muestra la figura 3.5 se carga el transcodificador *up_class>rclass* y se reservan los recursos necesarios para dicho transcodificador, se agrega el nodo 3 a la tabla de estados con la entrada {1|3|3|3|3|y|n|2|-1|n} y se envía un mensaje *join_ack* para confirmar la conexión.

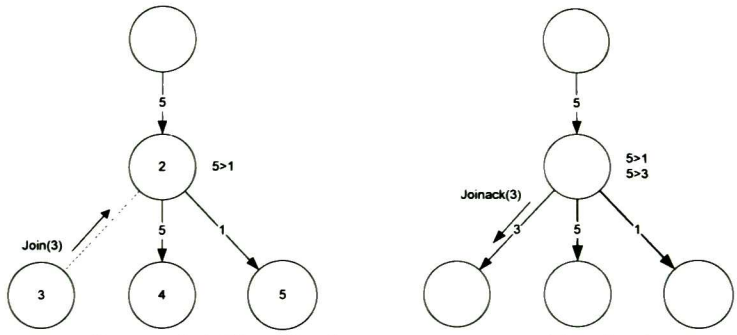


Figura 3.5 Llegada de un mensaje join. Estado 4.

Estado 5: El nodo activo es parte del árbol multicast de la sesión deseada, el nodo no cuenta con la clase pedida por el usuario, la clase pedida es menor a la clase de entrada *up_class* y el nodo no cuenta con los recursos suficientes para llevar a cabo la transcodificación. En este caso se lleva cabo la adaptación a los recursos del nodo activo que se detalla en la sección 3.2.5. Como muestra la figura 3.6 se envía un mensaje *ctrl-difreq* a todos los vecinos del nodo 2, y se espera por los mensajes de respuesta *ctrl-dify* y *ctrl-difn* de los vecinos.

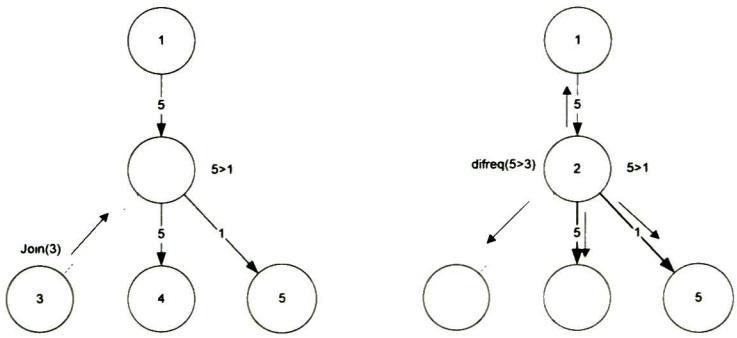


Figura 3.6 Llegada de un mensaje join. Estado 5.

Casos: Para los siguientes estados (6, 7 y 8) es necesario cambiar la clase de entrada *up_class* por una mayor enviando un mensaje *join* al nodo padre. Se tienen algunos casos que determinan la necesidad de agregar algún transcodificador extra. Tomando en cuenta que el usuario podría estar ya recibiendo alguna clase.

Caso 1: El usuario no está recibiendo ninguna clase. En este caso es necesario agregar un nuevo transcodificador.

Caso 2: El usuario que hace la petición está recibiendo la clase *up_class* y no es enviada a otros usuarios. En este caso no es necesario ningún transcodificador extra.

Caso 3: El usuario que hace la petición está recibiendo la clase *up_class* y si es enviada a otros nodos. En este caso es necesario cargar un transcodificador extra para no dejar de atender a los usuarios que están recibiendo *up_class* una vez que sea cambiada.

Caso 4: El usuario utiliza un transcodificador y ningún otro nodo lo utiliza. En este caso se libera el transcodificador que estaba utilizando el nodo y se carga uno nuevo para los usuarios que reciben *up_class.* Entonces el número de transcodificadores se mantiene.

Caso 5: El usuario que hace la petición utiliza un transcodificador y algún otro nodo lo utiliza. En este caso es necesario agregar un transcodificador extra para los usuarios que reciben *up_class.*

Estado 6: El nodo activo es parte del árbol multicast de la sesión deseada, el nodo no cuenta con la clase pedida por el usuario, la clase pedida es mayor a la clase de entrada *up_class,* el enlace de entrada no se encuentra degradado y no es necesario un transcodificador extra (casos 2 y 4). Como muestran las figuras 3.7 (a) y (b) se envía un mensaje *join* al nodo padre pidiendo el aumento de la clase, se guardan los datos del usuario en la variable *req_node* y se espera la llegada de un mensaje *join_ack* o *join_error* para cambiar el estado de la tabla. Cuando llega un mensaje join_ack se agrega la entrada {1|3|5|5|y|n|-1|-1|n} a la tabla del nodo 2 y se envía un mensaje *join_ack* al nodo 3.

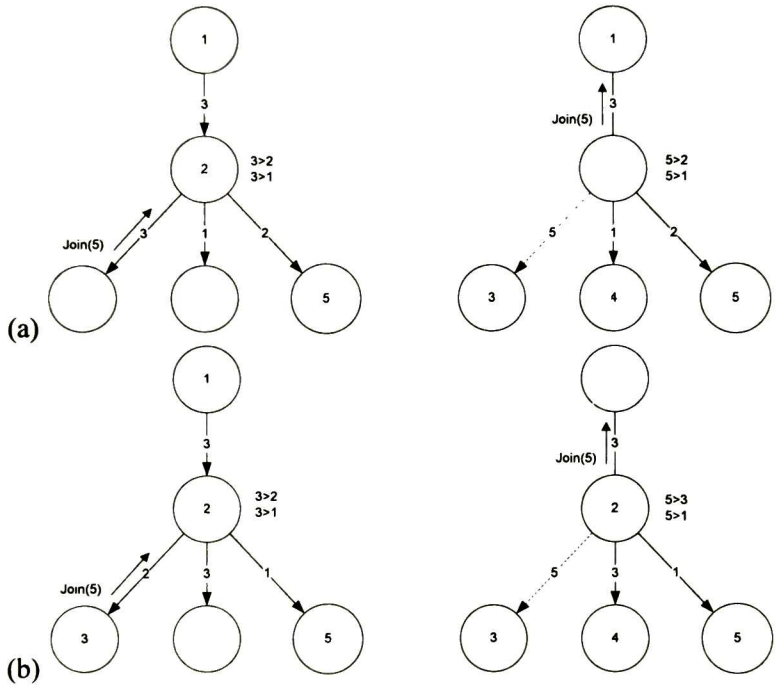


Figura 3.7 Llegada de un mensaje join. Estado 6. (a) Caso 2 y (b) Caso 4.

Estado 7: El nodo activo es parte del árbol multicast de la sesión deseada, el nodo no cuenta con la clase pedida por el usuario, la clase pedida es mayor a la clase de entrada *up_class,* el enlace de entrada no se encuentra degradado, es necesario un transcodificador extra (casos 1, 3 y 5) y el nodo cuenta con los recursos suficientes para cargar el transcodificador. En este estado se envía un mensaje *join* al nodo padre pidiendo el aumento de la clase, se guardan los datos del usuario en la variable *req_node,* se reservan los recursos, se carga el transcodificador y se espera la llegada de un mensaje *join_ack* o *join_error* para cambiar el estado de la tabla.

Estado 8: El nodo activo es parte del árbol multicast de la sesión deseada, el nodo no cuenta con la clase pedida por el usuario, la clase pedida es mayor a la clase de entrada *up_class,* el enlace de entrada no se encuentra degradado, es necesario un transcodificador extra (casos 1, 3 y 5) y el nodo no cuenta con los recursos suficientes para cargar el transcodificador. En este estado el nodo hace un simulacro de adaptación a los recursos sin cambiar el estado del nodo ni el de sus vecinos, una vez hecho el simulacro si existen recursos se envía un mensaje *join,* sino se busca una clase degradada que pueda ser enviada al usuario (Se detalla en la sección 3.2.5).

Llegada de un mensaje *join* al servidor de video.

En caso de que la petición llegue hasta el servidor de video, ya sea porque en el camino no se encontraron nodos activos que pudieran proveer la sesión de video deseada o porque la sesión de video no había sido iniciada por el servidor de video, el usuario recibirá el video "directamente" del servidor creando una nueva rama del árbol multicast. Si la sesión de video no ha sido iniciada el servidor inicia la transmisión de la nueva sesión, en caso de que el servidor no pueda iniciar la transmisión envía un mensaje *join_error* al usuario.

Mensajes *join_ack* y *join_error.*

En algunos de los estados el nodo activo espera por un mensaje *join_ack* o *join_error,* cuando llega un mensaje *join_error* al nodo significa que los nodos padres no fueron capaces de atender la petición de conexión, en este caso el nodo debe de reenviar dicho mensaje de error al usuario que hizo la petición cuyos datos se encuentran almacenados en la variable *req_node.* Cuando llega un mensaje *join_ack* al nodo se debe de actualizar la variable *up_class* con la nueva clase de video que será recibida contenida en el campo *vclass* del mensaje. Se actualiza la variable *up_node* con la dirección del nodo fuente. Se agrega a la tabla de estados una entrada correspondiente al nuevo usuario o se actualiza su entrada con la nueva clase que va a recibir en caso de que ya estuviese conectado, los valores del usuario se encuentran almacenados en la variable *req_node.* Se buscan en la tabla de estados del nodo los usuarios afectados por el cambio de la clase de video de entrada, los usuarios afectados son donde la clase pedida *rc* sea menor a *up_class* y el campo *tn* sea -1 y se actualizan marcando el campo *tn* con la dirección del nodo actual. Todos los transcodificadores usados son actualizados para adaptar el nuevo flujo de entrada a los flujos de salida y se envía un mensaje *join_ack* al nodo que solicito la clase de video. En caso de que sea necesario agregar un transcodificador y el nodo activo no cuente con los recursos necesarios para hospedarlo (estado 8) se inicia la etapa de adaptación a los recursos del nodo activo detallada en la sección 3.2.5.

Con éste método se colocan los transcodificadores necesarios al mismo tiempo que se realiza la conexión y el establecimiento del árbol multicast de forma distribuida permitiendo que los nodos se puedan conectar en cualquier instante a alguna sesión de video.

3.2.5 Adaptación a los recursos de los nodos y de los enlaces.

Recursos en los nodos activos.

Los recursos de los nodos activos se pueden caracterizar de dos formas dependiendo de las diferencias entre los transcodificadores. La primera, si los transcodificadores que son cargados por la aplicación activa requieren una cantidad de recursos similares, en este caso se pueden caracterizar los recursos de acuerdo al número de transcodificadores que un nodo activo puede albergar, cuando un transcodificador es cargado en un nodo activo se disminuye en uno el número de transcodificadores que se pueden albergar. Y la segunda, si los transcodificadores varían en el número de recursos que necesitan, en este caso se puede definir una unidad básica para los recursos de los nodos y cuantificar los recursos del nodo con dicha medida. En cada nodo activo se mantiene una tabla con la relación de recursos que cada transcodificador necesita, así, cuando un transcodificador se cargue en un nodo activo se disminuyen los recursos de acuerdo a la tabla. Las caracterizaciones anteriores necesitan que el número de recursos que utiliza un transcodificador sea conocido a priori.

Adaptación a los recursos de los nodos finales.

La adaptación a los recursos de los nodos finales es hecha por medio de la transcodificación del video original, basándose en las peticiones de los usuarios. Se asume que los nodos finales conocen sus limitantes para el procesamiento de video y conocen las clases de video que se ofrecen. De esta manera, pueden pedir la clase de video que más se acerque a sus capacidades.

Adaptación a los recursos de los nodos activos.

Khan et al en [Khan02] realizan la adaptación a los recursos de un nodo basados en dos niveles: optimización de la computación y difusión de la computación. En la *Optimización de la computación* se pierde cierta calidad del video a cambio de disminuir la complejidad computacional del transcodificador. Se utiliza un modulo que se habilita cuando se detecta que los recursos del nodo son insuficientes, este modulo calcula los nuevos vectores de movimiento directamente de los vectores del flujo de entrada. En la *Difusión de la computación,* si el nodo activo es incapaz de realizar la transcodificación, todos o algunos de los componentes del transcodificador son reubicados en algún nodo vecino.

Lambrecht et al en [Lambrecht05] realizan una búsqueda de nodos con capacidad de procesamiento insuficiente para realizar la transcodificación, los nodos encontrados son aligerados migrando los transcodificadores a los nodos vecinos. Este proceso se realiza en una etapa llamada post-optimización llevada a cabo después del establecimiento del árbol multicast. Para migrar los transcodificadores proponen tres métodos: *Push upstream*, en donde el transcodificador es enviado hacia el nodo activo padre del árbol multicast, *push downstream*, en donde el transcodificador es enviado hacia un nodo activo hijo en el árbol multicast y *push to proxy*, en donde el transcodificador es enviado hacia algún nodo activo adyacente que puede o no encontrarse en el árbol multicast.

En [Duysburgh04] se realiza la verificación de los recursos del nodo durante el establecimiento del árbol multicast y la localización de los transcodificadores. También se presenta una comparativa entre los tres métodos para la difusión de la computación. En sus simulaciones prueban los tres métodos por separado, donde los transcodificadores pueden ser migrados cualquier número de saltos en

la red pero en un sentido a la vez. Los resultados señalan que *push upstream* es el método con menor aumento en el uso de ancho de banda de la red, mientras que *push downstream* es la que mayor aumento presenta. En cuanto al número de transcodificadores en el árbol multicast los métodos de *push to proxy* y *push upstream* tuvieron resultados muy semejantes con un número menor de transcodificadores que *push downstream*.

El realizar la verificación de los recursos del nodo durante el proceso de conexión, permite iniciar automáticamente los mecanismos de adaptación en caso de que no existan los recursos necesarios. Con la técnica de optimización de la computación se optimiza el proceso de transcodificación para disminuir la complejidad computacional pero se pierde cierta calidad en el video, esto no es deseado si se busca la satisfacción de los requerimientos del usuario, pero es bueno si es lo único que se puede ofrecer. La difusión de procesamiento es una técnica que no castiga la calidad del video, pero genera un mayor tráfico en la red.

Adaptación a los recursos de los nodos activos en la arquitectura propuesta.

La adaptación a los recursos de los nodos activos se realiza en dos niveles *difusión de la transcodificación* y *degradación de la clase*. La difusión puede ser realizada en cualquiera de los tres sentidos *upstream*, *downstream* y *to proxy* tratando de optimizar los recursos de los enlaces y de los nodos la elección del mejor vecino se realiza en dos niveles: cantidad de recursos de los vecinos y eficiencia del sentido de la difusión. A diferencia de los trabajos relacionados donde la difusión se realiza dando 'n' saltos en alguno de los sentidos, en la arquitectura presente se da un solo salto en cualquiera de los tres sentidos para facilitar el mecanismo de restablecimiento de los transcodificadores cuando se liberan recursos en los nodos. En caso de que no existan recursos en los nodos vecinos se pasa al segundo nivel de adaptación o *degradación de la clase*.

Difusión de la computación.

Cuando llega un paquete *join* o *join_ack* al nodo activo, se analiza si es necesario agregar un nuevo transcodificador de la clase de entrada *up_class* a la clase pedida por el usuario *rclass*. Si es necesario transcodificar se verifica si el nodo tiene los recursos suficientes para llevar a cabo la transcodificación. Si el nodo no tiene los recursos suficientes, envía mensajes de petición de difusión (*ctrl-difreq*) a todos sus vecinos almacenados en la variable *neighbours* e inicializa las variables *ack*, para contar el número de respuestas recibidas de los vecinos, *candidate* para guardar el mejor candidato para la difusión y *node_tc* con los datos del nodo que va a ser transcodificado para saber el sentido en el que se va a realizar la difusión. Al recibir un mensaje *ctrl-difreq* el nodo se verifica si tiene los recursos necesarios para albergar el transcodificador y envía una respuesta ya sea positiva o negativa (*ctrl-dify* o *ctrl-difn*) al nodo que hizo la petición. Los mensajes *ctrl-difn* simplemente aumentan el contador *ack* y son descartados, mientras que los mensajes *ctrl-dify* contienen información acerca de los recursos y el sentido en que se movería la transcodificación hacia el nodo vecino, la información se compara con la variable *candidate*, si el nuevo vecino es mejor entonces se almacenan sus datos en la variable *candidate* asegurando que el mejor de los vecinos se encuentre en dicha variable. Una vez que el nodo recibe todas las respuestas el nodo vecino es notificado por medio de un mensaje de confirmación (*ctrl-difack*) para que inicie la transcodificación. La difusión de la transcodificación se da en los estados 5 y 8 de la conexión mencionados en la sección 3.2.4.

Para la elección del vecino al cual va a ser difundido el transcodificador se pueden utilizar varias métricas, nuestra propuesta es escoger primero al nodo con más recursos para no sobrecargar a los nodos con menos recursos y en el caso que existan nodos con recursos similares se escoge el mejor sentido para la difusión de acuerdo a los resultados mostrados en [Duysburgh04], siendo el orden de prioridad, primero *up*, después *proxy* y por último *down*.

Para conocer el sentido del nodo vecino la lista de vecinos llamada *neighbours* contiene un campo para almacenar el sentido del nodo. Cuando se recibe un mensaje *join* el nodo fuente se marca con el sentido *down* y cuando se recibe un mensaje *join_ack* el nodo fuente se marca con el sentido *up* en la lista de vecinos.

La variable *node_tc* almacena el nodo cuyo transcodificador va a ser difundido y puede ser igual a *req_node* (almacena el nodo que realiza la petición) si el nodo que necesita el transcodificador es el mismo que el que realiza la petición (estado 5 de la conexión) y es diferente cuando se pide otra clase mayor al nodo de arriba y el nodo que se va a transcodificar es diferente al que hizo la petición (estado 8 de la conexión).

Cuando el sentido del nodo *candidate* es *up* sin importar los valores de las demás variables el sentido es hacia arriba. Ya que por ser un árbol sólo existe un nodo padre y es común a todos los hijos. Cuando *node_tc* y *candidate* son iguales el sentido es hacia abajo. En la figura 3.8(a) *req_node*=3, *node_tc*=3 y *candidate*=3, así que el sentido en relación al nodo 3 es hacia abajo. Y en la figura 3.8 (b) *req_node*=3, *node_tc*=5 y *candidate*=5, así que el sentido en relación al nodo 5 es hacia abajo también.

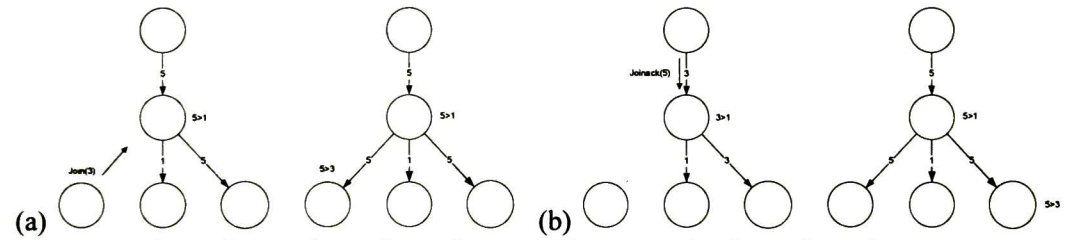


Figura 3.8 Difusión hacia abajo. (a) node_tc igual a req_node. (b) node_tc distinto a req_node.

Cuando *node_tc* y *candidate* son diferentes el sentido es hacia un lado. En la figura 3.9 (a) *req_node*=3, *node_tc*=3 y *candidate*=5, así que el sentido en relación al nodo 3 es hacia un lado. Y en la figura 3.9 (b) *req_node*=3, *node_tc*=5 y *candidate*=3, así que el sentido en relación al nodo 5 es hacia un lado también.

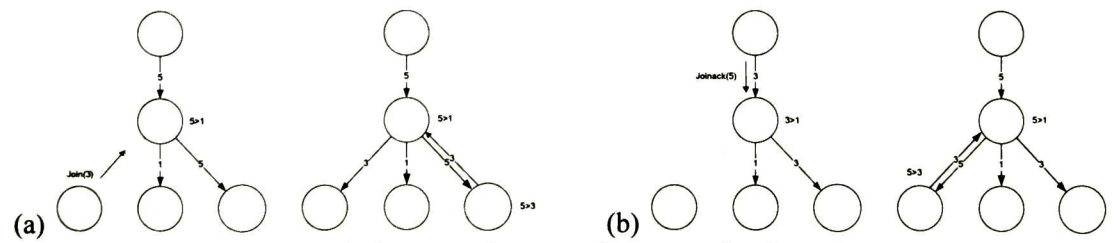


Figura 3.9 Difusión hacia un lado. (a) node_tc igual a req_node. (b) node_tc distinto a req_node.

Un caso especial se tiene cuando se cumple que *req_node* y *node_tc* son distintas y además *req_node* y *candidate* también son distintas. Como muestra la figura 3.10 donde *req_node*=3, *node_tc*=5 y *candidate*=4, la dirección para el nodo 4 es hacia abajo y para el nodo 5 es hacia un lado, a este caso lo llamamos hacia un lado y abajo.

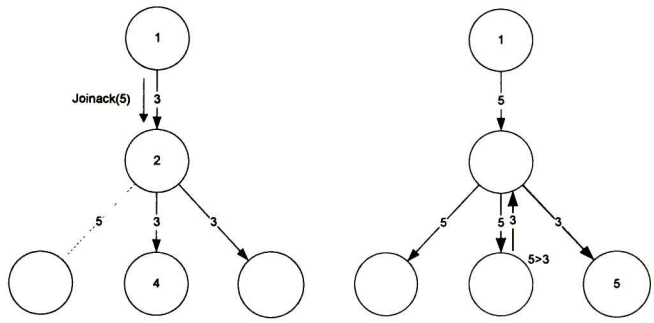


Figura 3.10 Difusión hacia un lado y abajo.

No es importante la relación entre *node_tc* y *req_node* para definir el sentido de la difusión. Pero con fines de decidir el envío de mensajes si es importante. Si *node_tc* y *req_node* son iguales y el sentido es hacia abajo se envía un mensaje *ctrl-difack,* marcado con cero y el sentido *down* en el campo de datos, hacia *node_tc*. Y si el sentido es hacia un lado, se envía un mensaje *ctrl-difack*, marcado con cero y el sentido *proxy*, hacia el nodo *node_tc* y un mensaje *join_ack* al nodo *req_node*. Cuando *node_tc* y *req_node* son diferentes y el sentido es hacia abajo se envía un mensaje *ctrl-difack,* marcado con uno y el sentido *down* en el campo de datos, hacia *node_tc* y un mensaje *join_ack* hacia *req_node*. Y si el sentido es hacia un lado, se envía un mensaje *ctrl-difack*, marcado con uno y el sentido *proxy*, hacia el nodo *node_tc* y un mensaje *join_ack* al nodo *req_node*.

Dependiendo del sentido donde se lleve a cabo la difusión del transcodificador es la modificación en la tabla de estados que realizan los nodos inmiscuidos en este proceso.

Hacia arriba, cuando el transcodificador es enviado hacia el nodo padre o lo que es lo mismo hacia arriba en el árbol multicast. Como muestra la figura 3.11(a) el nodo 2 que es el nodo sin recursos, este agrega la entrada {1|3|3|3|3|n|n|1|-1|n} y el nodo 1 el cual llevará a cabo la transcodificación agrega la entrada {1|2|3|3|3|y|y|1|-1|n}.

Hacia abajo, cuando el transcodificador es enviado al hijo que realizó la petición se modifica su entrada para que se le envíe la clase *up_class*. Como muestra la figura 3.11(b) el nodo 2 modifica la entrada del nodo 5 con {1|3|3|3|3|n|y|5|-1|n} para poder restablecer la clase y se agrega la entrada {1|3|5|5|5|n|n|3|-1|n}. Mientras que el nodo 3 agrega la entrada {1|x|3|3|3|y|y|1|-1|n} donde x es el nodo que pidió la clase.

Hacia los lados, cuando el transcodificador es enviado a un nodo hijo diferente al que hizo la petición o a algún nodo fuera del árbol multicast, como muestra la figura 3.11(c) el nodo 2 agrega la entrada {1|5|5|5|5|y|n|-1|-1|n} para enviar la clase *up_class* al nodo donde se realizará la transcodificación y la entrada {1|3|3|3|3|n|n|5|-1|n} para reenviar la clase al nodo que hizo la petición. Mientras que el nodo 5 agrega la entrada {1|2|3|3|3|y|y|5|-1|n}.

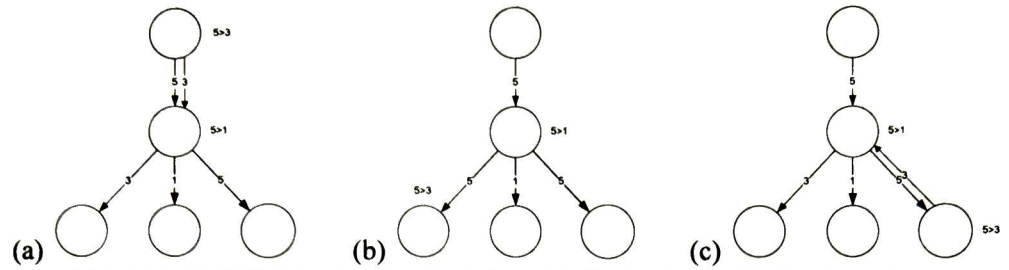


Figura 3.11 Difusión. (a) Hacia arriba. (b) Hacia abajo. (c) Hacia un lado.

Hacia un lado y abajo, cuando existe más de un nodo utilizando la clase que se va a transcodificar y la difusión se realiza hacia uno de estos nodos, como muestra la figura 3.10 el nodo 2 agrega al nodo *req_node* con la entrada {1|3|5|5|5|y|n|-1|-1|n}, actualiza la entrada del nodo 4 quien realizará la transcodificación para enviarle la clase *up_class* con la entrada {1|4|5|5|5|y|n|-1|-1|n} y actualiza las entradas de los demás nodos que utilizan la clase 3 en este ejemplo el nodo 5 con la entrada {1|5|3|3|3|n|n|4|-1|n}. Mientras que el nodo 4 agrega la entrada {1|2|3|3|3|y|y|4|-1|n}.

Cuando un mensaje *ctrl-difack* llega a un nodo se verifica el sentido y la bandera del mensaje, según el sentido se agregan las entradas correspondientes y se actualiza la clase de entrada *up_class*. Si el sentido del mensaje es *down* y la bandera es cero significa que el mensaje sirve como confirmación de la conexión así que se envía un mensaje *join_ack* a *req_node*. Si la bandera es uno no se envía el mensaje pues el nodo que hizo la petición se encuentra en otra rama. Si el sentido es *proxy* y la bandera es uno se envía un mensaje *join_ack* al nodo *req_node* ya que significa que *node_tc* y *req_node* son el mismo y el mensaje sirve como confirmación de la conexión. Si el sentido es *proxydown* y *up* no se envían mensajes.

Degradación de la clase.

Cuando no se encuentra ningún candidato para realizar la difusión de la transcodificación, esto es que todas las respuestas de los vecinos son negativas (*ctrl-difn*), se verifica si existen en el nodo clases menores a la pedida por el usuario, si existen, se escoge la mayor y se transmite hacia el usuario para reducir la probabilidad de no recibir un flujo. Se actualiza el campo *dnc* de la tabla de estados con la clase degradada y *dn* con la dirección del nodo actual y se envía un mensaje de degradación *ctrl-node_deg* al nodo afectado, este mensaje de degradación sirve también como confirmación de la conexión. En caso de que no existan clases menores para enviar se envía un mensaje de error *join_error* al usuario, siento un intento fallido de conexión.

Si un mensaje *ctrl-node_deg* llega a un nodo activo se actualiza la clase de entrada *up_class* con la nueva clase de entrada, se actualizan las entradas de los nodos afectados por la degradación marcando los campos *dnc* con la nueva clase y *dn* con la dirección del nodo fuente y se envía un mensaje *ctrl-node_deg* para avisar a los nodos afectados del cambio.

El algoritmo de localización coloca los transcodificadores lo más cerca posible del servidor de video, esto hace que el flujo del video original, que es el que tiene la mayor tasa de bits, no viaje más de lo necesario, disminuyendo el tráfico en la red. Cuando un transcodificador es migrado a otro nodo su nueva localización no es la más óptima, para mantener la localización más óptima posible se realiza

una readaptación a la liberación de recursos que no se realiza en los demás trabajos relacionados.

Liberación de recursos y restauración.

La arquitectura propone un mecanismo para restaurar los cambios hechos por los mecanismos de adaptación, donde primero se restauran los nodos cuyas clases se encuentran degradadas aumentando su clase de video a la deseada y después, si es posible, se relocalizan los transcodificadores que han sido difundidos hacia otros nodos.

Cada vez que se liberan recursos en el nodo activo por la desconexión de un nodo activo, primero se verifican los campos *dnc* y *dn* de la tabla de estados en busca de nodos degradados por la falta de recursos del nodo, ya que estos usuarios se encuentran recibiendo una clase menor a la que pidieron. Si existe alguno se aumenta su clase de video y se actualizan las entradas de los nodos afectados cambiando el campo *dnc* con la nueva clase, si la clase *dnc* es igual a *rc* el campo dn se marca con -1. El restablecimiento se difunde por medio de un mensaje *ctrl-node_res.*

En caso de que no existiera ningún nodo degradado entonces se buscan en la tabla de estados transcodificadores migrados a otros nodos donde el campo *tn* sea diferente al nodo actual. Si se encuentran y además los recursos liberados son suficientes el transcodificador regresa al nodo donde debería estar, se envía un mensaje de restablecimiento al nodo vecino (*ctrl-tc_res*) y se actualizan las entradas en la tabla cambiando el valor del campo *tn* por -1 si va a recibir la clase de entrada o por la dirección del nodo actual si va a utilizar un transcodificador. Si existiera más de un transcodificador migrado se da prioridad a los migrados por el método *push downstream,* después por *push to proxy* y por último por *push upstream.* Lo anterior es en orden contrario al de la difusión, para restablecer primero los transcodificadores migrados por métodos con menor desempeño.

Cuando un mensaje *ctrl-node_res* llega a un nodo activo, se actualiza la clase de entrada *up_class* con la nueva clase de entrada, se actualizan las entradas de los nodos afectados por la degradación marcando los campos *dnc* con la nueva clase y *dn* con una dirección nula si *dnc* es igual a *rc* y se envía un mensaje *ctrl-node_res* para avisar a los nodos afectados del cambio.

Cuando un mensaje *ctrl-tc_res* llega a un nodo activo, se borra de la tabla de estados al nodo fuente del paquete cuya clase *rc* corresponda con la clase *vclass* del mensaje y el valor del campo *psh* sea verdadero. Después de borrar la entrada se verifica si se está utilizando el transcodificador por algún otro usuario, sino se utiliza se libera el transcodificador y los recursos que tenia.

Adaptación a los recursos de los enlaces.

Khan et al en [Khan02] primero recolectan información global acerca de la calidad de la red (QoN) para obtener un mapa de la red marcando los enlaces donde existen cuellos de botella. Después colocan transcodificadores cerca de los cuellos de botella para adaptarse a la capacidad de los enlaces. La adaptación se lleva a cabo disminuyendo la tasa de bits mediante un transcodificador que ellos proponen. Este método es centralizado y necesita de un mapa global de la red.

Kang et al en [Kang00] utilizan la tasa de paquetes perdidos para realizar la adaptación de un video por capas en una red activa. El nodo que recibe el video tiene un límite para la tasa de paquetes perdidos, si la tasa sobrepasa el límite envía una petición al nodo emisor pidiendo que le envíe menos capas del video disminuyendo así el tráfico en el enlace.

El trabajo de Khan sufre de centralización, por lo que la adaptación dinámica requiere de un cálculo continuo de la localización de los transcodificadores. En cambio la adaptación distribuida mediante la recolección de información de los enlaces locales provee una adaptación más dinámica a menor costo.

Adaptación a los recursos de los enlaces en la arquitectura propuesta.

Se pueden utilizar métricas de estimación como la tasa de paquetes perdidos, ancho de banda disponible, retardo, etc. Para proveer información a los nodos y llevar a cabo la adaptación. En [Prasad03] se proponen métodos de estimación de algunas métricas de desempeño de la red a través de un camino, mediante la generación de tráfico conocido y su análisis después de recorrer la red. En la arquitectura propuesta se mide la tasa de paquetes perdidos del video original por ser una métrica que puede ser medida sin la necesidad de agregar tráfico a la red.

Para medir la tasa de paquetes perdidos, los paquetes de video *data* son numerados por el servidor en el campo *seq* del paquete. Cada nodo contiene un contador *seq_num* de llegada y un contador para el reenvío de los mensajes *send_seq_num* que son inicializados cuando el nodo se conecta a la sesión de video, además de una variable *packet_loss* para contar los paquetes perdidos cada determinado tiempo. Enseguida se muestra el algoritmo utilizado para calcular la pérdida de paquetes.

```
if(seq > seq_num){
      if(seq_num != 0){
            if(packet_loss > 0){
                  packet_loss = packet_loss + (seq - seq_num);
            }
            else {
                  packet_loss++;
            }
      }
      seq_num = seq + 1;
} else if (seq == seq_num){
      seq_num++;
}
```

Para el envío de paquetes cada nodo reenumera sus paquetes de salida de acuerdo con el contador *send_seq_num* para que los siguientes nodos midan sólo la perdida en el enlace adyacente a ellos. La pérdida de paquetes total para un nodo final será la suma de los paquetes perdidos de cada uno de los nodos activos en el camino hasta el servidor.

Periódicamente los nodos activos verifican el valor de su variable *packet_loss* y la comparan con el límite guardado en la variable *top_limit*, si el valor de *packet_loss* supera el límite envían un reporte de tipo 0 en un mensaje *ctrl-rep* para que el nodo padre degrade la clase de video buscando disminuir la perdida de paquetes. Por otro lado si el valor de *packet_loss* es cero se envía un reporte de tipo 1 notificando que no existe pérdida de paquetes en el enlace y si el valor de *packet_loss* está entre cero y el valor de *top_limit* se envía un reporte de tipo 2.

Cuando un nodo activo recibe un mensaje *ctrl-rep* actúa de acuerdo al tipo de reporte que recibe. Si el reporte es de tipo 2 significa que la tasa de pérdida de paquetes se encuentra en un umbral aceptable y simplemente lo descarta. Si es de tipo 1 significa que la tasa de pérdida de paquetes es cero y que existe la posibilidad de aumentar (restaurar) la clase de video si se encuentra degradada. Si el reporte es de tipo 0 significa que existe congestión en el enlace y es necesario degradar la clase para disminuir la tasa de pérdida de paquetes. Existen varios estados a la llegada de un paquete *ctrl-rep*, enseguida se enumeran los que contempla la arquitectura.

Estado 1, degradación de la clase: El reporte es de tipo 0, primero se verifica si efectivamente el nodo fuente está recibiendo un flujo del nodo actual ya sea la clase de entrada o alguna clase transcodificada, si es así se calcula la clase degradada *deg_class* disminuyendo en uno la clase que recibe el nodo actualmente, si la clase que recibe el nodo fuente es la clase 1 entonces no se puede degradar más y se descarta la petición, sino entonces existen varios sub-estados que son mostrados en la tabla 3.5 y detallados enseguida.

Existe deg_class	Recibe up_class	Transcodificador	Otro	Recursos	Caso
Si	Si	No	NA	NA	1.1
Si	No	Si	No	NA	1.2
No	Si	No	NA	Si	1.3
No	Si	No	NA	No	1.4
No	No	Si	Si	Si	1.5
No	No	Si	Si	No	1.6
No	No	Si	No	NA	1.7

Tabla 3.5 Estados de reporte tipo cero. NA=No Aplica.

Estado 1.1: Existe en el nodo la clase *deg_class* y el usuario recibe *up_class*. Como se muestra en la figura 3.12 el nodo 2 modifica la entrada del nodo 3 por {1|3|4|3|4|y|n|2|-1|y} marcando el campo *dl*=y y cambiando la clase *dnl*=3 y actualizando la información de donde se lleva a cabo la transcodificación, envía un mensaje *ctrl-link_deg* avisando el cambio de la clase al nodo 3.

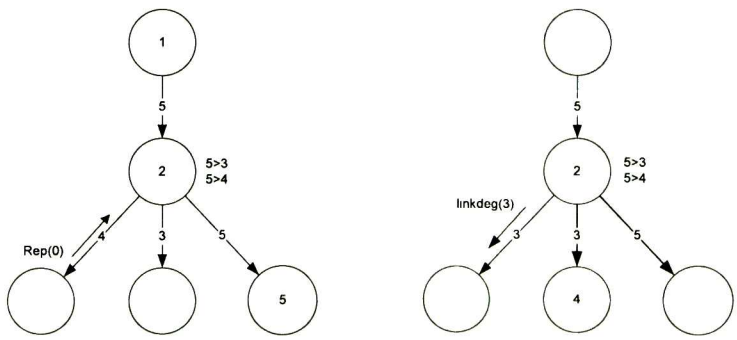


Figura 3.12 Llegada de un mensaje ctrl-rep. Estado 1.1.

Estado 1.2: Existe en el nodo la clase *deg_class*, el usuario utiliza un transcodificador y ningún otro nodo utiliza el transcodificador. Como se muestra en la figura 3.13 el nodo 2 modifica la entrada del nodo 3 por {1|3|4|3|4|y|n|2|-1|y} marcando el campo *dl*=y y cambiando la clase *dnl*=3 y actualizando la información de donde se lleva a cabo la transcodificación, envía un mensaje *ctrl-link_deg* avisando el cambio de la clase al nodo 3 y libera el transcodificador que no está en uso.

link_deg avisando el cambio de la clase al nodo 3 y libera el transcodificador que no está en uso.

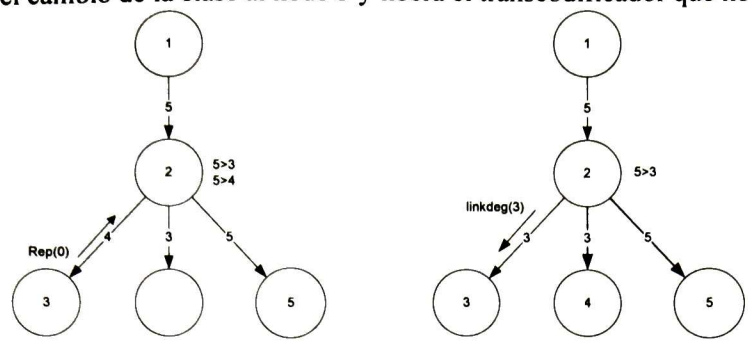


Figura 3.13 Llegada de un mensaje ctrl-rep. Estado 1.2.

Estado 1.3: No existe en el nodo la clase *deg_class*, el usuario recibe *up_class* y cuenta con recursos. Como se muestra en la figura 3.14 el nodo 2 carga un nuevo transcodificador y modifica la entrada del nodo 3 por {1|3|4|3|4|y|n|2|-1|y} marcando el campo *dl*=y y cambiando la clase *dnl*=3 y actualizando la información de donde se lleva a cabo la transcodificación, envía un mensaje *ctrl-link_deg* avisando el cambio de la clase al nodo 3.

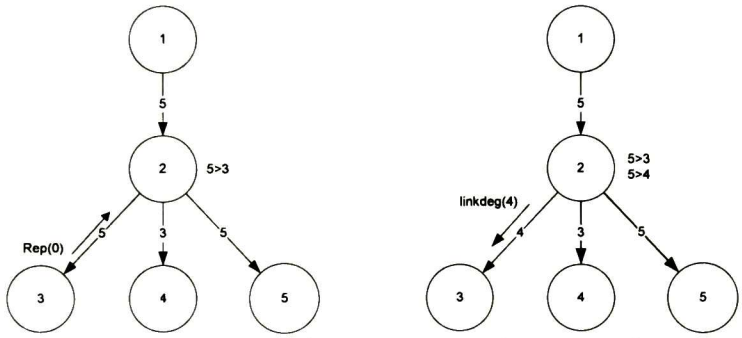


Figura 3.14 Llegada de un mensaje ctrl-rep. Estado 1.3.

Estado 1.4: No existe en el nodo la clase *deg_class*, el usuario recibe *up_class* y no cuenta con recursos. Como muestra la figura 3.15 no es posible degradar la clase y simplemente se ignora el reporte.

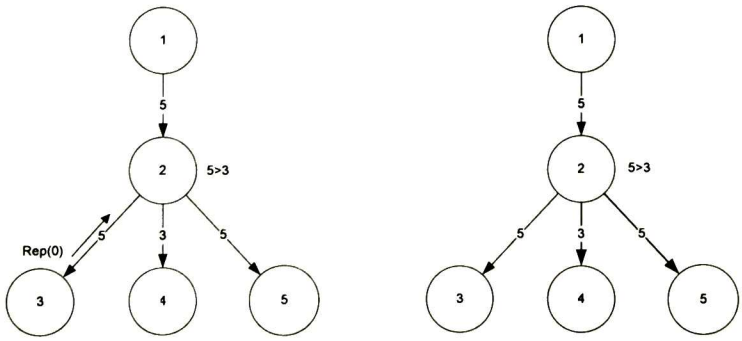


Figura 3.15 Llegada de un mensaje ctrl-rep. Estado 1.4.

Estado 1.5: No existe en el nodo la clase *deg_class*, el usuario utiliza un transcodificador, algún otro usuario utiliza el transcodificador y cuenta con recursos. Como muestra la figura 3.16 el nodo 2 carga un nuevo transcodificador y modifica la entrada del nodo 3 por {1|3|3|2|3|y|n|2|-1|y} marcando el campo *dl*=y y cambiando la clase *dnl*=2 y actualizando la información de donde se lleva a cabo la transcodificación, envía un mensaje *ctrl-link_deg* avisando el cambio de la clase al nodo 3.

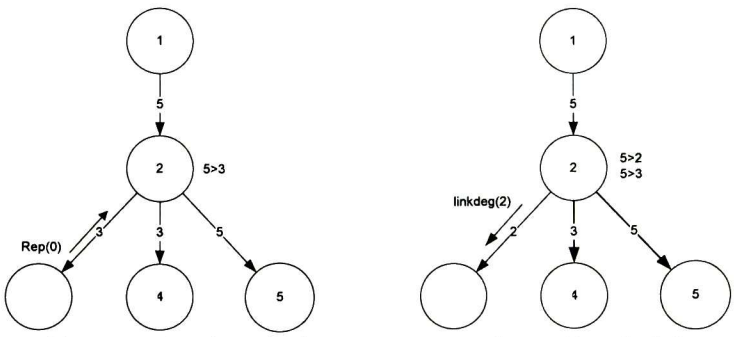


Figura 3.16 Llegada de un mensaje ctrl-rep. Estado 1.5.

Estado 1.6: No existe en el nodo la clase *deg_class*, el usuario utiliza un transcodificador, algún otro usuario utiliza el transcodificador y no cuenta con recursos. Al igual que en el estado 1.4 no es posible degradar la clase y simplemente se ignora el reporte.

Estado 1.7: No existe en el nodo la clase *deg_class*, el usuario utiliza un transcodificador, y ningún otro usuario utiliza el transcodificador. Como muestra la figura 3.17 el nodo 2 libera el transcodificador que utilizaba y carga el nuevo transcodificador que necesita, modifica la entrada del nodo 3 por {1|3|3|2|3|y|n|2|-1|y} marcando el campo *dl*=y y cambiando la clase *dnl*=2, envía un mensaje *ctrl-link_deg* avisando el cambio de la clase al nodo 3.

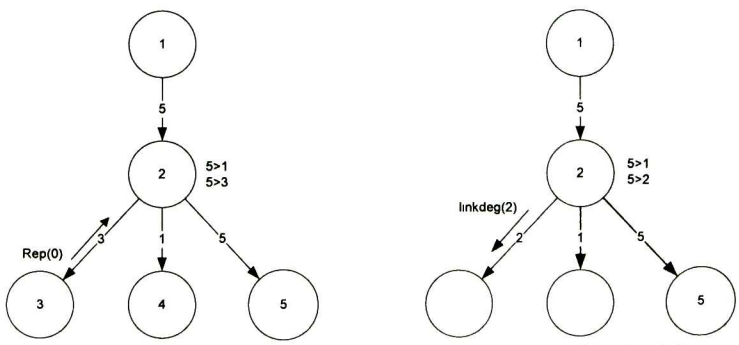


Figura 3.17 Llegada de un mensaje ctrl-rep. Estado 1.7.

Estado 2, restauración de la clase: El reporte es de tipo 1, en este estado se calcula la clase restaurada *res_class* aumentando en uno la clase que recibe el nodo actualmente y se verifica si el valor del campo *dl* es verdadero. Si la clase *res_class* es igual a la clase pedida por el usuario *rc*, se marca el campo *dl* con falso. Si es posible aumentar la clase se examinan varios sub-estados mostrados en la

tabla 3.6 y detallados enseguida.

Res_class = up_class	Otro usuario utiliza la clase antigua	Existe res_class	Recursos	Caso
Si	No	NA	NA	2.1
Si	Si	NA	NA	2.2
No	No	Si	NA	2.3
No	Si	Si	NA	2.4
No	NA	No	Si	2.5
No	NA	No	No	2.6

Tabla 3.6 Estados de reporte tipo uno. NA=No Aplica.

Estado 2.1: La clase *res_class* es igual a *up_class* y ningún otro usuario usa la clase que utilizaba el usuario. Como muestra la figura 3.18 el nodo 2 libera el transcodificador que utilizaba, se modifica la entrada del nodo 3 por {1|3|5|5|5|y|n|-1|-1|n} marcando el campo *dl*=n y cambiando la clase *dnl*=5, se envía un mensaje *ctrl-link_res* avisando el cambio de la clase al nodo 3.

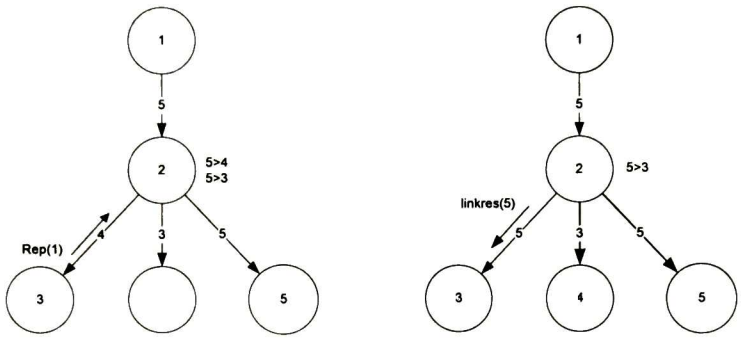


Figura 3.18 Llegada de un mensaje ctrl-rep. Estado 2.1.

Estado 2.2: La clase *res_class* es igual a *up_class* y algún otro usuario usa la clase que utilizaba el usuario. Como muestra la figura 3.19 en el nodo 2 se modifica la entrada del nodo 3 por {1|3|5|5|5|y|n|-1|-1|n} marcando el campo *dl*=n y cambiando la clase *dnl*=5, se envía un mensaje *ctrl-link_res* avisando el cambio de la clase al nodo 3.

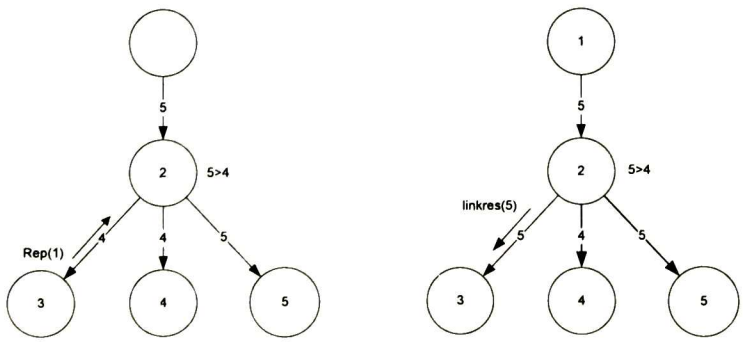


Figura 3.19 Llegada de un mensaje ctrl-rep. Estado 2.2.

Estado 2.3: La clase *res_class* no es igual a *up_class,* existe la clase *res_class* en el nodo y ningún otro usuario utiliza el transcodificador. Como muestra la figura 3.20 el nodo 2 libera el transcodificador que utilizaba, se modifica la entrada del nodo 3 por {1|3|4|4|4|y|n|-1|-1|n} marcando el campo *dl*=n y cambiando la clase *dnl*=4, se envía un mensaje *ctrl-link_res* avisando el cambio de la clase al nodo 3.

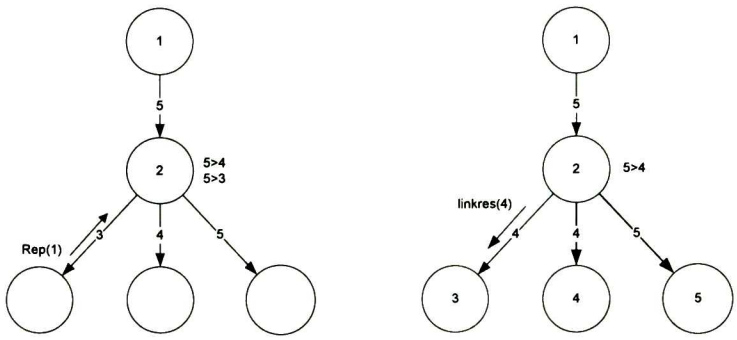


Figura 3.20 Llegada de un mensaje ctrl-rep. Estado 2.3.

Estado 2.4: La clase *res_class* no es igual a *up_class,* existe la clase *res_class* en el nodo y algún otro usuario utiliza el transcodificador. Como muestra la figura 3.21 en el nodo 2 se modifica la entrada del nodo 3 por {1|3|4|4|4|y|n|-1|-1|n} marcando el campo *dl*=n y cambiando la clase *dnl*=4, se envía un mensaje *ctrl-link_res* avisando el cambio de la clase al nodo 3.

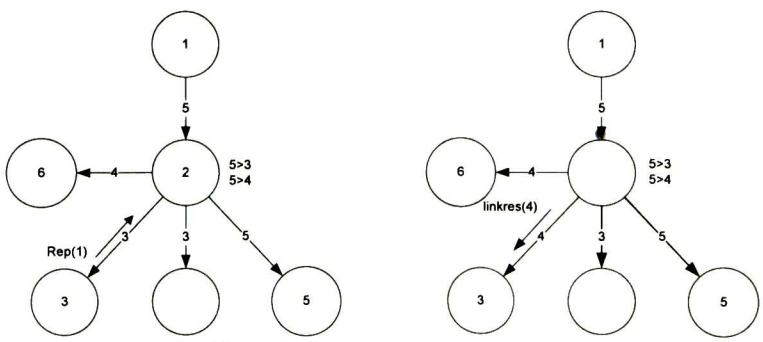


Figura 3.21 Llegada de un mensaje ctrl-rep. Estado 2.4.

Estado 2.5: La clase *res_class* no es igual a *up_class*, no existe la clase *res_class* en el nodo y existen recursos para transcodificar. Como muestra la figura 3.22 en el nodo 2 se carga el transcodificador que necesita, se modifica la entrada del nodo 3 por {1|3|4|4|4|y|n|-1|-1|n} marcando el campo *dl*=n y cambiando la clase *dnl*=4, se envía un mensaje *ctrl-link_res* avisando el cambio de la clase al nodo 3.

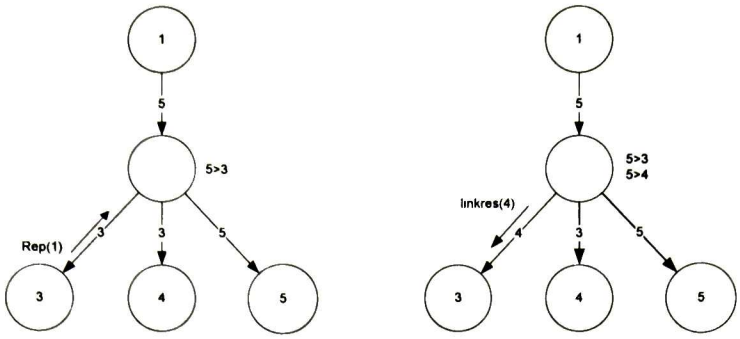


Figura 3.22 Llegada de un mensaje ctrl-rep. Estado 2.5.

Estado 2.6: La clase *res_class* no es igual a *up_class*, no existe la clase *res_class* en el nodo, no existen recursos para transcodificar. No es posible restaurar la clase y simplemente se ignora el reporte.

Estado 3: El reporte es de tipo 2. En este estado simplemente se ignora el reporte.

Mensajes *ctrl-link_deg* y *ctrl-link_res*.

Cuando un mensaje *ctrl-link_deg* llega a un nodo activo, se actualiza la variable *up_class* y se busca a los usuarios cuya clase enviada sea mayor a la nueva clase de entrada, estos son degradados a la clase *up_class*, el campo *dlc* toma el valor *up_class*.

Cuando un mensaje *ctrl-link_res* llega a un nodo activo, se actualiza la variable *up_class* y se busca a los usuarios cuya clase enviada se encuentre degrada debido a que su campo *rc* es mayor al *dlc*, estos son restaurados a la clase *up_class*, el campo *dlc* toma el valor *up_class*.

3.2.6 Desconexión y tolerancia a fallos de los nodos.

Duysburgh et al en [Duysburgh07] estudian la estabilidad de los árboles multicast para usuarios dinámicos que se conectan y desconectan aleatoriamente de la sesión de video utilizando el algoritmo propuesto por ellos en [Lambrecth06]. Cuando un usuario se desconecta de la sesión simplemente corta su rama, esto hace que el árbol multicast ya no sea el más óptimo, cuando otro usuario intenta conectarse se encuentra con un árbol sub-óptimo. Esto hace pensar en un método de desconexión más sofisticado.

Desconexión en la arquitectura propuesta.

La desconexión de un nodo puede ser notificada mediante un mensaje de desconexión (*leave*) o debida a algún error de conexión.

Cuando un usuario envía un mensaje de desconexión *leave* las entradas en la tabla de estados correspondientes al usuario cuyo valor en el campo *psh* sea falso son borradas. Si es necesario, se propaga el mensaje hacia arriba en el árbol dependiendo del estado en que se encuentre el nodo. Estos estados se presentan en la tabla 3.7 y son detallados enseguida.

Tabla vacía	Recibe up_class	Transcodificador	Otro nodo recibe la misma clase	Relación entre Degrade_class y big_class	Nodos degradados	Caso
Si	NA	NA	NA	NA	NA	1
No	Si	NA	No	>	NA	2.1
No	Si	NA	No	<	NA	2.2
No	Si	NA	No	=	NA	2.3
No	Si	NA	Si	NA	NA	3
No	NA	Si	No	NA	NA	4
No	NA	Si	Si	NA	Si	5.1
No	NA	Si	Si	NA	No	5.2

Tabla 3.7 Estados para la desconexión. NA=No Aplica

Estado 1: La tabla de estados se encuentra vacía. Como se muestra en la figura 3.23 el nodo 2 borra al nodo 3 de su tabla de estados y se desconecta de la sesión enviando un mensaje *leave* hacia su nodo padre en el carbol multicast.

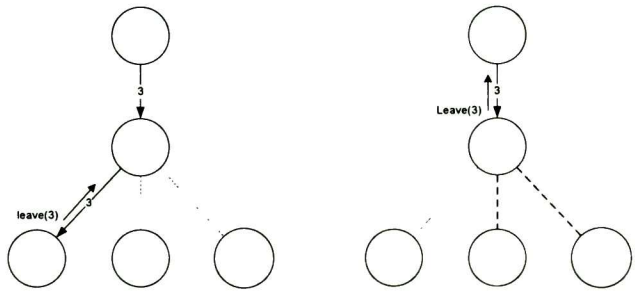


Figura 3.23 Llegada de un mensaje leave. Estado 1.

Estado 2: La tabla de estados no se encuentra vacía, el nodo liberado recibía *up_class* y ningún otro nodo la recibe. En este caso se tiene la posibilidad de actualizar la clase de entrada *up_class*, dando prioridad a los usuarios que reciben alguna clase degradada. La petición de cambio de clase se hace enviando un mensaje *ctrl-down_class* al nodo padre con la clase requerida en el campo *vclass*, una vez que se envía una solicitud se debe esperar un paquete *ctrl-down_classack* como confirmación para hacer los cambios en la tabla. Se tienen algunos sub-casos para decidir cuál clase será la que se va a pedir. Primero se busca la mayor clase enviada por el nodo (*big_class*) y la mayor clase degradada (*degrade_class*) en el nodo y después se comparan.

Estado 2.1: Si *degrade_class* es mayor que *big_class*, se pide la clase *degrade_class* al nodo padre con un mensaje *ctrl-down_class*. En este estado existen otros usuarios utilizando la misma clase que el nodo degradado. Como muestra la figura 3.24 el nodo 2 elimina las entradas del nodo 3 y actualiza la entrada del nodo 4 aumentando el campo *dnc* a 4 y finalmente se actualizan los transcodificadores para adaptar la nueva clase de entrada *up_class*.

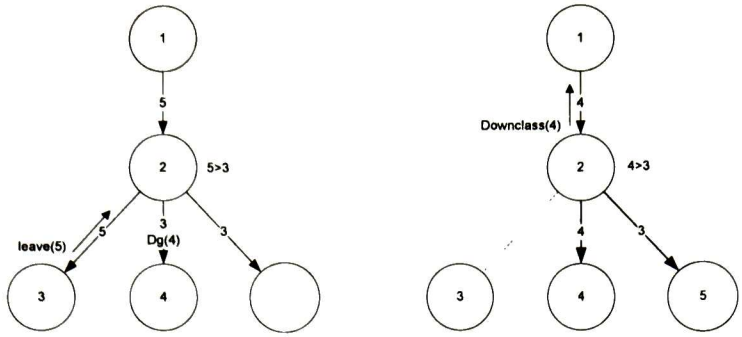


Figura 3.24 Llegada de un mensaje leave. Estado 2.1.

Estado 2.2: Si *degrade_class* es menor que *big_class*, se pide la clase *big_class* al nodo padre con un mensaje *ctrl-down_class*. En este estado se buscan nodos degradados, si no existen se libera el transcodificador que utilizaba el nodo que recibía *big_class*. Como muestra la figura 3.25 el nodo 2 elimina las entradas del nodo 3 y actualiza la entrada del nodo 5 aumentando el campo *dnc* a 2 y se actualizan los transcodificadores para adaptar la nueva clase de entrada *up_class*.

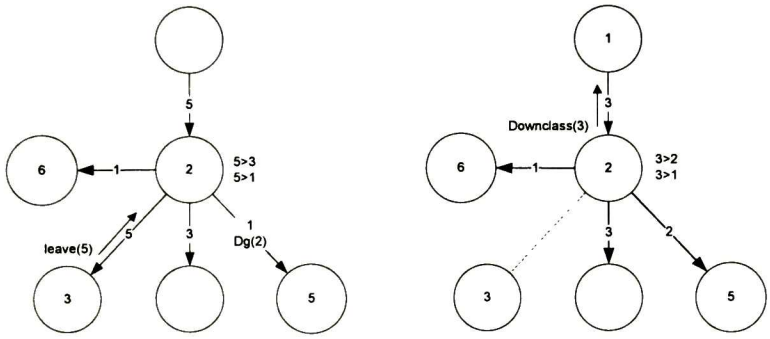


Figura 3.25 Llegada de un mensaje leave. Estado 2.2.

Estado 2.3: Si *degrade_class* es igual que *big_class*, se pide la clase al nodo padre con un mensaje *ctrl-down_class*. Como muestra la figura 3.26 el nodo 2 actualiza los transcodificadores para adaptar la nueva clase de entrada *up_class*.

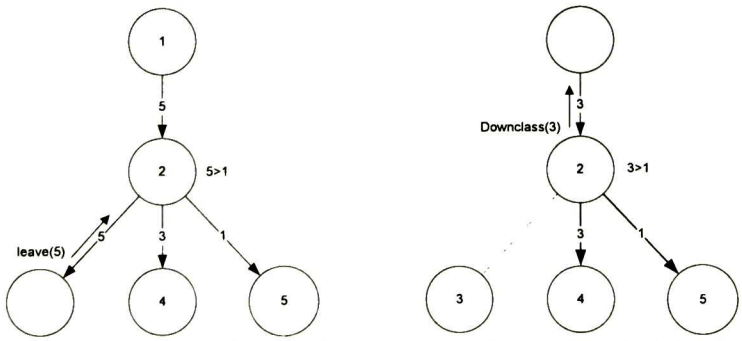


Figura 3.26 Llegada de un mensaje leave. Estado 2.3.

Estado 3: La tabla de estados no se encuentra vacía, el nodo liberado recibía *up_class* y algún otro nodo la recibe. Como se muestra en la figura 3.27 no se hace ningún cambio porque aún es necesario mantener la misma clase de entrada.

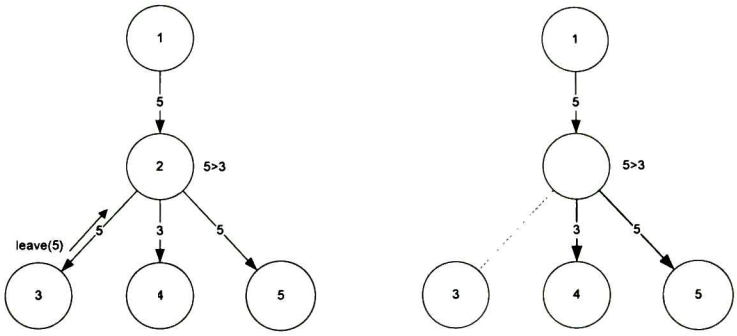


Figura 3.27 Llegada de un mensaje leave. Estado 3.

Estado 4: La tabla de estados no se encuentra vacía, el nodo liberado utiliza un transcodificador y ningún otro nodo lo utiliza. En este estado se buscan usuarios degradados para su restauración sino existen se libera el transcodificador local o migrado. En la figura 3.28 se restaura el nodo 5.

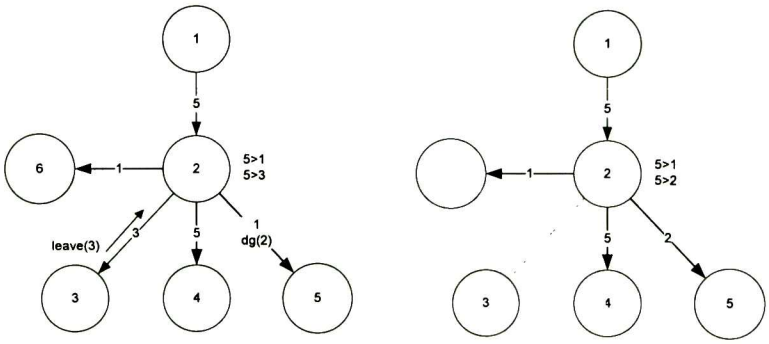


Figura 3.28 Llegada de un mensaje leave. Estado 4.

Estado 5: El nodo liberado utiliza un transcodificador y algún otro nodo lo utiliza. En este estado existen los siguientes sub-estados.

Estado 5.1: Los nodos que utilizan el mismo transcodificador se encuentran todos degradados. Como muestra la figura 3.29 se cambia el transcodificador para la clase degradada menor y se actualizan los demás nodos degradados con esta clase.

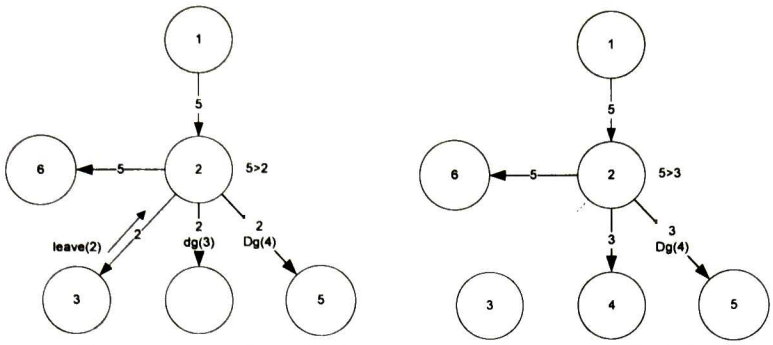


Figura 3.29 Llegada de un mensaje leave. Estado 5.1.

Estado 5.2: Al menos uno de los nodos que utilizan el mismo transcodificador no se encuentra degradado. Como se muestra en la figura 3.30 no se hace ningún cambio pues no existe liberación de recursos.

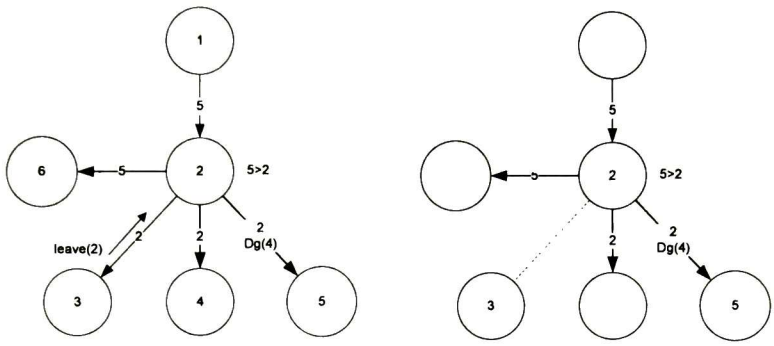


Figura 3.30 Llegada de un mensaje leave. Estado 5.2.

Mensaje *ctrl-down_class.*

Cuando una petición para disminuir la clase de video *ctrl-down_class* llega a un nodo activo se efectúan cambios en el nodo de acuerdo al estado en el que se encuentre el nodo, la tabla 3.8 muestra dichos estados que son detallados enseguida.

Recibe up_class	Transcodificador	Otro utiliza la misma clase	Existe la nueva clase.	Recursos	Relación entre degrade_class y vclass	Caso
Si	NA	Si	Si	NA	NA	1
Si	NA	Si	No	Si	NA	2
Si	NA	Si	No	No	NA	3
Si	NA	No	NA	NA	=	4.1
Si	NA	No	NA	NA	>	4.2
Si	NA	No	NA	NA	<	4.3
NA	Si	Si	Si	NA	NA	5
NA	Si	Si	No	Si	NA	6
NA	Si	Si	No	No	NA	7
NA	Si	No	Si	NA	NA	8
NA	Si	No	No	NA	NA	9

Tabla 3.8 Estados del nodo al recibir un mensaje ctrl-down_class. NA=No Aplica.

Estado 1: El nodo que hace la petición recibe *up_class*, algún otro nodo utiliza *up_class* y el nodo cuenta con la nueva clase. Como muestra la figura 3.31 el nodo 2 actualiza la entrada del nodo 3 cambiando el campo *rc*=3 y los campos *dnc=3* y *dlc=3* sólo si son mayores a 3 y se envía un mensaje *ctrl-down_classack* como confirmación.

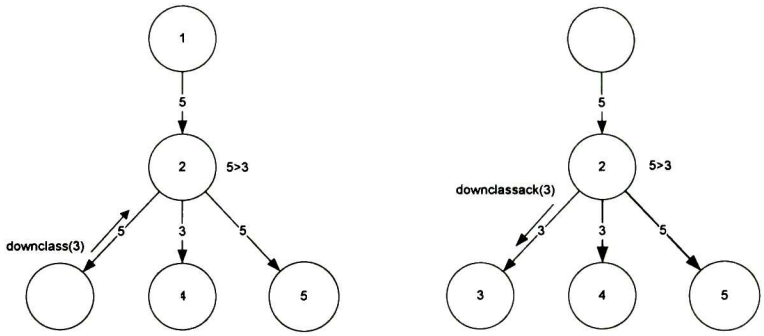


Figura 3.31 Llegada de un mensaje ctrl-down_class. Estado 1.

Estado 2: El nodo que hace la petición recibe *up_class*, algún otro nodo utiliza *up_class*, el nodo no cuenta con la nueva clase y tiene los recursos para llevar a cabo la transcodificación. Como muestra la figura 3.32 el nodo 2 actualiza el campo *rc*=3 y los campos *dnc* y *dlc* si son mayores a 3 de la entrada del nodo 3, agrega el nuevo transcodificador y actualiza el campo *tn*=2 correspondiente al nuevo transcodificador y se envía un mensaje *ctrl-down_classack* como confirmación.

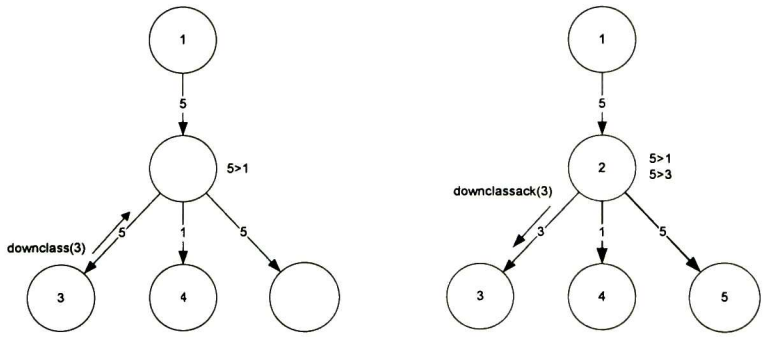


Figura 3.32 Llegada de un mensaje ctrl-down_class. Estado 2.

Estado 3: El nodo que hace la petición recibe *up_class*, algún otro nodo utiliza *up_class*, el nodo no cuenta con la nueva clase y no tiene los recursos para llevar a cabo la transcodificación. Como muestra la figura 3.33 no hay cambios y simplemente se envía un mensaje *ctrl-down_classack* con la clase 5.

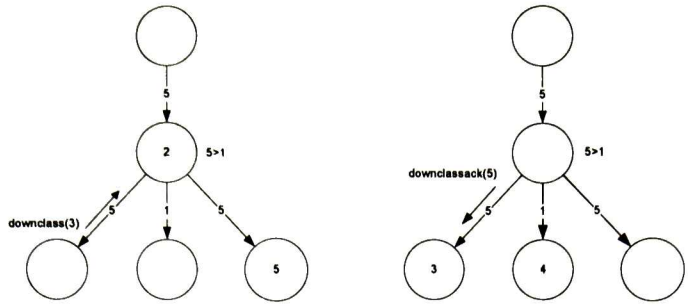


Figura 3.33 Llegada de un mensaje ctrl-down_class. Estado 3.

Estado 4: El nodo que hace la petición recibe *up_class* y ningún otro nodo la utiliza. En este caso se debe cambiar la clase *up_class* dando prioridad a los usuarios degradados. Para esto se tienen algunos sub-estados. Se busca la siguiente clase que se está enviando *down_class,* se busca la mayor clase degradada y se compara con *down_class,* la mayor de las dos se guarda en la variable *degrade_class* y se compara con la clase pedida por el usuario *vclass.*

Estado 4.1: Si la clase *degrade_class* es igual a *vclass*, se pide la clase al nodo padre con un mensaje *ctrl-down_class*. Como muestra la figura 3.34 los valores de *degrade_class*=4 y de *vclass*=4, únicamente se envía un mensaje *ctrl-down_class* al nodo 1 y se espera la respuesta.

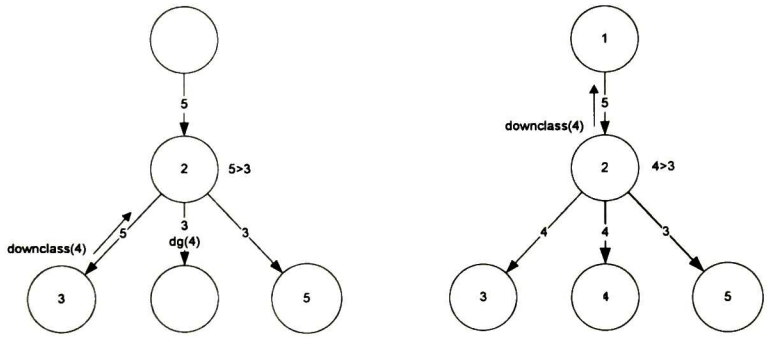


Figura 3.34 Llegada de un mensaje ctrl-down_class. Estado 4.1.

Estado 4.2: Si la clase *degrade_class* es mayor que *vclass*, se pide la clase *degrade_class* al nodo padre con un mensaje *ctrl-down_class*. Como muestra la figura 3.35 los valores de *degrade_class*=4 y de *vclass*=3, únicamente se envía un mensaje *ctrl-down_class* al nodo 1 y se espera la respuesta.

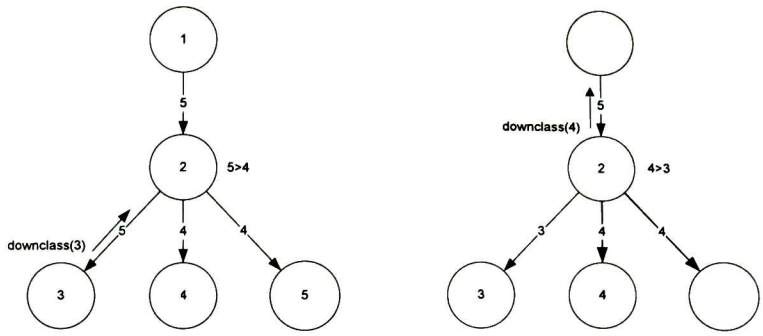


Figura 3.35 Llegada de un mensaje ctrl-down_class. Estado 4.2.

Estado 4.3: Si la clase *degrade_class* es menor que *vclass*, se pide la clase *vclass* al nodo padre con un mensaje *ctrl-down_class*. Como muestra la figura 3.36 los valores de *degrade_class*=2 y de *vclass*=4, únicamente se envía un mensaje *ctrl-down_class* al nodo 1 y se espera la respuesta.

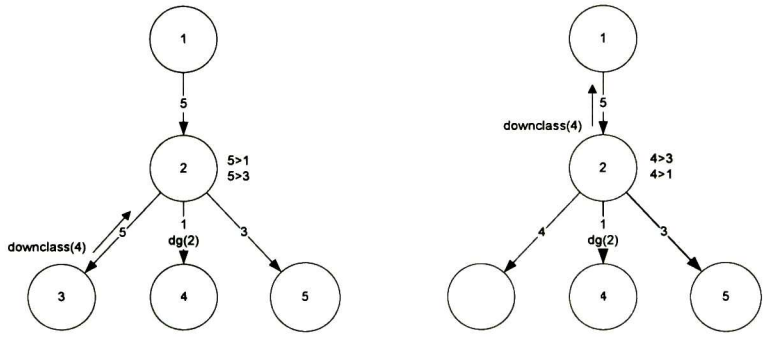


Figura 3.36 Llegada de un mensaje ctrl-down_class. Estado 4.3.

Estado 5: El nodo que hace la petición utiliza un transcodificador, algún otro nodo utiliza el transcodificador y el nodo cuenta con la nueva clase. Como muestra la figura 3.37 el nodo 2 actualiza el campo *rc*=3 y los campos *dnc* y *dlc* si son mayores a 3 de la entrada del nodo 3, además de los datos del transcodificador y envía un mensaje *ctrl-down_classack* como confirmación.

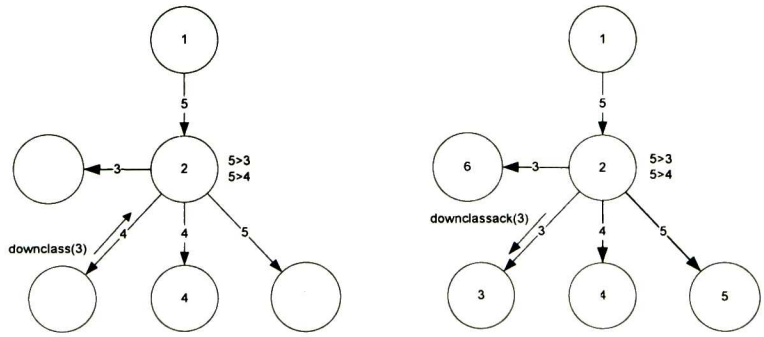


Figura 3.37 Llegada de un mensaje ctrl-down_class. Estado 5.

Estado 6: El nodo que hace la petición utiliza un transcodificador, algún otro nodo utiliza el transcodificador, el nodo no cuenta con la nueva clase y tiene los recursos para llevar a cabo la transcodificación. Como muestra la figura 3.38 el nodo 2 actualiza el campo *rc*=3 y los campos *dnc* y *dlc* si son mayores a 3 de la entrada del nodo 3, agrega el nuevo transcodificador y actualiza el campo *tn*=2 correspondiente al nuevo transcodificador y envía un mensaje *ctrl-down_classack* como confirmación.

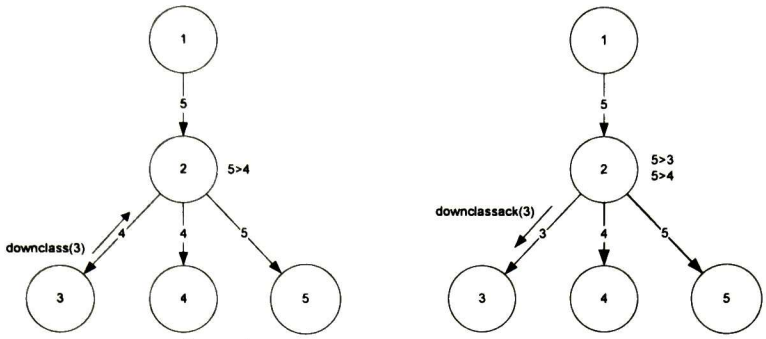


Figura 3.38 Llegada de un mensaje ctrl-down_class. Estado 6.

Estado 7: El nodo que hace la petición utiliza un transcodificador, algún otro nodo utiliza el transcodificador, el nodo no cuenta con la nueva clase y no tiene los recursos para llevar a cabo la transcodificación. Como muestra la figura 3.39 simplemente se envía un mensaje *ctrl-down_classack* con la clase 4.

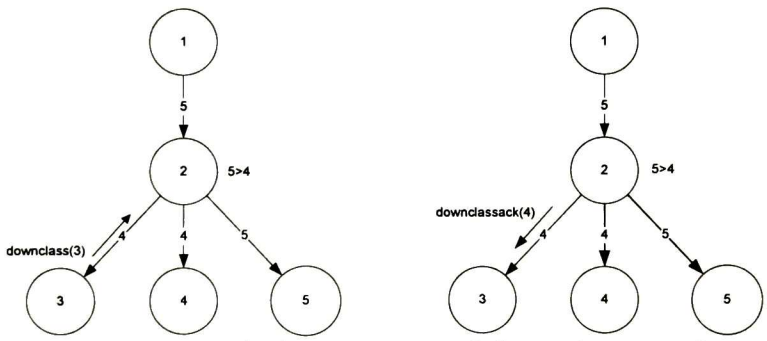


Figura 3.39 Llegada de un mensaje ctrl-down_class. Estado 7.

Estado 8: El nodo que hace la petición utiliza un transcodificador, ningún otro nodo utiliza el transcodificador y el nodo cuenta con la nueva clase. Como muestra la figura 3.40 el nodo 2 actualiza el campo *rc*=3 y los campos *dnc* y *dlc* si son mayores a 3 de la entrada del nodo 3, libera el antiguo transcodificador y actualiza el campo *tn*=2 correspondiente al nuevo transcodificador y envía un mensaje *ctrl-down_classack* como confirmación.

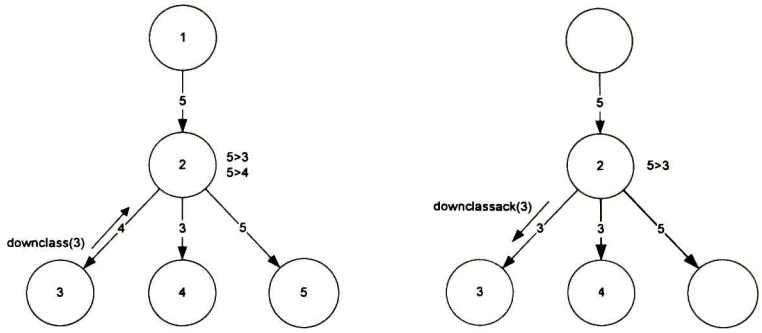


Figura 3.40 Llegada de un mensaje ctrl-down_class. Estado 8.

Estado 9: El nodo que hace la petición utiliza un transcodificador, ningún otro nodo utiliza el transcodificador y el nodo no cuenta con la nueva clase. Como muestra la figura 3.41 el nodo 2 actualiza el campo *rc*=3 y los campos *dnc* y *dlc* si son mayores a 3 de la entrada del nodo 3, libera el antiguo transcodificador y agrega el nuevo transcodificador y envía un mensaje *ctrl-down_classack* como confirmación.

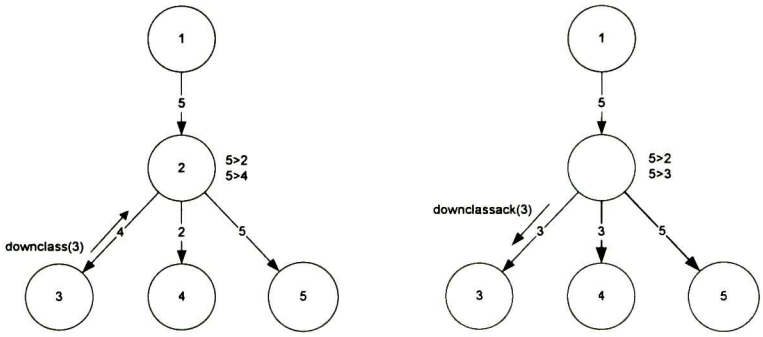


Figura 3.41 Llegada de un mensaje ctrl-down_class. Estado 9.

Mensaje *ctrl-down_classack.*

Cuando una confirmación de disminución de la clase de video *ctrl-down_classack* llega al nodo, se verifica si la nueva clase *vclass* es diferente a la clase *up_class* recibida anteriormente, si es diferente se actualiza el valor de *up_class* con la nueva clase *vclass,* se actualiza la tabla de estados y los transcodificadores para soportar la nueva clase de entrada. Se liberan recursos de acuerdo con los casos para la llegada de mensajes *ctrl-down_class.*

Desconexión por fallo.

Para saber cuando un nodo se desconecta debido a algún error, se utilizan los mensajes de reporte *ctrl-rep* que deben llegar periódicamente al nodo, si durante un periodo de tiempo no se han recibido reportes se puede asumir que el nodo esta desconectado y borrarlo como si hubiese llegado un mensaje de desconexión proveniente de dicho nodo.

Mientras que los nodos que se encuentra debajo de la falla en el árbol multicast se percatan de este porque dejan de recibir el video, de esta forma deben enviar un mensaje de desconexión al nodo de arriba y volver a intentar conectarse por otro camino.

3.3 Conclusiones

De acuerdo con lo descrito en este capítulo los nodos activos tienen un estado interno de acuerdo a sus variables y su tabla. El estado de un nodo determina la reacción del mismo ante la llegada de un mensaje. Los nodos activos actúan como agentes reactivos que reaccionan a un estímulo o entrada.

La arquitectura propuesta busca cumplir con los objetivos de un sistema distribuido, soluciona el problema de la heterogeneidad mediante la adaptación del flujo de video o transcodificación. El sistema puede ser extendido al número de sesiones y clases de video que se deseen, además deja abierta la posibilidad de utilizar cualquier tipo de transcodificador. En cuanto a seguridad la arquitectura propuesta depende de la arquitectura de red activa sobre la que se monte para su seguridad. El sistema es escalable de acuerdo a los recursos con los que cuenta la red y los nodos, entre más recursos más usuarios son soportados. Se provee un mecanismo de tolerancia a fallas de comunicación. Los usuarios únicamente necesitan conocer la dirección del servidor de video al cual se van a conectar siendo transparente para ellos el mecanismo y el lugar donde se realiza la transcodificación.

La arquitectura propuesta utiliza la transcodificación para adaptar un video que se encuentra en un servidor al formato que es requerido por los usuarios. Para aumentar la probabilidad de obtener un flujo de video para los usuarios ante la falta de recursos de los nodos activos para realizar la transcodificación la arquitectura propuesta incorpora dos niveles de adaptación, la difusión de la computación y la degradación de la clase. Además, toma en cuenta la posibilidad de que se liberen recursos en los nodos y utiliza un mecanismo de restauración para los dos niveles de adaptación. Para disminuir la pérdida de paquetes en la red debido a la falta de recursos en los enlaces la arquitectura propuesta utiliza un mecanismo basado en el envío de reportes periódicos para disminuir el flujo de video ante una saturación de los enlaces y la restauración del flujo en caso de liberación de los recursos.

A diferencia de otros trabajos la difusión de la transcodificación puede ser en cualquier sentido (*up*, *down* y *proxy*) buscando que sea el nodo que cuenta con más recursos el que lleve a cabo la transcodificación. Además para llevar a cabo la restauración de la difusión esta se lleva únicamente con los vecinos inmediatos y no con todos los nodos de la red con se hace en otros trabajos.

Para proporcionar un menor tiempo de conexión se utilizan algoritmos distribuidos para el establecimiento del árbol multicast y la localización de los transcodificadores. Esto permite que los escenarios sean dinámicos permitiendo la conexión y desconexión aleatoria de los usuarios.

CAPITULO 4

4 Formalización de la arquitectura.

Resumen: Un método muy útil para probar el correcto funcionamiento de un sistema es la formalización matemática. En este capítulo se muestra una formalización de la arquitectura propuesta utilizando el modelo de redes de Petri. Se opta por las redes de Petri sobre otros modelos matemáticos debido a que ofrece una forma gráfica sencilla para representar comportamientos complejos y un soporte matemático simple. Al principio del capítulo se hace una breve introducción a las redes de Petri basada en definiciones dadas por Mellado en [Mellado97]. Después se analizan las propiedades de acotamiento, libertad de bloqueos y vivacidad mediante la herramienta PIPE2 [wpipe] que son interesantes para el modelado de la arquitectura.

4.1 Breve introducción a las redes de Petri.

La estructura de una red de Petri es un grafo dirigido bipartido. Se distinguen dos tipos de nodos: los lugares y las transiciones, estos nodos no pueden estar relacionados con otros nodos del mismo tipo (bipartido). Generalmente los lugares son representados gráficamente por círculos, las transiciones por rectángulos y los arcos por flechas. Los arcos múltiples que relacionan de igual forma dos nodos son representados por un solo arco con un peso que representa el número de arcos múltiples. Para dar un comportamiento dinámico a la red de Petri se utilizan marcas dentro de los lugares de la red que son representadas por puntos o por un número entero dentro del lugar. Un marcado inicial es el conjunto de lugares marcados al inicio de la ejecución de una Red de Petri.

La evolución de una red de Petri es dada por las transiciones las cuales son las encargadas de mover las marcas desde los lugares de entrada hasta los lugares de salida. A esta acción de mover marcas "a través" de una transición se le llama disparo. La única condición para que se lleve a cabo el disparo de una transición es que los lugares de entrada tengan las mismas marcas o más que los arcos de salida que se relacionan con la transición, si se cumple esta condición se dice que la transición se encuentra habilitada. También existen arcos que se habilitan con la ausencia de marcas en un lugar estos arcos son llamados arcos complementados. Los arcos complementados pueden ser representados mediante arcos normales para su análisis.

En la figura 4.1(a) existe una transición T0 y cinco lugares P0, P1, P2, P3 y P4. Los lugares de entrada de la transición T0 son P0, P1 y P2, y los de salida P3 y P4. El arco de P1 a T0 representa la existencia de dos arcos múltiples. La transición T0 se encuentra habilitada y puede ser disparada. Al ser disparada se quita de los lugares de entrada un número de marcas igual al peso de los arcos de entrada a la transición y se pone en los lugares de salida un número de marcas igual al peso de los arcos de salida, la figura 4.1(b) muestra la red de Petri después del disparo de T0.

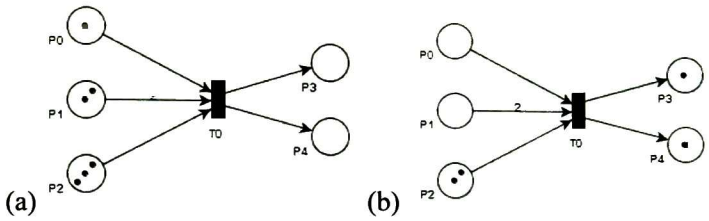


Figura 4.1 Ejemplo de una red de petri.

Cuando todos los arcos de una red de Petri tienen peso unitario, se trata de una *red de Petri ordinaria*, de otro modo es una *red de Petri generalizada*. Cuando a los lugares y/o a las transiciones de una red de Petri se les asocia un significado o evento real, como la señal de un sensor o de un botón, estamos hablando de una *red de Petri interpretada*.

Modelo matemático de las redes de Petri.

Formalmente una red de Petri generalizada es una cuádrupla G = (P, T, Pre, Post), donde,

- P es un conjunto de lugares finito ($|P| = n \geq 0$).
- T es un conjunto de transiciones finito ($|T| = m \geq 0$).
- Pre: P x T $\rightarrow$ N es una función de incidencia previa la cual representa el peso de los arcos de entrada a las transiciones.
- Post: P x T $\rightarrow$ N es una función de incidencia posterior la cual representa el peso de los arcos de salida de las transiciones.

Los conjuntos P y T cumplen con: $P \cap T = \emptyset$; $P \cup T = \emptyset$ (Grafo bipartido). Una función de marcado se define como μ: P $\rightarrow$ N.

Para representar las funciones de incidencia se pueden utilizar matrices de n x m, donde cada elemento Pre (pi,tj) o Post (pi,tj) denota el peso del arco que relaciona pi con tj. Y el marcado se pude denotar como un vector $[\mu(p1)\ldots\ \mu(pn)]^T$ donde μ(pi) es el número de marcas en el lugar pi.

Una transición tj $\in$ T está habilitada respecto a un marcado μ si y sólo si $\forall pi \in P,\ \mu(pi) \geq Pre(pi, tj)$; El disparo de una transición puede ser expresado como:

$$\mu'(pi) = \mu(pi) - Pre(pi, tj) + Post(pi, tj),\ \forall pi \in P \qquad (1)$$

La relación (1) puede escribirse también como:

$$\mu_k = \mu_{k-1} + Av_k \qquad (2)$$

Donde, A es la matriz de incidencia de la red se define como A = [aij]; aij = Post(pi,tj) Pre(pi,tj) y v_k es el vector de disparo que se define como v_k: T $\rightarrow$ {0,1}, el cual representa las transiciones que son disparadas en un instante k. La ecuación (2) es conocida como ecuación de estado de la red. Y por medio de esta ecuación se puede calcular el conjunto de marcados alcanzables desde un marcado inicial μ_0, llamado conjunto de alcanzabilidad R(μ_0).

Subclases de la redes de Petri.

Las redes de Petri ordinarias son divididas en subclases, cada subclase tiene restricciones estructurales con efecto de estudiar sus propiedades. Las subclases más estudiadas son las máquinas de estados, los grafos marcados y las redes libres de elección. En las máquinas de estados cada transición tiene sólo un lugar de entrada y uno de salida, estas permiten modelar situaciones de elección más no sincronización de procesos. En los grafos marcados cada lugar tiene sólo una transición de entrada y una de salida, estas permiten la sincronización y la distribución más no es posible expresar decisiones y situaciones de confluencia. En las redes de libre elección (free-choice) cada arco que sale de un lugar es el único que sale o el único que entra una transición y no ambas, en estas redes se trata de que las elecciones dependan únicamente del marcado de un lugar, permiten un inicio simultáneo, la sincronización y en una forma más restringida la elección entre alternativas.

Propiedades a probar de las redes de Petri.

Las propiedades que nos interesa estudiar en nuestro modelo de red de Petri son la vivacidad, el acotamiento y la libertad de bloqueos.

Vivacidad (*liveness*). Se dice que una transición es potencialmente disparable en un marcado μ, si existe un marcado $\mu' \in R(\mu)$ que la habilita. Una transición está viva si es potencialmente disparable para cada marcado $\mu \in R(\mu)$. Una red de Petri está viva con respecto a un marcado inicial μ_0 si y sólo si todas sus transiciones están vivas. La propiedad de vivacidad en el modelo significa que el sistema modelado puede alcanzar cualquier estado desde cualquier otro.

Libre de bloqueos (*deadlocks*). Si para un marcado dado es posible disparar ninguna transición se dice que se tiene un bloqueo. Una red de Petri en donde todos los marcados $\mu \in R(\mu)$ no provocan un bloqueo se dice que la red es libre de bloqueos. Si una red es viva entonces es libre de bloqueos. La propiedad de libertad de bloqueos en el modelo significa que el sistema modelado no se bloquea en ningún estado.

Acotamiento (*boundedness*). Una red en la cual no existen lugares que pueden acumular un número de marcas indefinido se dice que es acotada. Un lugar pi $\in$ P de una red de Petri con un marcado inicial μ_0 está k-acotado si y sólo si $\forall \mu \in R(\mu_0),\ \mu(pi) \leq k$. Si k es el número de marcas más grande en la red se dice que la red está k-acotada. Si una red de Petri es 1-acotada se dice que es un red a salvo (*safe*). La propiedad de acotamiento en el modelo significa que el sistema modelado tiene un número finito de estados.

Análisis por reducción.

El análisis por reducción se basa en la aplicación de reglas de reducción con el objetivo de simplificar la red de Petri conservando sus propiedades de vivacidad y acotamiento. Las reglas se aplican paso a paso hasta obtener una red de Petri de tamaño manejable para ser analizada por métodos como el grafo de cobertura o por medio de los invariantes. Se enuncian las reglas de reducción utilizadas en este trabajo.

Substitución de un lugar. Como muestra la figura 4.2 el lugar P1 puede ser eliminado sin afectar las propiedades de la red. Las nuevas transiciones emulan los disparos de las antiguas transiciones. Por ejemplo el disparo de la transición T01 equivale a disparar las transiciones T0 y T1. Ahora un lugar pi es sustituible si:

a) $\forall\, tj \in pi \qquad tj = \{pi\}\, y\, Pre(pi, tj) = 1$
b) $\cdot pi \cap pi \cdot = \emptyset$
c) $\exists tj \in pi \cdot, tj \neq \emptyset$

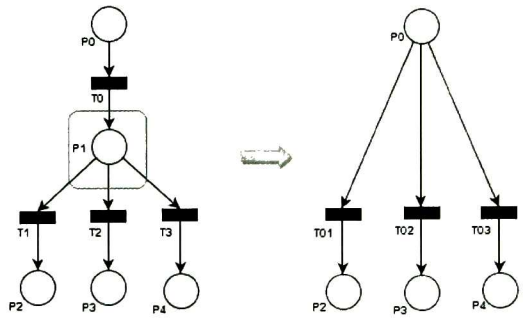


Figura 4.2 Substitución de un lugar.

Transiciones idénticas. Dos transiciones son idénticas si y sólo si $tj = \cdot tr\ y\ tj \cdot = tr \cdot$. Como se muestra en la figura 4.3 las transiciones T1 y T2 son idénticas y puede ser eliminada una de las dos.

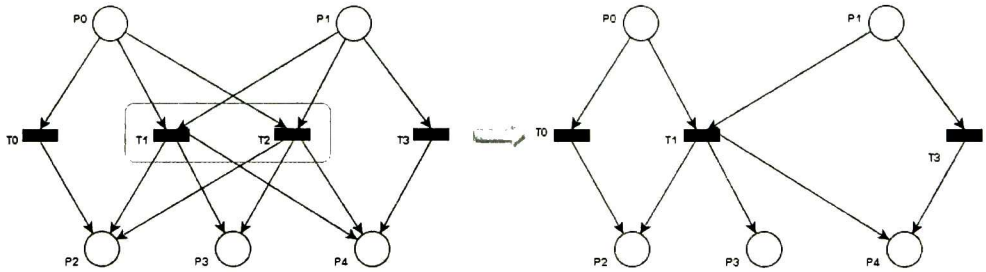


Figura 4.3 Transiciones idénticas.

Análisis por el grafo de cobertura.

Este método se basa en el análisis de todas las posibles evoluciones del marcado en la red de Petri. Se trata de generar un grafo con los marcados alcanzables a partir de un marcado inicial simulando todas las posibles evoluciones del marcado. El marcado inicial es el nodo de partida para el grafo y los demás marcados se van generando a partir del disparo de las transiciones habilitadas por el marcado. En [Mellado97] se detalla un algoritmo para la creación del grafo de cobertura. Con la creación del grafo de cobertura y su análisis se pueden deducir las propiedades dinámicas de la red de Petri.

Una red de Petri es *acotada* ssi el símbolo ω nunca aparece en los nodos del grafo de cobertura (el símbolo ω aparece en un nodo cuando los marcados crecen indefinidamente). Una red de Petri es *libre de bloqueos* ssi el grafo de cobertura no tiene nodos terminales. Los nodos terminales son aquellos que no tienen arcos de salida cuyo marcado no habilita ninguna transición. Una red de Petri es *repetitiva* ssi el grafo contiene un circuito dirigido que contiene todas las transiciones. Una red de Petri es *reversible* ssi cada nodo esta en al menos un circuito dirigido conteniendo al marcado inicial, el grafo es fuertemente conexo. Una red de Petri es *viva* ssi en el grafo de cobertura, todas las componentes fuertemente conexas contienen todas las transiciones.

4.2 Modelado del protocolo mediante redes de Petri.

Para modelar la conexión y desconexión del protocolo utilizado en la arquitectura propuesta se utilizaron redes de Petri interpretadas, se creó una para cada tipo de mensaje que llega a los nodos activos. En las redes existen lugares compartidos que son unidos después para analizar el protocolo completo.

El lugar *sec* asegura que sólo se atenderá un mensaje a la vez esto para evitar la inestabilidad del protocolo, en la realidad es necesario tener una cola para almacenar los mensajes hasta que puedan ser procesados. Los lugares *ses* y *nses* representan la existencia o no existencia de la sesión de video en el nodo respectivamente. Así mismo los lugares *class* y *nclass* representan la existencia o no existencia de clases de video en el nodo. Los lugares *sim* y *nsim* sirven para definir si la petición de difusión es una simulación o no. Los lugares *neigh* y *nneigh* representan a los vecinos del nodo activo. Y los lugares *wait* y *waitdif* son lugares de espera de la llegada de otro mensaje. Todas las transiciones en las redes de Petri representan un evento como la llegada de un mensaje, el envío de un mensaje, el resultado de una consulta de la tabla de estados del nodo ó una comparación de las variables del nodo.

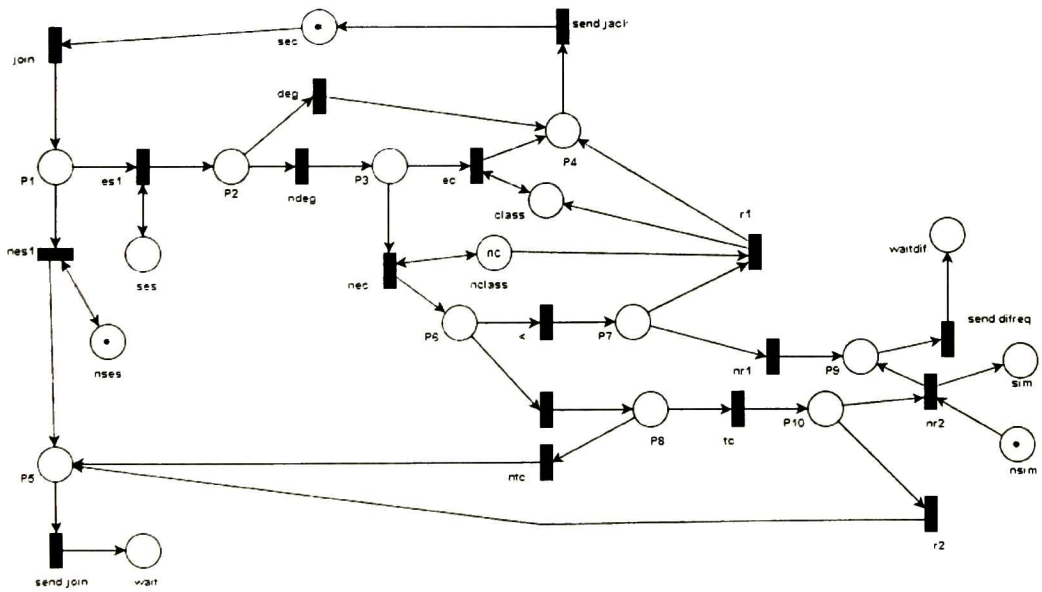


Figura 4.4 RP mensaje join

Transición	Evento
join	Llegada de un mensaje join
es1	Consulta (Existe la sesión).
nes1	Consulta (No existe la sesión).
deg	Consulta (El nodo se encuentra degradado).
ndeg	Consulta (El nodo no se encuentra degradado).
ec	Consulta (Existe la clase en el nodo).
nec	Consulta (No existe la clase en el nodo).
<	Consulta (La clase pedida es menor a up_class).
>	Consulta (La clase pedida es mayor a up_class).
r1	Consulta (Existen recursos).
nr1	Consulta (No existen recursos).
r2	Consulta (Existen recursos).
nr2	Consulta (No existen recursos).
tc	Consulta (Es necesario agregar un transcodificador).
ntc	Consulta (No es necesario agregar un transcodificador).
send difreq	Envía mensaje ctrl-difreq.
send join	Envía mensaje join.
send jack	Envía mensaje join_ack.

Tabla 4.1 Tabla de eventos, join.

La figura 4.4 muestra los estados del protocolo a la llegada de un mensaje join, el modelo se basa en los estados descritos anteriormente en la tabla 3.4. Y la figura 4.5 muestra la llegada de un mensaje join_ack. La figura 4.6 muestra los estados del protocolo a la llegada de los mensajes *dify* y *difn* y la elección del candidato para la difusión.

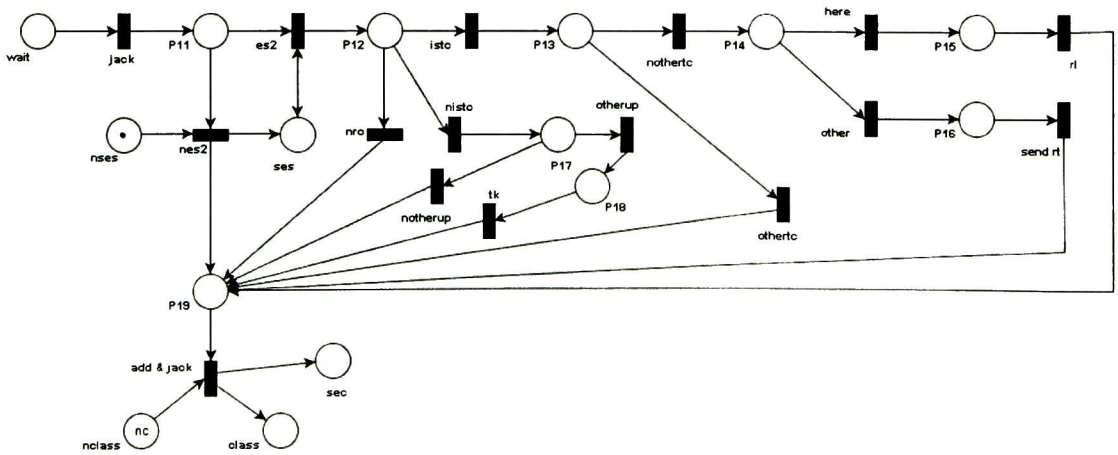


Figura 4.5 RP mensaje join_ack

Transición	Evento
jack	Llegada de mensaje join_ack.
es2	Consulta (Existe sesión).
nes2	Consulta (No existe sesión).
nrc	Consulta (No es rclass).
istc	Consulta (Utilizaba transcodificador).
nistc	Consulta (No utilizaba transcodificador).
othertc	Consulta (Otro utiliza el transcodificador).
nothertc	Consulta (Ninguno utiliza el transcodificador).
here	Consulta (Transcodificador local).
other	Consulta (Transcodificador migrado).
otherup	Consulta (Otro utiliza up_class).
notherup	Consulta (Ninguno utiliza up_class).
rl	Libera recursos.
tk	Toma recursos.
send rt	Envía ctrl-tc_res
add & jack	Agrega a la tabla y envía mensaje join_ack.

Tabla 4.2 Tabla de eventos, join_ack.

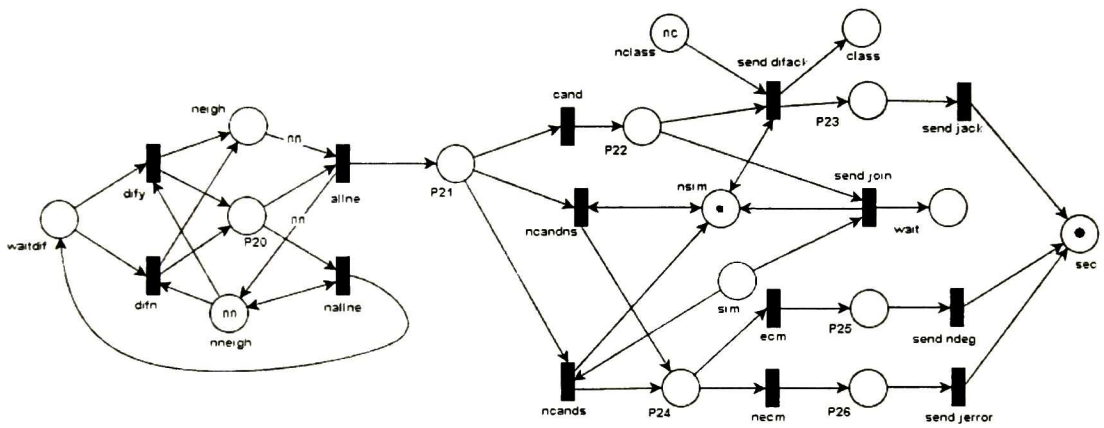


Figura 4.6 RP mensajes dify y difn

Transición	Evento
dify	Llegada de mensaje ctrl-dify.
difn	Llegada de mensaje ctrl-difn.
allne	Consulta (Todos los vecinos contestaron).
nallne	Consulta (Faltan vecinos por contestar).
cand	Consulta (Existen candidatos).
ncandns	Consulta (No existen candidatos no es simulación).
ncands	Consulta (No existen candidatos es una simulación).
ecm	Consulta (Existe una clase menor).
necm	Consulta (No existe una clase menor).
send difack	Envío de mensaje ctrl-difack.
send jack	Envío de mensaje join_ack.
send join	Envío de mensaje join.
send ndeg	Envío de mensaje ctrl-node_deg.
send jerror	Envío de mensaje join_error.

Tabla 4.3 Tabla de eventos, dify y difn.

La figura 4.7 muestra los estados a la llegada de un mensaje leave, se basa en los estados descritos anteriormente en la tabla 3.7.

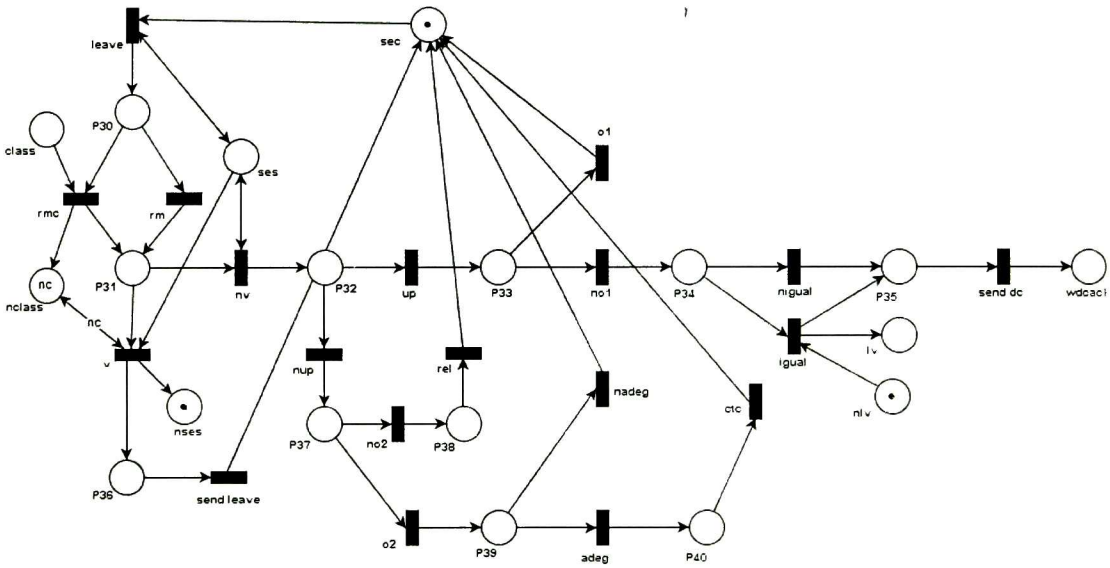


Figura 4.7 RP mensaje leave.

Transición	Evento
leave	Llegada de mensaje leave
rmc	Consulta (Elimina una clase del nodo).
rm	Consulta (No elimina una clase del nodo).
v	Consulta (La tabla queda vacía).
nv	Consulta (La tabla no queda vacía).
up	Consulta (Recibe up_class).
nup	Consulta (No recibe up_class).
o1	Consulta (Otro nodo recibe la misma clase).
no1	Consulta (Ningún nodo recibe la misma clase).
o2	Consulta (Otro nodo recibe la misma clase).
no2	Consulta (Ningún nodo recibe la misma clase).
adeg	Consulta (Todos los nodos están degradados).
nadeg	Consulta (No todos los nodos están degradados).
igual	Consulta (La clase mayor es igual a la degradada).
nigual	Consulta (La clase mayor no es igual a la degradada).
rel	Libera recursos.
send dc	Envío de mensaje ctrl-down_class.
send leave	Envío de mensaje leave.

Tabla 4.4 Tabla de eventos, leave.

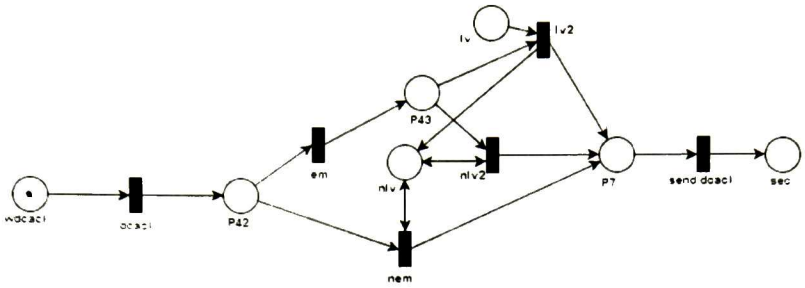


Figura 4.8 RP mensaje down_classack

Transición	Evento
dcack	Llegada de mensaje ctrl-down_class
em	Consulta (La clase es menor a up_class).
nem	Consulta (La clase no es menor a up_class).
lv2	Consulta (Libera recursos).
nlv2	Consulta (No libera recursos).
send dcack	Envío de mensaje ctrl-down_classack

Tabla 4.5 Tabla de eventos, down_classack.

4.3 Análisis de las propiedades del protocolo mediante el modelo de redes de Petri.

Para el análisis del protocolo se aplicaron las reglas de reducción para disminuir el tamaño de las redes de Petri y poder juntaras para crear una red de Petri global. Las redes de Petri resultantes se analizaron utilizando la herramienta de análisis para redes de Petri PIPE (Platform Independent Petri Net Editor) [wpipe] obteniendo las invariantes y el grafo de cobertura.

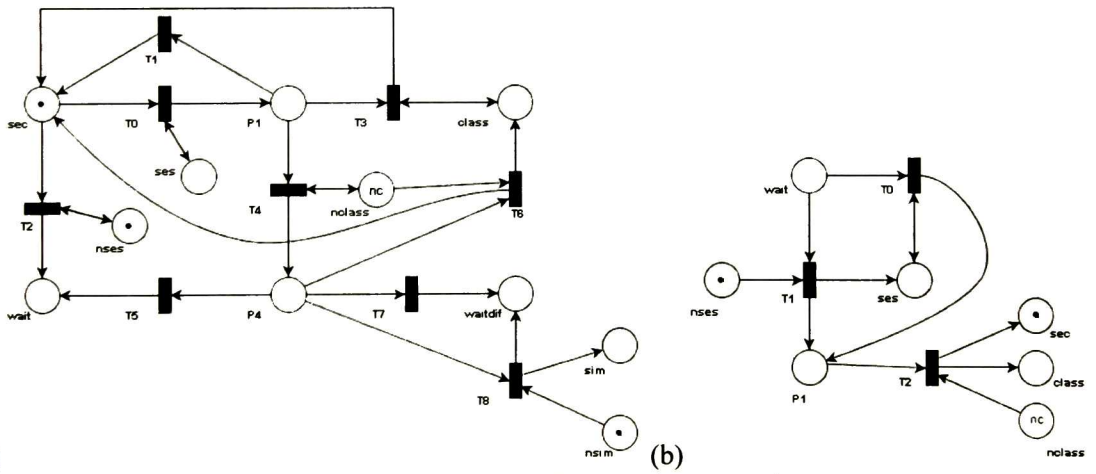


Figura 4.9 RP reducida (a) join (b) join_ack.

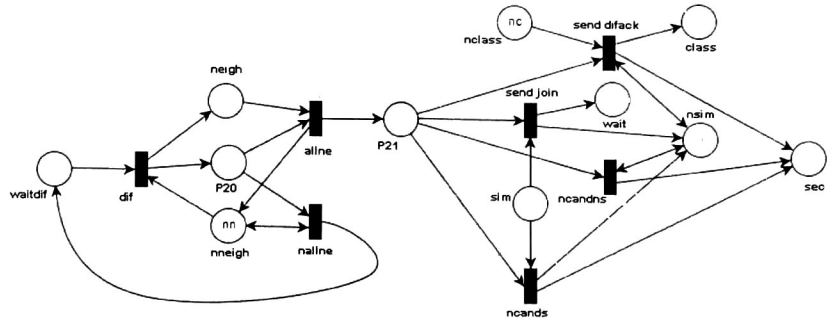


Figura 4.10 RP reducida mensajes dify difn.

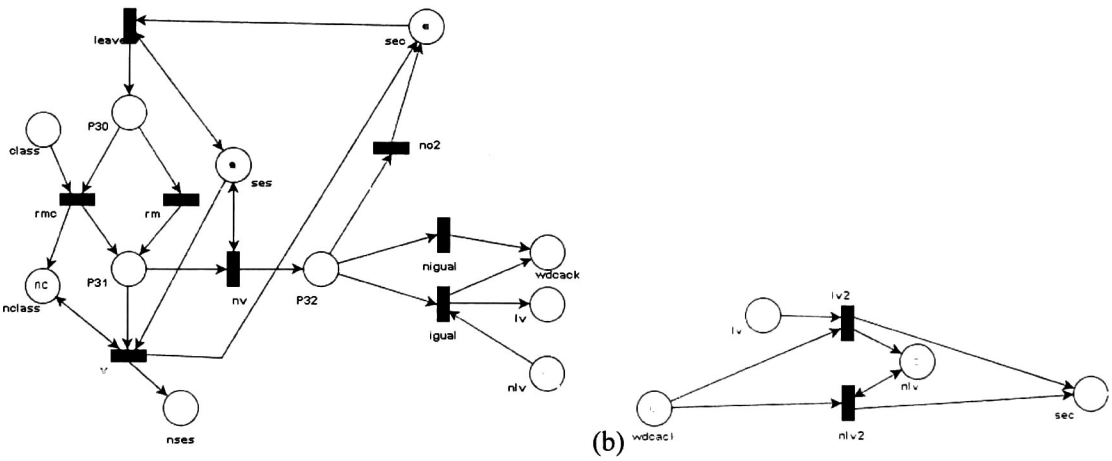


Figura 4.11 RP reducida (a) leave (b) down_classack

De acuerdo a los resultados obtenidos mediante el simulador PIPE que se muestran en la figura 4.13 la red de Petri de la figura 4.12 es libre de bloqueos y acotada, no es segura porque no es 1-acotada sino que su acotamiento depende del número de clases, sesiones y vecinos que se manejen en el modelo.

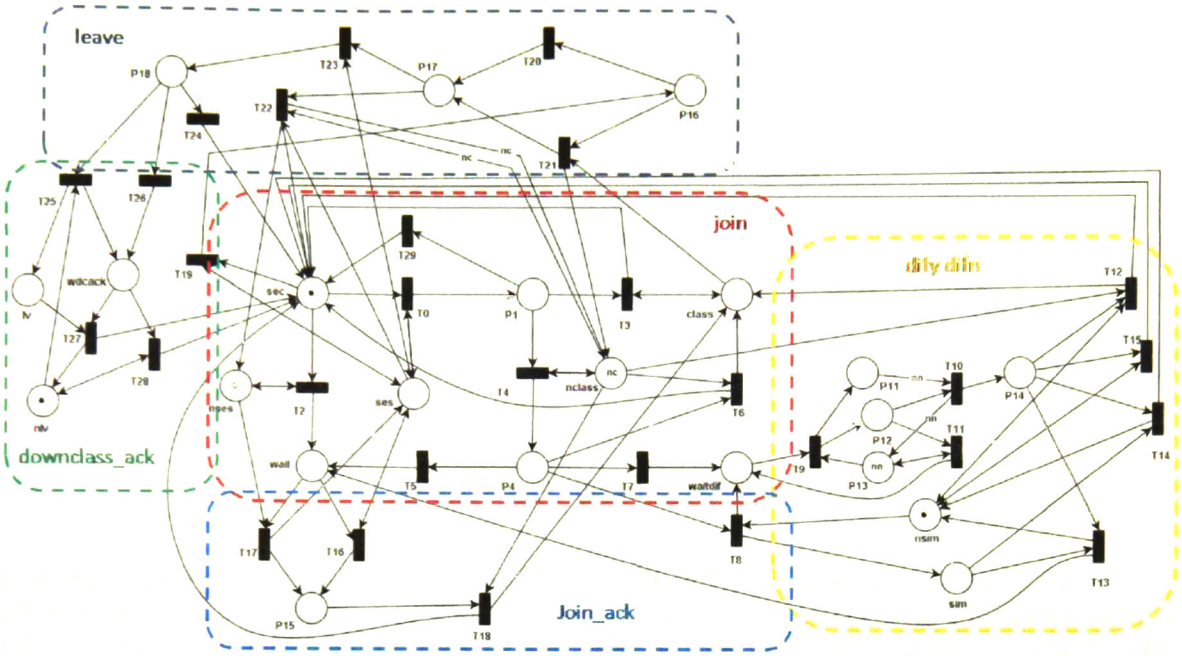


Figura 4.12 Modelo completo del protocolo.

Petri net state space analysis results

Bounded	true
Safe	false
Deadlock	false

Figura 4.13 Resultado del análisis en PIPE2.

Se sabe que una red de Petri es *viva* ssi en el grafo de cobertura, todas las componentes fuertemente conexas contienen todas las transiciones. En este caso el grafo de cobertura sólo tiene una componente, esto simplifica el problema a probar que esta componente es fuertemente conexa. Un digrafo es fuertemente conexo si para cualesquiera dos vértices v y w, v es accesible desde w y viceversa. Un vértice v es accesible desde un vértice w si existe un camino dirigido en el digrafo que inicia en w y termina v.

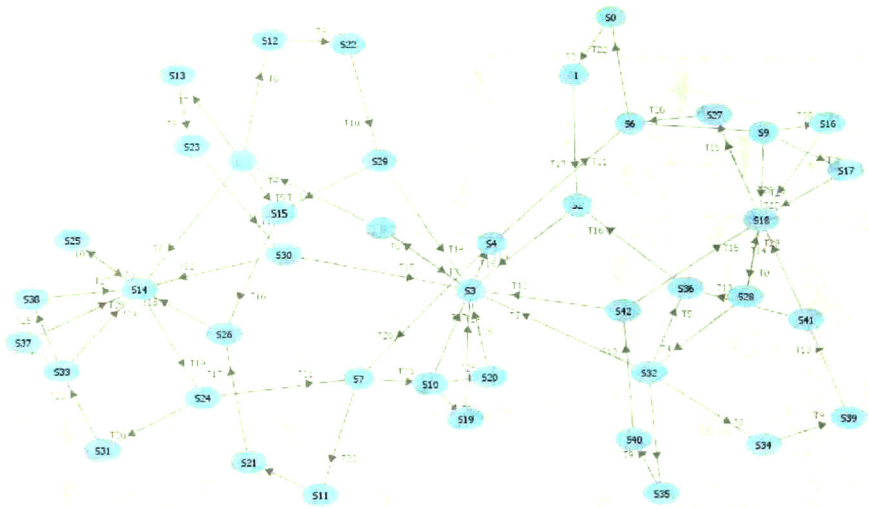


Figura 4.14 Grafo de cobertura.

Para determinar la vivacidad del protocolo modelado en la figura 4.12 se genero el grafo de cobertura en PIPE2 mostrado en la figura 4.14. Debido a que la herramienta PIPE2 no genera grafos muy grandes se restringió para dos clases de video (*nc*) y un vecino (*nn*), suponiendo que los grafos para más clases contendrán las mismas propiedades. Después, se generó la matriz de adyacencia la cual sirvió como entrada al algoritmo de Warshall para calcular la matriz de cerraduras transitivas, el algoritmo se corrió en MATLAB 7.1 y se muestra abajo, donde en un principio A es la matriz de adyacencia del grafo de cobertura de la figura 4.14 multiplicada por la matriz identidad para llenar la diagonal de unos y al finalizar el algoritmo A contiene la matriz de cerraduras transitivas.

```
A = [
1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
0,0,0,0,1,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
0,0,0,1,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
1,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
0,0,0,0,0,0,0,1,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,1,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0;
0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1;
0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
```

```
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0;
0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0;
0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0;
0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,1,1,0,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,1,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0;
0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0;
0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1;
];

n   43;
for k = 1:n
  for i = 1:n
    for j = 1:n
        if A(i,j) == 0
           A (i,j)    A(i,k)    A(k,j);
        end;
    end;
  end;
end;
```

El resultado del algoritmo de Warshall fue una matriz de puros unos, esto significa que cualquier vértice es accesible desde cualquier otro, de acuerdo con esto se puede afirmar que el grafo de cobertura es fuertemente conexo. Por lo tanto la red de Petri de la figura 4.12 es viva.

4.4 Conclusiones

De acuerdo con los resultados del simulador PIPE2 la red de Petri utilizada para modelar el comportamiento de un nodo es acotada y libre de bloqueos, esto significa que la arquitectura tiene un número finito de estados y que no se bloquea en ninguno de los estados. Además, se demostró por medio del grafo de cobertura que la red de Petri es viva, esto significa que el protocolo puede alcanzar cualquier estado desde cualquier otro.

CAPITULO 5

5 Trabajo experimental.

Resumen: En este capítulo se muestran varias simulaciones de la arquitectura propuesta realizadas en el simulador NS2 (Network Simulator 2) [wns2]. Para validar que la programación de las simulaciones fuera correcta se realizaron varias pruebas de funcionamiento que complementan el análisis de la formalización mostrada en el capítulo 4. También, hicieron simulaciones para estudiar el desempeño de la arquitectura propuesta en varios escenarios mostrados en la sección 5.2.3.

5.1 Simulación.

NS2 es probablemente uno de los simuladores de código abierto (*open source*) más utilizados en la investigación de nuevas tecnologías de red. Fue desarrollado por la universidad de California en Berkeley. NS2 combina los lenguajes C++ y OTcl, el primero utilizado para programar los protocolos de red y el funcionamiento de la red, mientras que el segundo es utilizado para definir la topología de la red, los protocolos y los eventos que se quieren simular. El simulador se basa en eventos discretos que son almacenados en una cola para su ejecución.

Existen herramientas de generación automática de escenarios como *cbrgen* y *setdest*. El simulador escrito en su mayoría en C++ ejecuta los escenarios y genera trazas de resultados. NS2 proporciona una herramienta de visualización llamada NAM, esta lee las trazas de salida del simulador y visualiza el comportamiento de la red en un entorno gráfico. Existen otras trazas que pueden ser leídas en PERL o graficadas mediante la herramienta *xgraph* que no son utilizadas en este trabajo. La figura 5.1 muestra los módulos del simulador NS2.

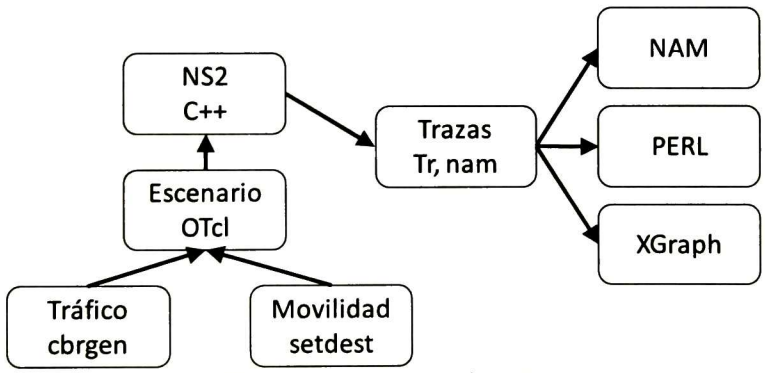


Figura 5.1 Módulos de NS2.

La simulación se realizó y ejecutó sobre el sistema operativo Linux SUSE 10.1 utilizando la plataforma NS2 versión 2.31. El sistema operativo se monto en una máquina virtual mediante el programa VMware 5.5.1 instalada en una computadora móvil con procesador Core 2 Duo a 1.73Ghz y con 1Gb de memoria RAM.

Estructura de las clases creadas en C para la simulación.

La figura 5.2 muestra el diagrama de clases de NS2 con la incorporación de la clase Anode definida para nuestro protocolo. La figura 5.3 muestra el diagrama de clases de la simulación. En la clase Anode se define el comportamiento de la arquitectura, esta clase hereda de la clase Agent ya definida en el simulador, las clases que heredan de la clase Agent pueden ser instanciadas dentro de los nodos de la red. En la simulación cada nodo de la red contiene una instancia de la clase Anode y cada objeto Anode crea una instancia de las clases Anode_rtable y Anode_timer. La clase Anode_rtable define la tabla de estados de los nodos y las operaciones que se pueden realizar sobre la tabla mientras que la clase Anode_timer genera un evento cada vez que expira un contador para medir periódicamente la tasa de paquetes perdidos.

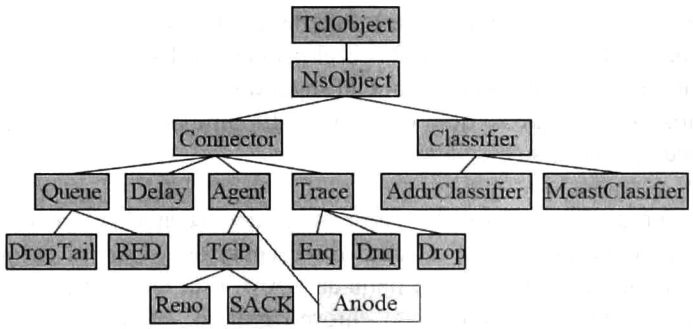


Figura 5.2 Jerarquía de clases de NS2

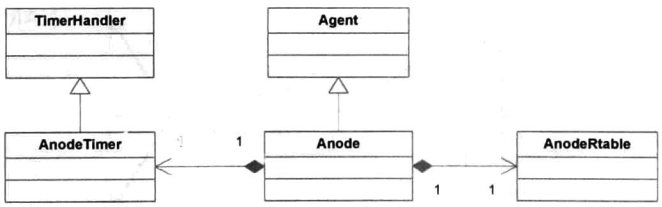


Figura 5.3 Diagrama de Clases.

Creación y configuración de los escenarios en TCL

La tabla 5.1 muestra las variables y operaciones en lenguaje TCL que se utilizan para definir los parámetros de la simulación y los eventos de los nodos. Donde las variables 'p(x)' hacen referencia al agente del nodo 'x'.

Nombre	Tipo	Uso en tcl	Descripción
Agent/Anode	Operación	`set p(0) [new Agent/Anode]`	Crea un Nuevo agente Anode y guarda la referencia en p(0).
type_node	Variable	`$p(0) type_node 0`	Especifica el tipo de nodo. 0 servidor, 1 Nodo activo y 2 usuario.
resources	Variable	`$p(0) set resources 4`	Especifica el número de transcodificadores que puede albergar el nodo.
Toplimit	Variable	`$p(1) set toplimit 5.0`	Especifica el límite para la tasa de paquetes perdidos.
report_interval	Variable	`$p(1) set report_interval 1.0`	Especifica el intervalo de medición en ms
link_adaptation	Variable	`$p(1) set link_adaptation 0`	Activa (1) o desactiva (0) la adaptación a los recursos de los enlaces.
node_adaptation	Variable	`$p(1) set node_adaptation 0`	Activa (1) o desactiva (0) la adaptación a los recursos de los nodos.
Join	Operación	`$ns at 0.3 "$p(2) join 1 1"`	Envía un mensaje de conexión al servidor de video. Los parámetros son el número de sesión y la clase de video respectivamente.
Leave	Operación	`$ns at 0.5 "$p(2) leave 1 1"`	Envía un mensaje de desconexión al servidor de video. Los parámetros son el número de sesión y la clase de video respectivamente.

Tabla 5.1 API del protocolo en Tcl.

Simulación del envío de video y transcodificación de video.

Para la simulación de tráfico de video real se utilizaron algunas trazas de video real y un script en Tcl que transforma la traza de video en una traza que puede ser leída por el generador de tráfico de NS2. Las trazas y el script fueron realizadas por la Universidad del estado de Arizona (ASU) y se encuentran disponibles en su web [wasu]. Aquí se muestra el script para transformar las trazas.

```
#Script para la simulación de la transmisión de video real.
## Generate the video trace file:
set original_file_name Verbose_Simpsons_256.dat
set trace_file_name video.dat
set original_file_id [open $original_file_name r]
set trace_file_id [open $trace_file_name w]
set last_time 0
fconfigure $trace_file_id -encoding binary
fconfigure $trace_file_id -translation binary
while {[eof $original_file_id] == 0} {
  gets $original_file_id current_line
  if {[string length $current_line] == 0 ||
  [string compare [string index $current_line 0] "#"] == 0} {
  continue
  }
  scan $current_line "%d%s%d" next_time type length
  set time [expr 1000*($next_time-$last_time)]
  set last_time $next_time
  puts -nonewline $trace_file_id [binary format "II" $time $length]
}
close $original_file_id
close $trace_file_id

## read the video trace file:
set trace_file [new Tracefile]
$trace_file filename $trace_file_name
set video [new Application/Traffic/Trace]
$video attach-agent $udp
$video attach-tracefile $trace_file
```

Para poder iniciar una transmisión es necesario agregar al nodo servidor (nodo cero) un agente UDP y un agente Sink, como se muestra abajo. El nodo servidor automáticamente encapsula los paquetes UDP dentro de paquetes *data*, estos paquetes son utilizados por la arquitectura para transportar el video, y los envía a los nodos conectados a la sesión de video.

```
#Se agrega un agente UDP al nodo 0
set udp [new Agent/UDP]
$udp set fid_ 1
$ns attach-agent $n(0) $udp
$ns add-agent-trace $udp udp
set snk [new Agent/Null]
$snk set fid_ 1
$ns attach-agent $n(0) $snk
$ns connect $udp $snk
```

Debido a que sólo se quiere estudiar el tráfico en la red la transcodificación se simplifico a cambiar el tamaño de los paquetes para simular el efecto de *transrating* mediante la eliminación de información del video. Para esto se utilizó la fórmula (5.1). Donde el tamaño del paquete nuevo (*tin*) depende de la clase de video que se le debe enviar al nodo (*rclass*), el tamaño del paquete de entrada (*tout*) y de la clase de video del paquete de entrada (*vclass*). Las clases de video en la formula (*rclass* y *vclass*) pueden ser expresadas en base al número de clase o a la tasa de bits de la clase.

$$tin = rclass * tout / vclass. \qquad (5.1)$$

Cambios en el código fuente en C++ de NS2.

Para agregar nuestro agente en el simulador NS2 se deben realizar varios cambios en el código fuente, estos cambios se detallan en el artículo [Ros04]. Además se realizaron otros cambios en el código fuente del simulador para adaptarlo a las necesidades de la arquitectura, estos cambios se detallan en esta sección.

Para que los nodos sean capaces de leer todos los paquetes del protocolo que pasen por ellos aunque no estén dirigidos a ellos (hacerlos nodos activos), se realizaron algunos cambios en el código fuente de NS2. La figura 5.4 muestra la estructura de un nodo y la forma en que se envían los paquetes. Para que un nodo pasara todos los paquetes activos a nuestro agente *Anode* se modificaron los clasificadores *Addr-classifier* y *Port-Classifier* que se encuentran en los archivos classifier.cc y classifier-port.cc

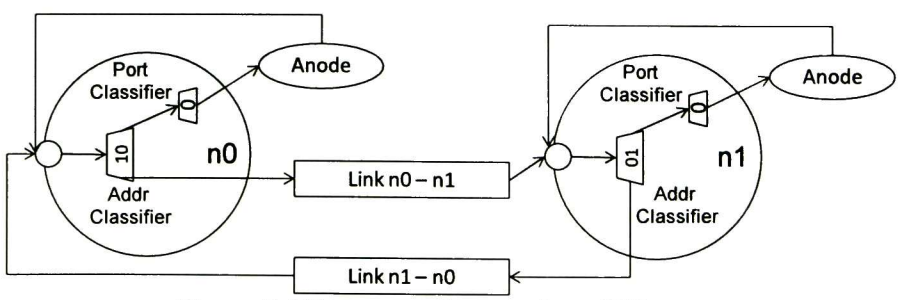


Figura 5.4 Estructura de un nodo en NS2.

Para identificar los mensajes de entrada y los de salida se utilizó el campo *error* de la cabecera IP, si el campo es 0 el mensaje es de entrada y si es 1 es de salida, así que todos los mensajes que envía el agente Anode deben estar marcados con 1 para que sean enviados. Antes de enviar un paquete se modifica el valor del campo *error* poniéndolo en 0 para que el nodo destino lo lea como paquete de entrada. Los paquetes que interesan al agente son los de tipo PT_ANODE y PT_UDP, los primeros son los mensajes de control del protocolo y los segundos son usados para transportar el video. El siguiente fragmento de código muestra los cambios hechos en la función *recv* del archivo classifier.cc.

```
void Classifier::recv(Packet* p, Handler*h){
        NsObject* node   find(p);
        hdr_cmn* cmnh = hdr_cmn::access(p);
        if((cmnh->ptype() == PT_ANODE || cmnh->ptype() == PT_UDP) && cmnh->error()==0){
                node = slot_[0];
        } else if (node == NULL) {
                Packet::free(p);
                return;
        }
        if (cmnh->error() == 1)
                cmnh->error() = 0;
        node->recv(p,h);
}
```

Para que los paquetes sean entregados a nuestro agente Anode, se cambia el puerto destino de todos los paquetes de tipo PT_UDP y PT_ANODE que son los que nos interesan. El siguiente fragmento de código muestra los cambios hechos en la función *classify* del archivo classifier-port.cc para cambiar el puerto de los paquetes.

```
int PortClassifier::classify(Packet *p) {
        // Port classifier returns the destination port.  No shifting
        // or masking is required since in the 32-bit addressing,
        // ports are stored in   seperate variable.
        hdr_ip* iph   hdr_ip::access(p);
        hdr_cmn* cmnh   hdr_cmn::access(p);
        if(cmnh->ptype() == PT_ANODE || cmnh->ptype() == PT_UDP){
                return(0);
        } else
                return iph->dport();
}
```

Con el objetivo de simular un protocolo que asegure la entrega de los paquetes de control como TCP, se debe asegurar que los paquetes de control no se pierdan, esto se logró modificando la función *enque()* de la clase DropTail, como se muestra en el código de abajo. Se utiliza el campo *prio* de la cabecera IP definiendo tres niveles de prioridad cero uno y dos. Los paquetes con prioridad cero (paquetes tipo *data*) pueden ser tirados de la cola pues utilizan el protocolo UDP, los paquetes con prioridad 1 (paquetes de degradación de clases) no son tirados y se colocan al final de la cola y los paquetes con prioridad 2 (demás paquetes de control) no son tirados y son puestos al principio de la cola dándoles mayor prioridad sobre los demás.

```
//Función enque() modificada
void DropTail::enque(Packet* p){
  if (summarystats) {
    Queue::updateStats(qib_?q_->byteLength():q_->length());
  }
  int qlimBytes = qlim_   mean_pktsize_;
  if ((!qib_ && (q_->length() + 1) >= qlim_) ||
   (qib_ && (q_->byteLength() + hdr_cmn::access(p)->size()) >= qlimBytes)){
        // if the queue would overflow if we added this packet...
        if (drop_front_) { /* remove from head of queue */
                q_->enque(p);
```

```
                Packet *pp = q_->deque();
                drop(pp);
        } else {
                if(hdr_ip::access(p)->prio() == 2 && hdr_cmn::access(p)->ptype() == PT_ANODE){
                        q_->enque(p);
                } else if(hdr_ip::access(p)->prio() == 1 && hdr_cmn::access(p)->ptype() == PT_ANODE){
                        q_->enqueHead(p);
                } else
                        drop(p);
        }
  } else {
        if(hdr_ip::access(p)->prio() == 1 && hdr_cmn::access(p)->ptype() == PT_ANODE){
                q_->enqueHead(p);
        } else
                q_->enque(p);
  }
}
```

5.2 Simulaciones.

Las simulaciones son divididas en dos, pruebas de validación del funcionamiento de la arquitectura y pruebas para el análisis del desempeño de la arquitectura. Las pruebas de validación tienen como objetivo probar el correcto funcionamiento de la simulación para que se comporte de acuerdo a la arquitectura propuesta en el capítulo 3, mientras que las pruebas de desempeño sirven para estudiar el desempeño que tiene la arquitectura propuesta. Además se describe un escenario ideal donde se asegura la satisfacción de todos los usuarios mediante técnicas de QoS.

5.2.1 Definición de un escenario ideal y aseguramiento mediante técnicas de QoS.

Un escenario ideal o perfecto es aquel en el que todos los nodos activos y todos los enlaces cuentan con los recursos suficientes para satisfacer las necesidades de todos los usuarios. Un nodo con recursos suficientes es aquel que tiene la capacidad de albergar n-1 transcodificadores, donde n es el número de clases de video que soporta la aplicación, esto es razonable ya que un nodo activo que recibe una clase *up_class,* en el peor de los casos deberá adaptar la clase *up_class* a todas las clases inferiores, esto es a *up_class*-1, *up_class*-2,..., 1. Así el número de clases inferiores será *up_class*-1 si *up_class* es la clase mayor entonces se tiene el peor de los casos teniendo n-1 clases inferiores donde n es la clase mayor o lo que es lo mismo el número de clases. Esto es mostrado en las pruebas de desempeño referentes a la probabilidad de conexión.

En un escenario donde todos los nodos tienen los suficientes recursos, un enlace con recursos suficientes es aquel que soporta la tasa de bits de la clase de video más alta. Esto se debe a que el algoritmo de conexión utilizado en la arquitectura cuando no existe difusión de la computación crea un árbol dirigido, asegurando que los enlaces sólo transportarán una clase de video a la vez.

En el caso donde los nodos activos y los enlaces de la red cuenten con recursos suficientes, se pueden utilizar técnicas de QoS para reservar recursos tanto en los nodos como en los enlaces, y así garantizar que todos los usuarios serán siempre satisfechos.

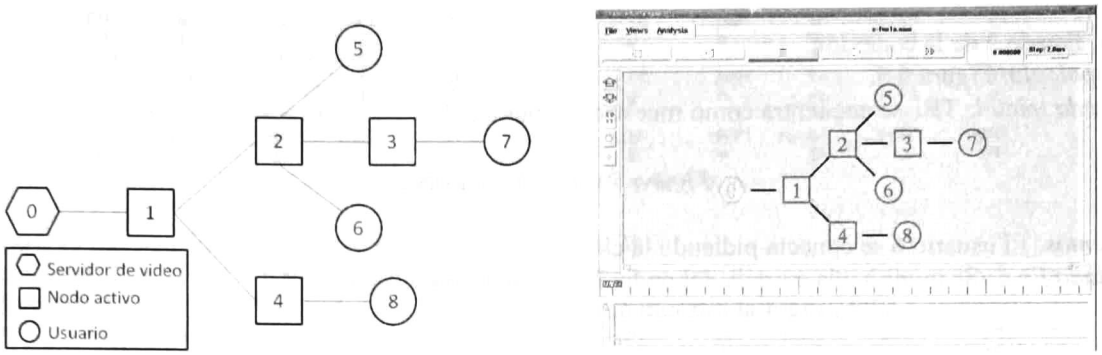


Figura 5.5 Topología para pruebas de validación 1.

5.2.2 Pruebas de validación.

Cada prueba de validación corresponde a uno o varios de los estados especificados en el capítulo 3. Las topologías utilizadas para las pruebas de validación tienen los elementos mínimos para propiciar el estado que se quiere probar. Cada prueba tiene un estado inicial y un estado final de las tablas de estados de los nodos implicados. Para cambiar el estado inicial se genera un evento que debe tener como resultado el estado final. El estado final es comparado con la descripción presentada en el capítulo 3.

Nomenclatura: Para simplificar el término "Tabla de estados en el nodo x" se utiliza TEx.

Nomenclatura: En las pruebas se muestran figuras que contienen las tablas de estados de los nodos afectados, cada renglón de la tabla comienza con un identificador del nodo llamado "PX" donde "X" es la dirección del nodo de la tabla mostrada, el significado de los campos de la tabla es mostrado en la tabla 3.1.

Pruebas para los estados de la conexión.

a) Estado 1 de la conexión.
Topología: Figura 5.5.
Estado inicial: Las TE de todos los nodos están vacías.
Eventos: El usuario 6 se conecta pidiendo la clase 5.
Estado final: Los nodos 0, 1 y 2 agregan a sus hijos en el árbol multicast como muestra la figura 5.6.

```
P0      ses     dest    rc      dlc     dnc     tc      psh     tn      dn      dl
P0      1       1       5       5       5       y       n       -1      -1      n
1 Receive join_ack packet
P1      ses     dest    rc      dlc     dnc     tc      psh     tn      dn      dl
P1      1       2       5       5       5       y       n       -1      -1      n
2 Receive join_ack packet
P2      ses     dest    rc      dlc     dnc     tc      psh     tn      dn      dl
P2      1       6       5       5       5       y       n       -1      -1      n
```

Figura 5.6 Tablas de estados.

Observaciones: De acuerdo al estado 1 de la conexión (sección 3.2.4) el estado final es correcto.

b) Estado 2 de la conexión.
Topología: Figura 5.5.
Estado inicial: TE1 se encuentra como muestra la Figura 5.7.

P1	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P1	1	2	4	3	4	y	n	1	-1	y

Figura 5.7 Tablas de estados.

Eventos: El usuario 6 se conecta pidiendo la clase 5.
Estado final: Se modifica la entrada del nodo 2 como muestra la figura 5.8.

P1	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P1	1	2	5	3	5	y	n	-1	-1	y

Figura 5.8 Tablas de estados.

Observaciones: De acuerdo al estado 2 de la conexión (sección 3.2.4) el estado final es correcto.

c) Estado 3 de la conexión.
Topología: figura 5.5.
Estado inicial: La TE2 se encuentra como muestra la figura 5.6.
Eventos: El usuario 7 se conecta pidiendo la clase 5.
Estado final: Se agrega a la TE2 la entrada del nodo 3 como muestra la figura 5.9.

P2	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P2	1	6	5	5	5	y	n	-1	-1	n
P2	1	3	5	5	5	y	n	-1	-1	n

Figura 5.9 Tablas de estados.

Observaciones: De acuerdo al estado 3 de la conexión (sección 3.2.4) el estado final es correcto.

d) Estado 4 de la conexión.
Topología: Figura 5.5.
Estado inicial: La TE2 se encuentra como muestra la figura 5.9.
Eventos: El usuario 5 se conecta pidiendo la clase 3.
Estado final: Se agrega a la TE2 la entrada del nodo 5 como muestra la figura 5.10.

P2	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P2	1	6	5	5	5	y	n	-1	-1	n
P2	1	3	5	5	5	y	n	-1	-1	n
P2	1	5	3	3	3	y	n	2	-1	n

Figura 5.10 Tablas de estados.

Observaciones: De acuerdo al estado 4 de la conexión (sección 3.2.4) el estado final es correcto.

e) Estado 5 de la conexión.
Este estado se encuentra implícito en las pruebas de difusión de la computación mostradas más adelante en esta misma sección.

f) Estado 6 de la conexión, caso 2.
Topología: Figura 5.5.
Estado inicial: Las TE0, TE1 y TE2 se encuentran como muestra la Figura 5.11.

P0	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P0	1	1	1	1	1	y	n	0	-1	n
1 Receive join_ack packet										
P1	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P1	1	2	1	1	1	y	n	-1	-1	n
2 Receive join_ack packet										
P2	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P2	1	5	1	1	1	y	n	-1	-1	n

Figura 5.11 Tablas de estados.

Eventos: El usuario 7 se conecta pidiendo la clase 3. Join(3).
Estado final: Las TE0 y TE1 se actualizan con la nueva clase enviada, se agrega el nodo 3 a la TE2 y se actualiza el campo *tn=2* para el nodo 5 en la TE2. Como muestra la Figura 5.12.

P0	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P0	1	1	3	3	3	y	n	0	-1	n
1 Receive join_ack packet										
P1	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P1	1	2	3	3	3	y	n	-1	-1	n
2 Receive join_ack packet										
P2	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P2	1	3	3	3	3	y	n	-1	-1	n
P2	1	5	1	1	1	y	n	2	-1	n
3 Receive join_ack packet										
P3	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P3	1	7	3	3	3	y	n	-1	-1	n

Figura 5.12 Tablas de estados.

Observaciones: De acuerdo al estado 6 caso 2 de la conexión (sección 3.2.4) el estado final es correcto.

g) Estado 6 de la conexión, caso 4.
Topología: Figura 5.5.
Estado inicial: Las TE0, TE1 y TE2 se encuentran como muestra la Figura 5.13.

P0	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P0	1	1	4	4	4	y	n	0	-1	n
1 Receive join_ack packet										
P1	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P1	1	4	4	4	4	y	n	-1	-1	n
P1	1	2	3	3	3	y	n	1	-1	n
P2	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P2	1	3	3	3	3	y	n	-1	-1	n
P2	1	5	1	1	1	y	n	2	-1	n

Figura 5.13 Tablas de estados.

Eventos: El usuario 6 se conecta pidiendo la clase 3.
Estado final: Las TE0 y TE1 se actualizan con la nueva clase 5, se agrega a la TE2 la entrada del nodo 3 y se actualiza el campo *tn=2* para el nodo 3 en la TE2 como muestra la Figura 5.14.

P0	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P0	1	1	5	5	5	y	n	-1	-1	n
1 Receive join_ack packet										
P1	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P1	1	2	5	5	5	y	n	-1	-1	n
P1	1	4	4	4	4	y	n	1	-1	n
2 Receive join_ack packet										
P2	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P2	1	5	1	1	1	y	n	2	-1	n
P2	1	6	5	5	5	y	n	-1	-1	n
P2	1	3	3	3	3	y	n	2	-1	n

Figura 5.14 Tablas de estados.

Observaciones: De acuerdo al estado 6 caso 4 de la conexión (sección 3.2.4) el estado final es correcto.

h) Estado 7 de la conexión, caso 1.
Topología: Figura 5.5.
Estado inicial: Las TE0, TE1 y TE2 se encuentran como muestra la Figura 5.12.
Eventos: El usuario 8 se conecta pidiendo la clase 4.
Estado final: Las TE0 y TE1 se actualizan con la nueva clase 4, se agrega a la TE1 la entrada del nodo 4 y se actualiza el campo *tn=1* para el nodo 2 en la TE2 como muestra la figura 5.15.

```
P0      ses     dest    rc      dlc     dnc     tc      psh     tn      dn      dl
P0      1       1       4       4       4       y       n       0       -1      n
1 Receive join_ack packet
P1      ses     dest    rc      dlc     dnc     tc      psh     tn      dn      dl
P1      1       4       4       4       4       y       n       -1      -1      n
P1      1       2       3       3       3       y       n       1       -1      n
4 Receive join_ack packet
P4      ses     dest    rc      dlc     dnc     tc      psh     tn      dn      dl
P4      1       8       4       4       4       y       n       -1      -1      n
```

Figura 5.15 Tablas de estados.

Observaciones: De acuerdo al estado 7 caso 1 de la conexión (sección 3.2.4) el estado final es correcto.

i) Estado 7 de la conexión, caso 3.
Topología: Figura 5.5.
Estado inicial: La TE1 se encuentra como muestra la Figura 5.16.

```
P1      ses     dest    rc      dlc     dnc     tc      psh     tn      dn      dl
P1      1       2       3       3       3       y       n       -1      -1      n
P1      1       4       3       3       3       y       n       -1      -1      n
```

Figura 5.16 Tablas de estados.

Eventos: El usuario 6 se conecta pidiendo la clase 5.
Estado final: La TE1 se actualiza con la nueva clase 5 y se actualiza el campo *tn=1* para el nodo 4 como muestra la figura 5.17.

```
P1      ses     dest    rc      dlc     dnc     tc      psh     tn      dn      dl
P1      1       2       5       5       5       y       n       -1      -1      n
P1      1       4       3       3       3       y       n       1       -1      n
```

Figura 5.17 Tablas de estados.

Observaciones: De acuerdo al estado 7 caso 3 de la conexión (sección 3.2.4) el estado final es correcto.

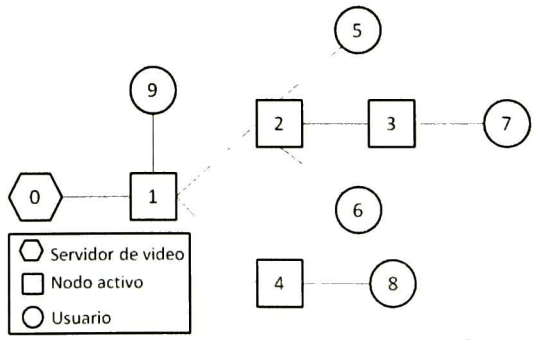


Figura 5.18 Topología de pruebas de validación 2.

j) Estado 7 de la conexión, caso 5.
Topología: Figura 5.18.
Estado inicial: La TE1 se encuentra como muestra la Figura 5.19.

P1	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P1	1	9	4	4	4	y	n	-1	-1	n
P1	1	2	3	3	3	y	n	1	-1	n
P1	1	4	3	3	3	y	n	1	-1	n

Figura 5.19 Tablas de estados.

Eventos: El usuario 6 se conecta pidiendo la clase 5.
Estado final: La TE1 se actualiza con la nueva clase 5 y se actualiza el campo *tn=1* para el nodo 9 y *tn=-1* para el nodo 2 como muestra la figura 5.20.

P1	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P1	1	4	3	3	3	y	n	1	-1	n
P1	1	2	5	5	5	y	n	-1	-1	n
P1	1	9	4	4	4	y	n	1	-1	n

Figura 5.20 Tablas de estados.

Observaciones: De acuerdo al estado 7 caso 5 de la conexión (sección 3.2.4) el estado final es correcto.

Pruebas de los estados de la desconexión.

a) Estado 1 de la desconexión.
Topología: Figura 5.5.
Estado inicial: Las TE0, TE1 y TE2 se encuentran como muestra la Figura 5.6.
Eventos: El usuario 6 se desconecta de la sesión 1.
Estado final: En las TE0, TE1 y TE2 se eliminan las entradas y quedan vacías como muestra la figura 5.21.

2 remove 5 transcode in -1 with vclass 3

P2	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl

1 remove 2 transcode in -1 with vclass 3

P1	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl

0 remove 1 transcode in 0 with vclass 3

P0	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl

Figura 5.21 Tablas de estados.

Observaciones: De acuerdo al estado 1 de la desconexión (sección 3.2.6) el estado final es correcto.

b) Estado 2.1 de la desconexión.
Topología: Figura 5.5.
Estado inicial: Las TE1, TE2 y TE3 se encuentran como muestra la Figura 5.22.

P1	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P1	1	2	5	5	5	y	n	-1	-1	n
P2	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P2	1	5	5	5	5	y	n	-1	-1	n
P2	1	6	3	3	3	y	n	2	-1	n
P2	1	3	4	4	3	y	n	2	2	n
P3	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P3	1	7	4	4	3	y	n	-1	2	n

Figura 5.22 Tablas de estados.

Eventos: El usuario 5 se desconecta de la sesión 1.

Estado final: En la TE1 se actualiza el valor de la nueva clase 4, en la TE2 se elimina la entrada del nodo 5 y se actualizan los campos *dnc=4* y *dn=-1* para el nodo 3 y en la TE3 se actualiza la nueva clase 4 y el campo *dn=-1* como muestra la figura 5.23.

P1	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P1	1	2	4	4	4	y	n	-1	-1	n
P2	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P2	1	6	3	3	3	y	n	2	-1	n
P2	1	3	4	4	4	y	n	-1	-1	n
P3	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P3	1	7	4	4	4	y	n	-1	-1	n

Figura 5.23 Tablas de estados.

Observaciones: De acuerdo al estado 2.1 de la desconexión (sección 3.2.6) el estado final es correcto.

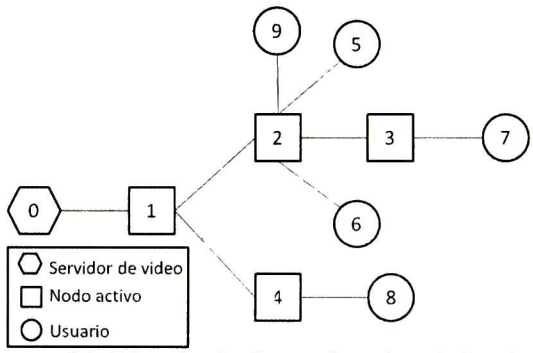


Figura 5.24 Topología de pruebas de validación 3.

c) Estado 2.2 de la desconexión.
Topología: Figura 5.24.
Estado inicial: La TE2 se encuentra como muestra la Figura 5.25.

P2	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P2	1	5	5	5	5	y	n	-1	-1	n
P2	1	6	3	3	3	y	n	2	-1	n
P2	1	3	1	1	1	y	n	2	-1	n
P2	1	9	2	2	1	y	n	2	2	n

Figura 5.25 Tablas de estados.

Eventos: El usuario 5 se desconecta de la sesión 1.
Estado final: En la TE2 se elimina la entrada del nodo 5 y se actualizan los campos *dnc=4* y *dn=-1* para el nodo 9 y el campo *tn=-1* para el nodo 6 como muestra la figura 5.26.

P2	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P2	1	3	1	1	1	y	n	2	-1	n
P2	1	6	3	3	3	y	n	-1	-1	n
P2	1	9	2	2	2	y	n	2	-1	n

Figura 5.26 Tablas de estados.

Observaciones: De acuerdo al estado 2.2 de la desconexión (sección 3.2.6) el estado final es correcto.

d) Estado 2.3 de la desconexión.
Topología: Figura 5.5.
Estado inicial: Las TE2 y TE3 se encuentran como muestra la Figura 5.27.

```
P0      ses     dest    rc      dlc     dnc     tc      psh     tn      dn      dl
P0      1       1       5       5       5       y       n       -1      -1      n
1 Receive join_ack packet
P1      ses     dest    rc      dlc     dnc     tc      psh     tn      dn      dl
P1      1       2       5       5       5       y       n       -1      -1      n
P2      ses     dest    rc      dlc     dnc     tc      psh     tn      dn      dl
P2      1       5       5       5       5       y       n       -1      -1      n
P2      1       3       3       3       3       y       n       2       -1      n
P2      1       6       1       1       1       y       n       2       -1      n
```

Figura 5.27 Tablas de estados.

Eventos: El usuario 5 se desconecta de la sesión 1.
Estado final: En la TE1 se actualiza el valor de la nueva clase 3 y en la TE2 se elimina la entrada del nodo 5, se actualiza el valor de la nueva clase y el campo *tn=-1* para el nodo 3 como muestra la figura 5.28.

```
P0      ses     dest    rc      dlc     dnc     tc      psh     tn      dn      dl
P0      1       1       3       3       3       y       n       0       -1      n
1 User receive downclass ack pkt 3
P1      ses     dest    rc      dlc     dnc     tc      psh     tn      dn      dl
P1      1       2       3       3       3       y       n       -1      -1      n
2 User receive downclass ack pkt 3
P2      ses     dest    rc      dlc     dnc     tc      psh     tn      dn      dl
P2      1       6       1       1       1       y       n       2       -1      n
P2      1       3       3       3       3       y       n       -1      -1      n
```

Figura 5.28 Tablas de estados.

Observaciones: De acuerdo al estado 2.3 de la desconexión (sección 3.2.6) el estado final es correcto.

e) Estado 3 de la desconexión.
Topología: Figura 5.5.
Estado inicial: La TE2 se encuentra como muestra la Figura 5.29.

```
P2      ses     dest    rc      dlc     dnc     tc      psh     tn      dn      dl
P2      1       5       5       5       5       y       n       -1      -1      n
P2      1       3       3       3       3       y       n       2       -1      n
P2      1       6       5       5       5       y       n       -1      -1      n
```

Figura 5.29 Tablas de estados.

Eventos: El usuario 5 se desconecta de la sesión 1.
Estado final: En la TE2 se elimina la entrada del nodo 5 como muestra la figura 5.30.

```
P2      ses     dest    rc      dlc     dnc     tc      psh     tn      dn      dl
P2      1       3       3       3       3       y       n       2       -1      n
P2      1       6       5       5       5       y       n       -1      -1      n
```

Figura 5.30 Tablas de estados.

Observaciones: De acuerdo al estado 3 de la desconexión (sección 3.2.6) el estado final es correcto.

f) Estado 4 de la desconexión.
Topología: Figura 5.24.
Estado inicial: La TE2 se encuentra como muestra la Figura 5.31.

P2	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P2	1	5	5	5	5	y	n	-1	-1	n
P2	1	6	3	3	3	y	n	2	-1	n
P2	1	3	1	1	1	y	n	2	-1	n
P2	1	9	2	2	1	y	n	2	2	n

Figura 5.31 Tablas de estados.

Eventos: El usuario 6 se desconecta de la sesión 1.
Estado final: En la TE2 se elimina la entrada del nodo 6 y se restaura la clase del nodo 9 cambiando los campos *dnc=2* y *dn=-1* como muestra la figura 5.32.

P2	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P2	1	5	5	5	5	y	n	-1	-1	n
P2	1	3	1	1	1	y	n	2	-1	n
P2	1	9	2	2	2	y	n	2	-1	n

Figura 5.32 Tablas de estados.

Observaciones: De acuerdo al estado 4 de la desconexión (sección 3.2.6) el estado final es correcto.

g) Estado 5.1 de la desconexión.
Topología: Figura 5.24.
Estado inicial: La TE2 se encuentra como muestra la Figura 5.33.

P2	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P2	1	5	5	5	5	y	n	-1	-1	n
P2	1	6	2	2	2	y	n	2	-1	n
P2	1	3	3	3	2	y	n	2	2	n
P2	1	9	4	4	2	y		2	2	n

Figura 5.33 Tablas de estados.

Eventos: El usuario 6 se desconecta de la sesión 1.
Estado final: En la TE2 se elimina la entrada del nodo 6, se aumenta la clase de los nodos 3 y 9 marcando *dnc=3* y se modifica el campo *dn=-1* para el nodo 3 como muestra la figura 5.34.

P2	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P2	1	5	5	5	5	y	n	-1	-1	n
P2	1	3	3	3	3	y	n	2	-1	n
P2	1	9	4	4	3	y	n	2	2	n

Figura 5.34 Tablas de estados.

Observaciones: De acuerdo al estado 5.1 de la desconexión (sección 3.2.6) el estado final es correcto.

h) Estado 5.2 de la desconexión.
Topología: Figura 5.24.
Estado inicial: La TE2 se encuentra como muestra la Figura 5.35.

P2	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P2	1	5	5	5	5	y	n	-1	-1	n
P2	1	6	2	2	2	y	n	2	-1	n
P2	1	3	2	2	2	y	n	2	-1	n
P2	1	9	4	4	2	y	n	2	2	n

Figura 5.35 Tablas de estados.

Eventos: El usuario 6 se desconecta de la sesión 1.
Estado final: En la TE2 se elimina la entrada del nodo 6 como muestra la figura 5.36.

P2	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P2	1	5	5	5	5	y	n	-1	-1	n
P2	1	3	2	2	2	y	n	2	-1	n
P2	1	9	4	4	2	y	n	2	2	n

Figura 5.36 Tablas de estados.

Observaciones: De acuerdo al estado 5.2 de la desconexión (sección 3.2.6) el estado final es correcto.

Pruebas de la difusión y el restablecimiento de la transcodificación.

Se buscan probar los mecanismos de difusión de la transcodificación así como el restablecimiento de la transcodificación ante la liberación de recursos descritos en la sección 3.2.5.

a) Difusión hacia arriba.
Topología: Figura 5.5.
Estado inicial: Las TE1 y TE2 se encuentran como muestra la Figura 5.35.

P1	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P1	1	2	5	5	5	y	n	-1	-1	n
P2	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P2	1	5	5	5	5	y	n	-1	-1	n
P2	1	3	3	3	3	y	n	2	-1	n

Figura 5.37 Tablas de estados.

Eventos: El usuario 6 se conecta pidiendo la clase 1 y el nodo 2 no tiene recursos suficientes.
Estado final: En la TE2 se agrega la entrada del nodo 6 marcando los campos *tc=n* y *tn=1* y Se agrega una segunda entrada a la TE1 para 2 con la clase 1 marcando los campos *psh=y* y *tn=1* como muestra la figura 5.36.

P2	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P2	1	5	5	5	5	y	n	-1	-1	n
P2	1	3	3	3	3	y	n	2	-1	n
P2	1	6	1	1	1	n	n	1	-1	n
1 Receive data packet num 0. session=1, tam 1000, vclass=5										
P1	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P1	1	2	5	5	5	y	n	-1	-1	n
P1	1	2	1	1	1	y	y	1	-1	n

Figura 5.38 Tablas de estados.

Observaciones: De acuerdo a la difusión hacia arriba (sección 3.2.5) el estado final es correcto.

b) Restablecimiento hacia arriba.
Topología: Figura 5.5.
Estado inicial: Las TE1 y TE2 se encuentran como muestra la Figura 5.36.
Eventos: El usuario 7 se desconecta de la sesión y libera recursos en el nodo 2.
Estado final: En la TE2 se elimina la entrada del nodo 3 y se actualizan los campos *tc=y* y *tn=2* y en la TE1 se elimina la entrada para 2 donde *psh=y* como muestra la figura 5.37.

P2	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P2	1	5	5	5	5	y	n	-1	-1	n
P2	1	6	1	1	1	y	n	2	-1	n

P1	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P1	1	2	5	5	5	y	n	-1	-1	n

Figura 5.39 Tablas de estados.

Observaciones: De acuerdo al restablecimiento de la difusión hacia arriba (sección 3.2.5) el estado final es correcto.

c) Difusión hacia abajo.
Topología: Figura 5.5.
Estado inicial: La TE2 se encuentra como muestra la Figura 5.38 y la TE3 se encuentra vacía.

P2	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P2	1	5	5	5	5	y	n	-1	-1	n
P2	1	6	3	3	3	y	n	2	-1	n

Figura 5.40 Tablas de estados.

Eventos: El usuario 7 se conecta pidiendo la clase 1 y el nodo 2 no tiene recursos suficientes.
Estado final: En la TE2 se agregan dos entradas para el nodo 3, una con la clase 5 para que se realice la transcodificación marcando los campos *tc=n* y *tn*=3 y otra con la clase pedida 1 marcando los campos *tc=n, psh=y* y *tn=3.* En la TE3 se agrega la entrada para el nodo 7 marcando los campos *psh=y* y *tn=3* como muestra la figura 5.39.

P2	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P2	1	5	5	5	5	y	n	-1	-1	n
P2	1	6	3	3	3	y	n	2	-1	n
P2	1	3	5	5	5	n	n	3	-1	n
P2	1	3	1	1	1	n	y	3	-1	n
P3	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P3	1	7	1	1	1	y	y	3	-1	n

Figura 5.41 Tablas de estados.

Observaciones: De acuerdo a la difusión hacia abajo (sección 3.2.5) el estado final es correcto.

d) Restablecimiento hacia abajo.
Topología: Figura 5.5.
Estado inicial: Las TE2 y TE3 se encuentran como muestra la Figura 5.39.
Eventos: El usuario 6 se desconecta de la sesión y libera recursos en el nodo 2.
Estado final: En la TE2 se elimina la entrada del nodo 6 y la estrada del nodo 3 donde *tc=n* y *psh=n* y se actualiza l entrada del nodo 3 donde *tc=n* y *psh=y* con *tc=y, psh=n* y *tn=2* y en la TE3 se cambian los campos *psh=y* y *tn=-1* para el nodo 7 como muestra la figura 5.40.

P2	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P2	1	5	5	5	5	y	n	-1	-1	n
P2	1	3	1	1	1	y	n	2	-1	n
P3	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P3	1	7	1	1	1	y	n	-1	-1	n

Figura 5.42 Tablas de estados.

Observaciones: De acuerdo al restablecimiento de la difusión hacia abajo (sección 3.2.5) el estado final es correcto.

e) Difusión hacia un lado.
Topología: Figura 5.5.
Estado inicial: Las TE2 y TE3 se encuentran como muestra la Figura 5.41.

P2	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P2	1	5	5	5	5	y	n	-1	-1	n
P2	1	3	3	3	3	y	n	2	-1	n
3 Receive join_ack packet										
P3	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P3	1	7	3	3	3	y	n	-1	-1	n

Figura 5.43 Tablas de estados.

Eventos: El usuario 6 se conecta pidiendo la clase 1 y el nodo 2 no tiene recursos suficientes.
Estado final: En la TE2 se agrega la entrada del nodo 6 marcando los campos *tc=n* y *tn=3* y se agrega otra entrada del nodo 3 con la clase de entrada para que realice la transcodificación marcando el campo *tc=n*, en la TE3 se agrega una entrada para el nodo 2 con la clase 1 marcando los campos *psh=y* y *tn=3* como muestra la figura 5.42.

P2	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P2	1	5	5	5	5	y	n	-1	-1	n
P2	1	3	3	3	3	y	n	2	-1	n
P2	1	6	1	1	1	n	n	3	-1	n
P2	1	3	5	5	5	n	n	-1	-1	n
P3	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P3	1	7	3	3	3	y	n	-1	-1	n
P3	1	2	1	1	1	y	y	3	-1	n

Figura 5.44 Tablas de estados.

Observaciones: De acuerdo a la difusión hacia un lado (sección 3.2.5) el estado final es correcto.

f) Restablecimiento hacia un lado.
Topología: Figura 5.5.
Estado inicial: Las TE2 y TE3 se encuentran como muestra la Figura 5.42.
Eventos: El usuario 7 se desconecta de la sesión y libera recursos en el nodo 2.
Estado final: En la TE2 se eliminan las dos entradas del nodo 3 y se actualizan los campos *tc=y* y *tn=2* para el nodo 6 y en la TE3 se elimina la entrada para el nodo 7 como muestra la figura 5.43.

P2	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P2	1	5	5	5	5	y	n	-1	-1	n
P2	1	6	1	1	1	y	n	2	-1	n
P3	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl

Figura 5.45 Tablas de estados.

Observaciones: De acuerdo al restablecimiento de la difusión hacia un lado (sección 3.2.5) el estado final es correcto.

Pruebas de la degradación y restablecimiento de la clase debido a la falta de recursos del nodo activo.

a) Degradación debido a la falta de recursos del nodo.
Topología: Figura 5.5.
Estado inicial: La TE2 se encuentra como muestra la Figura 5.44.

P2	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P2	1	5	5	5	5	y	n	-1	-1	n
P2	1	3	1	1	1	y	n	2	-1	n

Figura 5.46 Tablas de estados.

Eventos: El usuario 6 se conecta pidiendo la clase 3 y no hay recursos en el nodo 2 ni en sus vecinos.
Estado final: En la TE2 se agrega la entrada del nodo 6 marcando los campos *dnc=1*, *tn=2* y *dn=2* como muestra la figura 5.45.

P2	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P2	1	5	5	5	5	y	n	-1	-1	n
P2	1	3	1	1	1	y	n	2	-1	n
P2	1	6	3	3	1	y	n	2	2	n

Figura 5.47 Tablas de estados.

Observaciones: De acuerdo a la degradación de la clase debido a la falta de recursos del nodo (sección 3.2.5) el estado final es correcto.

b) Restablecimiento de la degradación debido a la falta de recursos del nodo.
Topología: Figura 5.5.
Estado inicial: La TE2 se encuentra como muestra la Figura 5.45.
Eventos: El usuario 7 se desconecta de la sesión.
Estado final: En la TE2 se elimina la entrada del nodo 3 y se actualizan los campos *dnc=3* y *dn=-1* como muestra la figura 5.46.

P2	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P2	1	5	5	5	5	y	n	-1	-1	n
P2	1	6	3	3	3	y	n	2	-1	n

Figura 5.48 Tablas de estados.

Observaciones: De acuerdo al restablecimiento de la degradación de la clase debido a la falta de recursos del nodo (sección 3.2.5) el estado final es correcto.

Pruebas de la degradación y restablecimiento de la clase debido a la falta de recursos del enlace.

a) Degradación debido a la falta de recursos del enlace.
Topología: Figura 5.5.
Estado inicial: Las TE1, TE2 y TE3 se encuentran como muestra la Figura 5.47. Como se ve en las tablas todos los nodos están conectados.

P1	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P1	1	2	5	5	5	y	n	-1	-1	n
P1	1	4	3	3	3	y	n	1	-1	n
P2	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P2	1	6	5	5	5	y	n	-1	-1	n
P2	1	3	1	1	1	y	n	2	-1	n
P2	1	5	4	4	4	y	n	2	-1	n
P3	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P3	1	7	1	1	1	y	n	-1	-1	n

Figura 5.49 Tablas de estados.

Eventos: El usuario 8 inicia una transmisión de tráfico CBR con el usuario 7 congestionando el enlace entre los nodos 1 y 2.

Estado final: La degradación es paulatina mientras la tasa de pérdida de paquetes no se encuentre dentro del rango de aceptación. En la primera degradación, en la TE1 se modifican los campos *dlc=4* y *dl=y* para el nodo 2, y en la TE2 se modifica el campo *dlc=4* para el nodo 6 como muestra la figura 5.48a. Y en la segunda degradación, en la TE1 se modifica el campo *dlc=3* para el nodo 2, y en la TE2 se modifica el campo *dlc=3* para los nodos 5 y 6 como muestra la figura 5.48b.

a)

P1	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P1	1	4	3	3	3	y	n	1	-1	n
P1	1	2	5	4	5	y	n	1	-1	y
P2	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P2	1	3	1	1	1	y	n	2	-1	n
P2	1	5	4	4	4	y	n	2	-1	n
P2	1	6	5	4	5	y	n	-1	-1	n

b)

P1	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P1	1	4	3	3	3	y	n	1	-1	n
P1	1	2	5	3	5	y	n	1	-1	y
P2	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P2	1	3	1	1	1	y	n	2	-1	n
P2	1	5	4	3	4	y	n	2	-1	n
P2	1	6	5	3	5	y	n	-1	-1	n

Figura 5.50 Tablas de estados.

Observaciones: De acuerdo a la degradación de la clase debido a la falta de recursos del enlace (sección 3.2.5) el estado final es correcto.

b) Restablecimiento de la degradación debido a la falta de recursos del enlace.
Topología: Figura 5.5.
Estado inicial: Las TE1 y TE2 se encuentran como muestra la Figura 5.48b.
Eventos: El usuario 8 termina la transmisión con el usuario 7.
Estado final: La restauración es paulatina mientras se alcanza la clase deseada por el usuario. En la primera restauración, en la TE1 se modifica el campo *dlc=4* para el nodo 2, y en la TE2 se modifican el campo *dlc=4* para los nodos 5 y 6 como muestra la figura 5.49a. Y en la segunda restauración, en la TE1 se modifican los campos *dlc=5* y *dl=n* para el nodo 2, y en la TE2 se modifica el campo *dlc=5* para el nodo 6 como muestra la figura 5.49b.

a)

P1	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P1	1	4	3	3	3	y	n	1	-1	n
P1	1	2	5	4	5	y	n	1	-1	y
P2	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P2	1	3	1	1	1	y	n	2	-1	n
P2	1	5	4	4	4	y	n	2	-1	n
P2	1	6	5	4	5	y	n	-1	-1	n

b)

P1	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P1	1	4	3	3	3	y	n	1	-1	n
P1	1	2	5	5	5	y	n	-1	-1	n
P2	ses	dest	rc	dlc	dnc	tc	psh	tn	dn	dl
P2	1	3	1	1	1	y	n	2	-1	n
P2	1	5	4	4	4	y	n	2	-1	n
P2	1	6	5	5	5	y	n	-1	-1	n

Figura 5.51 Tablas de estados.

Observaciones: De acuerdo al restablecimiento de la degradación de la clase debido a la falta de recursos del enlace (sección 3.2.5) el estado final es correcto.

5.2.3 Escenarios de desempeño.

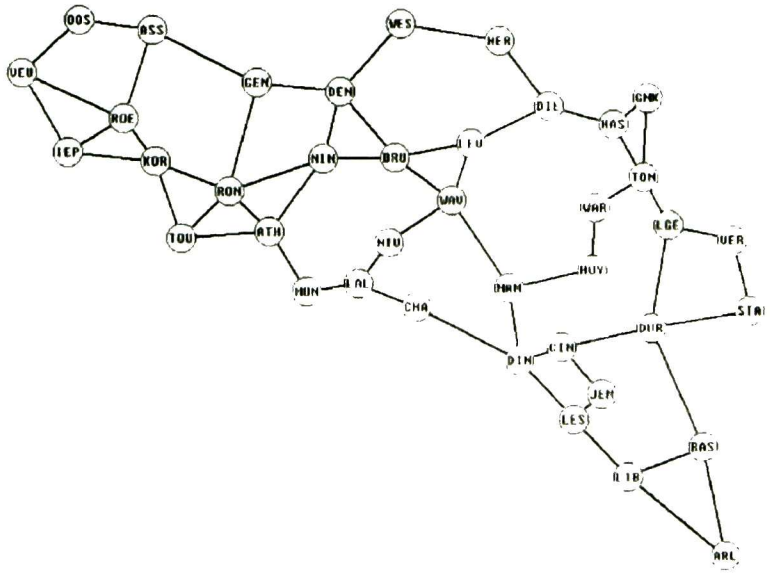
Definición de los escenarios de prueba.

La figura 5.52 (a) muestra la topología de la red de prueba que se utiliza en [Lambrecht06] para medir el desempeño de su solución propuesta y compararla con la técnica de transmisión de video "simulcasting". Y la figura 5.52 (b) muestra el mismo escenario de prueba creado en NS2 y mostrado con el visualizador NAM. Se optó por utilizar esta red de prueba por ser el trabajo que más se asemeja al nuestro, aunque en sus publicaciones no se especifican totalmente los escenarios.

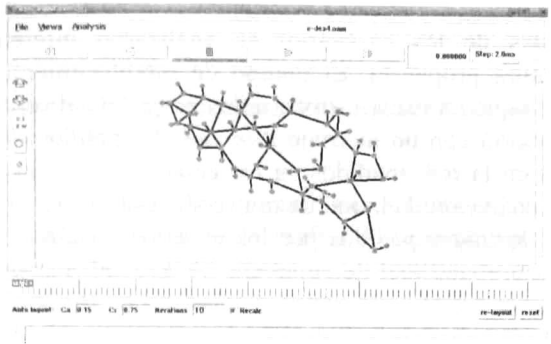
La red de prueba consta de 39 nodos activos, cada nodo activo tiene un usuario capaz de conectarse a la sesión de video, el servidor de video se localiza en el centro de la red conectado al nodo WAV (figura 5.52(a)). Todos los enlaces de la red son full dúplex con un retardo de 10 ms y las colas utilizadas son de tipo *Droptail*. El límite superior para la pérdida de paquetes (*top_limit*) es de 5 paquetes por segundo, y el intervalo de medición de la pérdida de paquetes es de un segundo.

La traza de video original utilizada en la simulación fue "Verbose_Simpsons_256.dat" que se encuentra disponible en [wasu]. La cual se encuentra codificada en el estándar H.263 con una tasa de bits de 256Kbps y un formato QCIF. La clase de video soportada por cada usuario es asignada aleatoriamente al principio de cada simulación. Se utilizaron 5 clases de video, con las tasas de bits de 256Kbps, 205Kbps, 155Kbps, 105Kbps y 51Kbps.

Para cada corrida se simularon 60 segundos reales, el intervalo de tiempo entre cada evento de conexión ó desconexión fue de 0.3 segundos para evitar conflictos con la llegada de los paquetes. En todas las simulaciones se ejecutaron 39 eventos de intentos de conexión o de desconexión. Por lo tanto, en 12 segundos se ejecutan todos los eventos, quedando 48 segundos de la transmisión con el árbol multicast estable o sin cambios.



(a)



(b)

Figura 5.52 Topología para pruebas de desempeño (a)[Lambrecht06], (b) NS2.

Como resultado de la combinación de las cinco propiedades consideradas para los escenarios (D, RN, RE, AN y AE) se obtuvieron 32 de ellos como se observa en la tabla 5.4. En la tabla la columna *D* especifica si los usuarios se comportan de forma dinámica o estática. Las columnas *RN* y *RE* especifican si existen los recursos suficientes en los nodos y en los enlaces, respectivamente. Las columnas *AN* y *AE* especifican si los mecanismos de adaptación a los recursos de los nodos y de los enlaces, respectivamente, se encuentran activos.

En los escenarios estáticos los 39 usuarios intentaban conectarse una vez cada uno en orden aleatorio, pudiendo o no tener éxito en la conexión. En cambio en los escenarios dinámicos existen 39 eventos que pueden ser de conexión o de desconexión de un nodo escogido aleatoriamente para cada evento.

Para simular los recursos de los nodos se utilizó una variable con la cantidad de transcodificadores que el nodo puede albergar. En los escenarios donde los recursos de los nodos son insuficientes (RN=no), los recursos son asignados aleatoriamente entre 0 y 4. Y en los que son suficientes (RN=si) a todos los nodos les son asignados 4 recursos, que de acuerdo con la definición de un escenario ideal es el mínimo necesario para satisfacer a los usuarios con 5 clases de video.

En los escenarios donde los recursos de los enlaces son suficientes (RE=si) existen dos casos, en el primer caso existe la difusión de la computación, esto es, los escenarios no tienen recursos de los nodos (RN=no) y se encuentra activada la adaptación a los nodos (AN=si), en este caso los enlaces tienen un ancho de banda de 512Kbps suficiente para transportar las dos clases de video de mayor tasa de bits. En el segundo caso no existe difusión de la computación, esto es en los escenarios que no caen en el primer caso, en este caso el ancho de banda de los enlaces es de 256Kbps suficiente para transportar la clase de video mayor. También en los escenarios donde los recursos de los enlaces no son suficientes (RE=no) existen los dos casos, en el primero donde existe la difusión (RN=no y AN=si), todos los enlaces tienen un ancho de banda de 256Kbps suficiente sólo para transportar la clase de video más alta provocando congestionamiento en los enlaces utilizados para la difusión. Y en el caso donde no existe difusión se genera tráfico cruzado desde el usuario del nodo WAV hacia el usuario en el nodo VEU de la figura 5.53(a), saturando el enlace del nodo WAV al nodo BRU. El tráfico cruzado generado es de 105 Kbps equivalente al envío de una clase 2 de otra sesión de video.

Durante las corridas de los escenarios se analizaron diferentes métricas para medir el desempeño de la arquitectura propuesta. El tiempo de establecimiento de la conexión (TC), es el tiempo que transcurre desde que un usuario envía un mensaje *join* al servidor de video hasta que recibe la confirmación de la conexión con un mensaje *join_ack*. La pérdida de paquetes (PP), es el número total de paquetes perdidos en la red, medidos en los enlaces de entrada de cada nodo. El número de transcodificadores (NTC), es la cantidad total de transcodificadores en la red necesarios para adaptar el video original a todos los formatos pedidos por los usuarios cuando todos los usuarios de la red se encuentran conectados. La probabilidad de conexión (PC), es el porcentaje de usuarios que lograron conectarse exitosamente a la sesión de video deseada. Los resultados de las simulaciones son mostrados en las últimas 4 columnas de la tabla 5.4.

NE	D	RN	RE	AN	AE	TC	PP	PC	NTC
1	No	No	No	No	No	0.05761	129.34	80.7%	NA
2	No	No	No	No	Si	0.05879	61.76	77.5%	NA
3	No	No	No	Si	No	0.06279	3015.36	99.5%	NA
4	No	No	No	Si	Si	0.06859	2587.65	99.4%	NA
5	No	No	Si	No	No	0.05610	0	79.9%	NA
6	No	No	Si	No	Si	0.05623	0	83.0%	NA
7	No	No	Si	Si	No	0.05739	0	99.4%	NA
8	No	No	Si	Si	Si	0.05741	0	99.6%	NA
9	No	Si	No	No	No	0.05906	146.84	100%	26.46
10	No	Si	No	No	Si	0.05702	57.16	100%	23
11	No	Si	No	Si	No	0.05754	154.2	100%	25.88
12	No	Si	No	Si	Si	0.05738	56.88	100%	23.88
13	No	Si	Si	No	No	0.05644	0	100%	26.2
14	No	Si	Si	No	Si	0.05712	0	100%	26.38
15	No	Si	Si	Si	No	0.05782	0	100%	26.12
16	No	Si	Si	Si	Si	0.05674	0	100%	25.33
17	Si	No	No	No	No	0.06727	118.25	76.6%	NA
18	Si	No	No	No	Si	0.06831	61.63	77.4%	NA
19	Si	No	No	Si	No	0.06940	1857.40	99.6%	NA
20	Si	No	No	Si	Si	0.06924	1717.77	99.9%	NA
21	Si	No	Si	No	No	0.06527	0	83.1%	NA
22	Si	No	Si	No	Si	0.06664	0	80.4%	NA
23	Si	No	Si	Si	No	0.06566	0	99.5%	NA
24	Si	No	Si	Si	Si	0.06585	0	99.3%	NA
25	Si	Si	No	No	No	0.06602	130.88	100%	NA
26	Si	Si	No	No	Si	0.06836	50.62	100%	NA
27	Si	Si	No	Si	No	0.06722	121.22	100%	NA
28	Si	Si	No	Si	Si	0.06646	53.22	100%	NA
29	Si	Si	Si	No	No	0.06577	0	100%	NA
30	Si	Si	Si	No	Si	0.06640	0	100%	NA
31	Si	Si	Si	Si	No	0.06958	0	100%	NA
32	Si	Si	Si	Si	Si	0.07103	0	100%	NA

Tabla 5.2 Escenarios para pruebas de desempeño.

5.3 Análisis de resultados de las pruebas de desempeño.

En esta sección se analizan los resultados obtenidos de las simulaciones y que son mostrados en la tabla 5.4. Como nota la función de aleatoriedad de NS2 (*random( )*) utilizada en las simulaciones sigue una distribución gaussiana.

Tiempo de conexión

El tiempo de conexión promedio entre todos los escenarios calculado a partir de los resultados de la tabla 5.2 es de 62.89 milisegundos.

La figura 5.53 muestra los tiempos de conexión en cada escenario y el tiempo de conexión promedio (media). Como se puede observar en la gráfica los tiempos para los escenarios estáticos (escenarios del 1 al 16) se encuentran en su mayoría por debajo de la media, mientras que los dinámicos (escenarios del 17 al 32) se encuentran todos por encima de la media. Esto puede ser provocado por las distancia entre los usuarios y el árbol multicast de la sesión. En un escenario estático las distancias se van haciendo menores conforme se conectan los usuarios provocando que los mensajes de conexión viajen menos. Mientras que en un escenario dinámico los usuarios pueden desconectarse de la sesión y posiblemente eliminar la rama del árbol multicast a la que se encuentra conectado, entonces cuando el mismo usuario u otro usuario necesite conectarse a la sesión se verá obligado a crear toda la rama de nuevo.

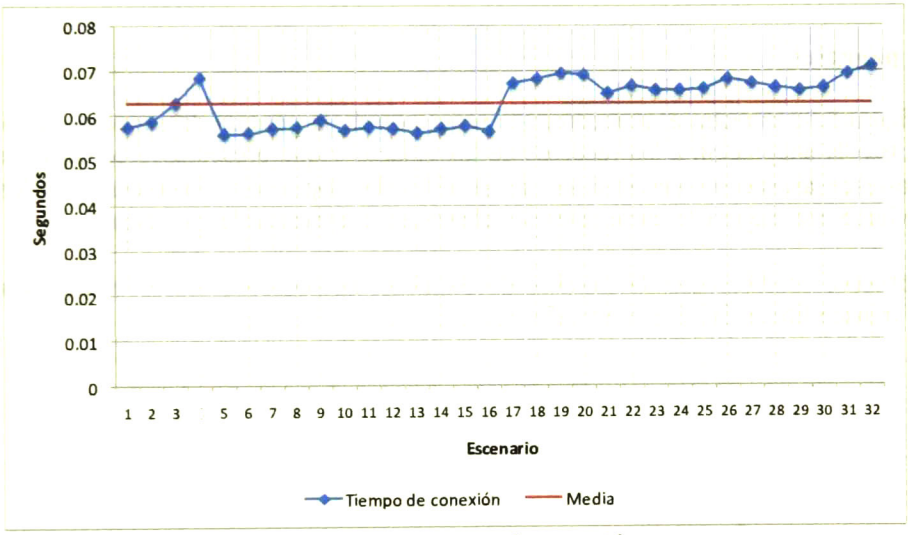


Figura 5.53 Tiempo de conexión.

La figura 5.54 muestra los resultados en el tiempo de conexión del trabajo de Duysburgh. En la figura se marca el tiempo de conexión promedio obtenido en las simulaciones, como se observa, nuestro tiempo de conexión es menor al que ellos obtuvieron con 40 nodos, esto puede ser debido a los tiempos de retardo de los enlaces que ellos utilizaron y que no especifican en su trabajo ó a que el hecho de utilizar difusión de la transcodificación en 'n' saltos hace que se retarde el proceso de conexión. En nuestra simulación se utiliza un retardo de 10ms para todos los enlaces.

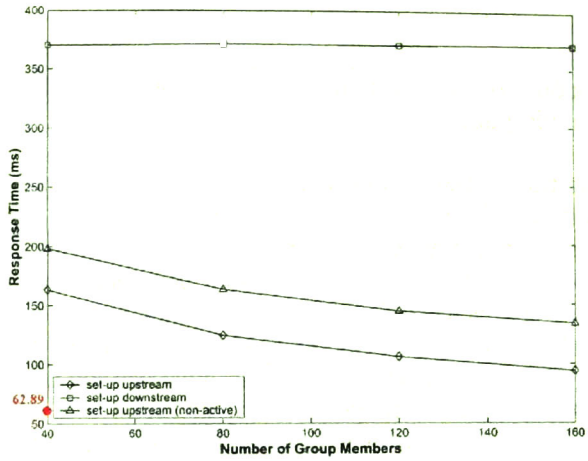


Figura 5.54 Tiempo de conexión [Duysburgh04].

Pérdida de paquetes.

La figura 5.55 muestra el porcentaje de reducción de la pérdida de paquetes de los escenarios donde se utilizó la adaptación a los recursos de los enlaces y que presentaron congestionamiento con respecto a los escenarios equivalentes donde no se utilizó la adaptación a los recursos de los enlaces. Se observa que hubo una mejora de entre el 40% y el 60% en la mayoría de los casos.

Se observó también que la pérdida de paquetes aumenta en los escenarios que presentan difusión de transcodificación (RN=no y AE=si).

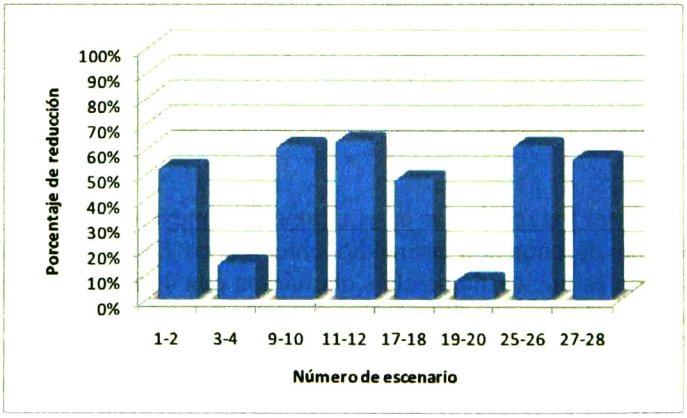


Figura 5.55 Porcentaje de reducción de la pérdida de paquetes en la red.

Probabilidad de conexión

Como se definió en la sección 5.2.1 en una red ideal todos los usuarios son satisfechos, como se puede observar en la tabla 5.2 esto se mantiene para todos los escenarios donde existen recursos suficientes de los nodos y de los enlaces (RN=si y RE=si) teniendo el 100% de probabilidad de conexión.

La figura 5.56 muestra la probabilidad de tener éxito en la conexión para los escenarios donde los recursos de los nodos son insuficientes (RN=no). Se observa que la probabilidad de conexión aumenta en los escenarios donde se utiliza la adaptación a los recursos de los nodos (AN=si) teniendo una probabilidad promedio de 99.53%, mientras que en los que no se utilizó la adaptación a los nodos (AN=no) fue en promedio del 79.83%, el aumento logrado es casi del 20%. Esto se debe a que el mecanismo de adaptación a los recursos de los nodos busca recursos en varios nodos y no únicamente en uno solo.

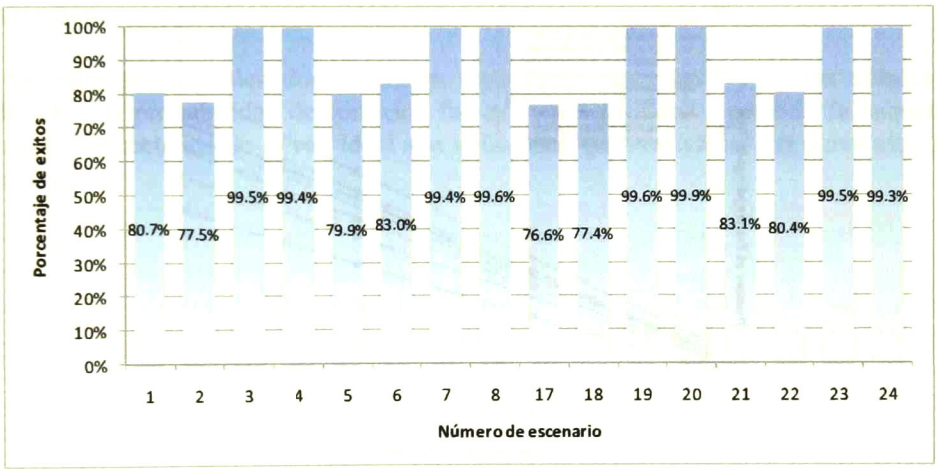


Figura 5.56 Probabilidad de conexión.

Número de Transcodificadores en la red.

La figura 5.57 muestra el número total de transcodificadores en la red después de que todos los usuarios se han conectado. Como se observa los valores varían entre los 23 y 27 transcodificadores. La figura 5.58 muestra el número de transcodificadores obtenidos por Duysburgh en su trabajo donde la difusión se realiza únicamente en un sentido y en 'n' saltos, sus resultados muestran que para redes con 40 nodos el número de transcodificadores en la red oscila entre los 20 y los 30 dependiendo del sentido de la difusión. Como se puede observar los valores obtenidos con nuestra arquitectura, donde la transcodificación se realiza en los tres sentidos pero en un solo salto, se encuentran en el mismo rango que los obtenidos por ellos.

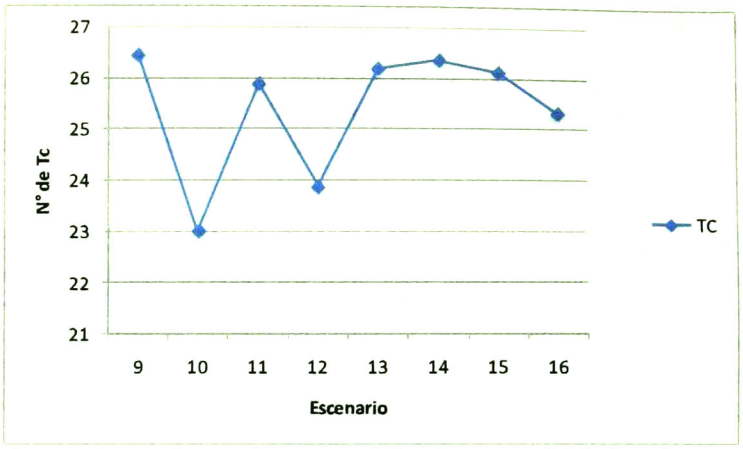


Figura 5.57 Número de transcodificadores en el árbol multicast.

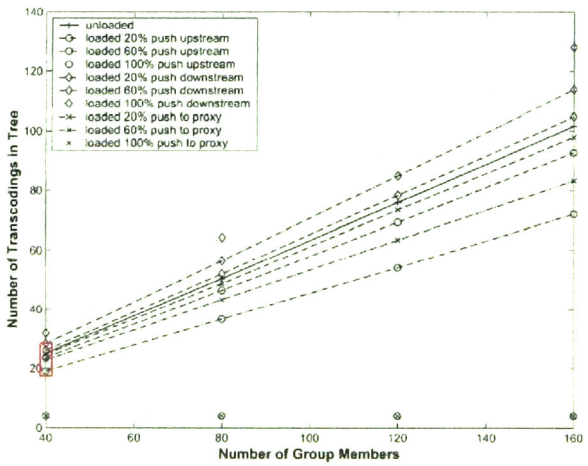


Figura 5.58 Número de transcodificadores en el árbol multicast [Duysburgh04].

5.4 Conclusiones.

Se realizó una validación de la simulación para probar que los estados de los nodos para la conexión, la desconexión y la adaptación a los recursos de los nodos y de los enlaces corresponden con los descritos en el capítulo 3. Todas las pruebas de validación fueron satisfactorias.

Se observó que los escenarios donde la adaptación a los recursos de los nodos se encontraba activada la probabilidad de obtener un flujo de video fue en promedio del 99.53%, mientras que en los escenarios donde no se activo fue en promedio del 79.83%. Se puede concluir que las técnicas de adaptación a los recursos de los nodos utilizadas en la arquitectura aumentan en casi un 20% la probabilidad de conexión cumpliendo con los objetivos de la tesis. Una desventaja observada de la difusión de la transcodificación es que conlleva a un aumento en el tráfico en la red que fue observado en escenarios donde los recursos de la red no son suficientes, aumentando la pérdida de paquetes. En los escenarios donde existen recursos de los enlaces suficientes la pérdida de paquetes fue de cero.

Se logró una reducción de la pérdida de paquetes total de la red hasta del 60% utilizando la técnica de adaptación a los recursos de los enlaces aquí propuesta. Esta técnica disminuye o degrada la clase de video de los usuarios, buscando reducir la pérdida de paquetes. De esta forma se reciben más paquetes del video en una resolución menor en lugar de recibir un video con una resolución mayor pero con fallas.

El tiempo promedio para el establecimiento de las conexiones exitosas obtenido en las simulaciones fue de 62.89 milisegundos que se encuentra por debajo de los tiempos obtenidos en el trabajo más parecido al nuestro, esto puede ser debido a los tiempos de retardo de los enlaces que ellos utilizaron y que no especifican en su trabajo ó a que el hecho de utilizar difusión de la transcodificación en 'n' saltos hace que se retarde el proceso de conexión.

En el caso del número de transcodificadores utilizados en la red se obtuvieron resultados muy semejantes a los obtenidos en el trabajo mostrado en [Duysburgh04], donde la difusión es realizada en 'n' saltos en una sola dirección a diferencia de nuestro trabajo donde se realiza en un solo salto en cualquier dirección.

En los escenarios ideales donde existen recursos de los nodos y de los enlaces suficientes (RN=Si y RE=Si) la probabilidad de conexión fue del 100% y no hubo pérdida de paquetes en la red constatando que los recursos de la red ideal son suficientes para satisfacer las demandas de todos los usuarios de la red.

CAPITULO 6

6 Conclusiones

Resumen: En este capítulo se presentan las conclusiones finales de la tesis y se expone un análisis de las limitantes de la tesis que pueden ser atacadas como trabajo a futuro.

6.1 Conclusiones de la tesis.

En esta tesis se presenta una arquitectura que ofrece una solución al problema de la heterogeneidad que afecta al desempeño de las transmisiones de video en las redes convencionales. Se muestra una comparación de los trabajos más importantes en cuanto a la transmisión de video se refiere y presenta un análisis detallado de los trabajos relacionados con el uso de la transcodificación en redes activas.

Se desarrolló una arquitectura distribuida que permite que los usuarios se conecten y desconecten en cualquier momento de una transmisión de video, además, con el uso de una arquitectura distribuida se elimina el retardo que ocasiona el obtener un mapa global de la red necesario en las arquitecturas centralizadas, esta arquitectura fue descrita en el capítulo 3. En la arquitectura los nodos activos se comportan como agentes reactivos que reaccionan a las señales de entrada o mensajes de control de acuerdo al estado en el que se encuentren, en la tesis se detallan varios de los estados en los que un nodo activo puede estar a la llegada de un mensaje y se describen las acciones que debe realizar el nodo activo. La arquitectura implementa mecanismos de adaptación a la falta de recursos tanto en los nodos como en los enlaces de la red, también incorpora mecanismos de restablecimiento ante la liberación de recursos en los nodos o en los enlaces. Los mecanismos de adaptación aumentan la probabilidad de conexión y disminuyen la pérdida de paquetes en la red. Los métodos de restablecimiento tratan de asemejar el árbol multicast de la sesión de video al árbol multicast que se crearía si no existieran los mecanismos de adaptación.

En el capítulo 4 se realizó un modelo de redes de Petri para la arquitectura propuesta donde se estudiaron las propiedades de libertad de bloqueos, acotamiento y vivacidad. Con esto se probó que la arquitectura no se bloquea en ningún estado, que el número de estados no crece indefinidamente y que siempre es posible llegar a un estado desde cualquier otro.

Para medir el desempeño de la arquitectura propuesta se realizaron varias simulaciones en NS2. La simulación fue validada mediante varias pruebas de funcionamiento donde se compararon los estados de los nodos obtenidos durante la simulación con los estados definidos en el capítulo 3. De acuerdo con los resultados obtenidos en las simulaciones: las técnicas de adaptación a los recursos de los nodos utilizadas en la arquitectura aumentan en casi un 20% la probabilidad de conexión; se logró una reducción de la pérdida de paquetes total de la red hasta del 60% utilizando la técnica de adaptación a los recursos de los enlaces aquí propuesta; el tiempo de establecimiento de la conexión promedio obtenido en las simulaciones fue de 62.89 milisegundos que se encuentra por debajo de los tiempos obtenidos en el trabajo que más se asemeja al nuestro mostrado en [Duysburgh04]; y el número total de transcodificadores utilizados en la red fue muy semejante a los obtenidos en dicho trabajo también.

La tabla 6.1 muestra los trabajos más relacionados con esta tesis, como se observa únicamente el trabado de *self-organising video* toma en cuenta los recursos de los enlaces, a diferencia de nuestra arquitectura el establecimiento de los nodos para la adaptación y la medición del tráfico de la red se hacen de forma centralizada, lo que ocasiona que el trabajo no tenga una adaptación dinámica. En cuando a la adaptación de los recursos de los nodos varios trabajos utilizan la difusión como método, en *self-organising video* los nodos a los cuales se va a difundir la computación se calculan de forma centralizada provocando que la adaptación no sea dinámica. En ARTES 1 la distribución de la computación se establece a priori entre los nodos que se encuentran cerca del satélite siendo también una adaptación no dinámica. En *Duysburgh active framework* la difusión se lleva a cabo de manera dinámica igual que en la arquitectura aquí propuesta, con la diferencia en que ellos realizan la difusión en un solo sentido y dando 'n' saltos y nosotros la hacemos en cualquier sentido y en un solo salto para facilitar la restauración de la difusión. La restauración ante la liberación de recursos tanto de los nodos como de los enlaces no es tomada en cuenta en ninguno de los trabajos relacionados. Y el único trabajo que no soporta la conexión y desconexión dinámica por su enfoque centralizado es *self-organising video.*

Otras contribuciones de la tesis fueron que se definió un mecanismo distribuido para la desconexión de los nodos que optimiza los recursos de la red, lo cual no se hace en otros trabajos. También, se creó un agente en NS2 que puede ser utilizado para estudios posteriores de la arquitectura propuesta y de otras arquitecturas de redes activas.

Trabajo	Retraso de conexión	Adaptación a los recursos de los enlaces	Adaptación a los recursos de los nodos	Restablecimiento Ante la liberación de los recursos	Escenario dinámico
Active Video[16]	Medio	NS	NS	NO	SI
Self-organising Video [17]	Alto, (Centralizado)	SI (Método centralizado).	SI (Distribución y optimización de la computación)	NO	NO
ARTES 1 [18]	Alto (cargado inicialmente) / bajo (pre-cargado en el nodo)	NS	SI (Distribución de la computación a priori)	NO	SI
MeGa [20]	Bajo	NS	NS	NO	SI
Duysburgh active framework [22]	Alto (cargado inicialmente) / bajo (pre-cargado en el nodo)	NS	SI (Difusión de la computación)	NO	SI
Arquitectura propuesta	Alto (cargado inicialmente) / bajo (pre-cargado en el nodo)	SI (Degradación de la tasa de bits)	SI (Difusión de la computación y degradación de la clase de video)	SI	SI

Tabla 6.1 Tabla de comparación del trabajo relacionado.

6.2 Trabajo a futuro.

En la arquitectura se asume que los usuarios conocen sus limitantes, los usuarios podrían darse cuenta de cuál clase de video se aproxima más a sus capacidades utilizando información acerca de la QoE del usuario y mejorar así el servicio. La información acerca de la QoE de los usuarios puede

utilizarse también para tomar decisiones como el dominio de transcodificación, el método de difusión por defecto (*up, down, proxy* ó combinado), para activar o desactivar los mecanismos de adaptación tanto a los nodos como a los enlaces, entre otros.

Ofrecer un tercer nivel de adaptación a los recursos de los nodos activos en cuanto al dominio de transcodificación se refiere, buscando obtener una probabilidad de obtener un flujo de video más cercana al 100%. El tercer nivel de adaptación podría utilizar un transcodificador a nivel paquete (*fully-compress domain*) que exige menos recursos de los nodos activos.

Estudiar y analizar la técnica de tolerancia a fallos de la red propuesta en el capítulo 3. Realizar un análisis más detallado sobre la estabilidad y desempeño del protocolo para poder aumentar su desempeño. Probar nuevas políticas y algoritmos para la selección del nodo al cuál se va a difundir la computación.

Bibliografía

[Amir95]E. Amir. S. McCanne, H. Zhang, "An Application Level Video Gateway", in *Proceedings of the ACM Multimedia 95*, San Francisco CA., 1995.

[Amir98] E. Amir, S. McCanne, R. Katz, "An Active Service Framework and its Application to Real-time Multimedia Transcoding", in *ACM SIGCOMM Computer Communication Review*, v.28 n.4, pp.178-189, Oct. 1998.

[Bai04] Y. Bai, M. Ito, "Qos control for video and audio communication in conventional and active networks: approaches and comparison", in *IEEE Communications*, Survey Tutorials vol. 6 No.1, pp 43-49, 2004.

[Banchs98] A. Banchs, W. Effelsberg, C. Tschudin, V. Turau, "Multicasting Multimedia Streams with Active Networks," in *23rd Annual IEEE International Conference on Local Computer Networks (LCN'98)*, pp. 150, 1998.

[Bhattacharjee96] S. Bhattacharjee, K. L.Calvert, and E. W.Zequra, "Implementation of an Active Networking Architecture", Technical Report, Georgia Institute of Technology, July 1996.

[Bush01] Stephen F. Bush and Amit B. Kulkarni, "Active Networks and Active Network Management", Kluwer Academic / Plenum Publishers, ISBN 0-306-46560-4, 2001.

[Calvert98] K. L. Calvert, S. Bhattacharjee, E. Zegura, and J. Sterbenz, "Directions in active networks" in *IEEE Commun. Mag.*, vol. 36, pp. 72–78, Oct. 1998.

[Coulouris01] G. Coulouris, "Sistemas Distribuidos, Conceptos y diseño", Addison-Wesley, 3ra edición, ISBN 84-7829-049-4, 2001.

[Decasper98] Dan Decasper and Bernhard Plattner, "DAN: Distributed Code Cashing for Active Networks", in *Proc. IEEE INFOCOM '98*, San Francisco, CA, 29 March–2 April 1998.

[Duysburgh00] B. Duysburgh, T. Lambrecht, B. Dhoedt, P. Demeester, "Data Transcoding in Multicast Sessions in active Networks", in *IWAN 2000*, LNCS 1942, pp. 130-144, 2000.

[Duysburgh04] B. Duysburgh, T. Lambrecht, F. De Turck, B. Dhoedt, P. Deemester, "An Active Networking Based Service for Media Transcoding in Multicast Sessions", in *IEEE Transactions on Systems, Man, and Cybernetics*, Part C vol. 34 No. 1, pp. 19-31, Feb 2004.

[Duysburgh07] B. Duysburgh, T. Lambrecht, F. De Turck, B. Dhoedt, P. Demeester, "Design and analysis of a stable set-up protocol for transcoding multicast trees in active networks", in *J. Netw. Comput. Appl.* Vol. 30, Iss. 4, pp. 1428-1444, Nov. 2007.

[Forouzan06] A. Forouzan, "Data communications and networking", McGraw-Hill, 4th edition, ISBN-13: 978-0073250328, Feb 2006.

[Galis04] A. Galis, S. Denazis, C. Brou, C. Klein, "Programmable Networks for IP Service Deployment", in *Artech House Publishers*, ISBN 1-58053-745-6, 2004.

[Ghanbari89] M. Ghanbari "Two-layer coding video signals for VBR networks", *IEE Journal on Selected areas in communications*, 7:5, pp. 771-781, June 1989.

[Ghanbari97] M. Ghanbari, C.J. Hughes, M.C. Sinclair and J.P Eade, "Principles of Performance engineering for telecommunication and information systems", IEE London UK, ISBN 0 85296 883 3, 1997.

[Ghanbari99] M. Ghanbari, "Video coding: an introduction to standard codecs", in *IEE Press*, ISBN 0 85296 762 4, Aug 1999.

[Gunter98] C. A. Gunter, S. M. Nettles, and J. M. Smith, "The SwitchWare Active Network Architecture", in *IEEE Network, special issue on Active and Programmable Networks*, May/June 1998, vol. 12, no. 3.

[Kang00] S. Kang, "The active traffic control Mechanism for Layered Multimedia Multicast in Active Networks", in *8th Int'l. Symp. Modeling Analysis and Simulation of comp. and telecommunications Sys.*, San Francisco, USA, Aug. 2000.

[Khan02] J. I. Khan, S. S. Yang, D. Patel, O. Komogortsev, Wansik Oh, Zhong Guo, Q. Gu, P. Mail, "Resource Adaptive Netcentric Systems on Active Network: A self-Organizing video stream that auto Morphs Itself while in Transit via Quasi-Active Network", *DARPA Active Networks Conference and Exposition (DANCE'02)*, pp. 427, 2002.

[Konstantinos99] P. Konstantinos, "Active Networks: Applications, Security, Safety, and Architectures" in IEEE Communications Surveys Tutorials, vol. 2, no. 1, 1999.

[Kouvelas98] I. Kouvelas, V. Hardman, J. Crowfort, "Network adaptive continuous media application through self organised transcoding", in *Proceedings of the Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV 98, July 1998.

[Lambrecht06] T. Lambrecht, B. Duysburgh, T. Wauters, F. De Turck, B. Dhoedt, P. Demeester, "Optimising multimedia transcoding multicast trees", in *Computer Networks*, Vol. 50, Issue 1, pp. 29-45, Jan 2006.

[Liu03] J. Liu, Bo Li, Ya-Qin Zhang, "Adaptive Video Multicast over the Internet", in *IEEE Multimedia,* vol. 10 pp. 22-33, Jan-Mar 2003.

[Lua04] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, S. Lim, "A survey and comparison of peer-to-peer overlay Network Schemes", in *IEEE Communications survey and tutorial*, pp 1-22, March 2004.

[McCanne96] S. McCanne, Van Jacobson, M. Vetterli, "Receiver-Driven Layered Multicast", *SIGCOMM,* Stanford, CA, pp. 117-130, Aug 1996.

[Mellado97] E. Mellado, "Introducción a las redes de Petri", Apuntes de curso. Facultad de Ciencias Físico-Matemáticas. UANL, Oct 1997.

[Najafi98] K. Najafi, A. Leon-Garcia, "Active Video: A novel approach to video distribution", in *2nd IFIP/IEEE International Conference Management of Multimedia Networks*, Versailles France, Nov 1998.

[Prasat03] R. Prasat, C. Dovrolis, "Bandwidth estimation: Metrics, measurement techniques, and tools", in *IEEE Network*, V. 17, Issue 6, pp. 27-35, Dec 2003.

[Ramirez08] J. Ramirez-Barba, M. Siller, "Video transcoding in active networks", in Worldcomp 2008, Las Vegas NV, Jul 2008.

[Ros04] Fco. J. Ros, P. Ruiz, "Implementing a new Manet unicast routing protocol in NS2", Free Software Foundation, Dec 2004.

[Sacks05] L. Sacks, H. K. Sellappan, S. Zachariadis, S. Bhatti, P. Kirstein, W. Fritsche, G. Gessler, K. Mayer, "On the manipulation of JPEG2000, in-flight, using active components on next generation satellites", in *IWAN2005 7th Annual International Working Conference on Active and Programmable Networks* , CICA, Sophia Antipolis, La Cote d'Azur, France, Nov 2005.

[Schwartz98] B. Schwartz, W. Zhou, and A. W. Jackson, "Smart Packets for Active Networks", BBN Technologies, Jan. 1998.

[Sentinelli07] A. Sentinelli, G. Marfia, M. Gerla, L. Kleinrock, "Will IPTV Ride the peer-to-peer stream?", in *IEEE communications Magazine*, pp 86-92, June 2007.

[Siller03] M. Siller, J. Woods, "A Quality of Experience Framework for Audio and Video Transmission", IEE WIAMIS 2003 Proceedings, pp. 288-293, 4th European Workshop on Image Analysis for Multimedia Interactive Services, London, UK, April 2003.

[Siller06] M. Siller, J. woods, "Using an agent based platform to map quality of service to experience in conventional and active networks", in *IEE Proceedings online no. 20060074*, Apr 2006.

[Smith99] J. M. Smith, K. L. Calvert, S. L. Murphy, H. K. Orman, and L. L. Peterson, "Activating networks: a progress report" in *IEEE Computer*, vol. 32, pp. 32–41, Apr. 1999.

[Tanenbaum03] A. S. Tanenbaum, "Redes de Computadoras", Cuarta Edición, Pearson-Printice-Hall, 2003.

[Tennenhouse97] D. Tennenhouse, "A survey of active network research" in *IEEE Communication Magazine*, vol. 35, pp. 80–86, Jan 1997.

[Vetro03] A. Vetro, C. Christopoulos, Huifang Sun, "Video transcoding architectures and techniques: an overview" in *IEEE Signal Processing Magazine*, Volume 20, Issue 2, pp. 18-29, Mar 2003.

[Walpole99] R. Walpole, R. Myers, "Probabilidad y estadística", 4ta edición, McGraw-Hill, ISBN 968-422-992-5.

[Wetherall96] D. Wetherall and D. Tennenhouse, "The ACTIVE_IP option", in *the 7th ACM SIGOPS European Workshop*, 1996.

[Wetherall98] D. Wetherall, J. Guttang, D. Tennenhouse, "ANTS: A toolkit for building dynamically deploying network protocols", in *IEEE OpenArch98*, pp. 117-129. Apr. 1998.

[Wu06] H.R. Wu, K.R. Rao, "Digital Video Image Quality and Perceptual Coding", Ed. Taylor and Francis Group, ISBN 0-8247-2777-0, 2006.

[wasu] Arizona State University, http://trace.eas.asu.edu/TRACE/trace.html.

[wmatlab] MatLab, http://www.mathworks.com/.

[wns2] Network Simulator 2, http://www.isi.edu/nsnam/ns/.

[wpipe] PIPE 2, http://pipe2.sourceforce.net.

[wvtsd] Video Transcoding and Streaming Demo, http://www.discover.uottawa.ca/~leizj/Experiences/TransServer/TransServer.html.

[Xiao99] X. Xiao, L. M. Ni, "Internet QoS: a big picture", in *IEEE Network*, vol. 13, pp. 8-18, Apr 1999.

[Yemini96] Y. Yemini and Sushil da Silva, "Towards Programmable Networks", in *Proc. IFIP/IEEE Int'l Workshop on Distributed Systems, Operations, and Management*, L'Aquila, Italy, 1996.

ANEXO I – Publicación.

Video Transmission and Transcoding in Active Networks

Joel Ramirez-Barba[1], **Mario Siller**[1]
[1]CINVESTAV, Zapopan, Jalisco, Mexico

Abstract *This paper presents a survey in video transmission and transcoding using active networks. The transcoding and active networks are used to enable node and network adaptability when the transmission is compromised by limited available resources. These approaches can be characterised by its transcoding code distribution method, set-up delay, network and node adaptation method, supported media format and transcoding technique. The video transmission solution performance will depend on the design and implementation overall issues and limitations of these characteristics.*

We present the main solutions and what we consider opened research questions in this area.

Key words: Active Networks, Video Transcoding, Survey.

1. Introduction

Pervasive computing and the internet are two key factors which have impacted the way networks are built and their diversity. This has evolved into what is referred to as heterogeneous networks. These networks are built from the interconnection of networks based on different technologies, protocols and resources. The heterogeneity applies also to the nodes themselves. They vary in size, mobility and processing power capacity. These include laptops, PCs, PDAs, sensors, mobile phones, etc. End users would normally like to receive information (voice, video or data) with the best possible quality. In current packet switch networks this is achieved by using Quality of Service protocols, as described in detail in [9,10]. In multimedia communications there are two approaches for media delivery: peer-to-peer approach (P2P) and Multicast. For the former media delivery is based on distributed architectures whilst for the latter in centralised ones. In peer-to-peer systems nodes could act as either server or client or even both [12]. The idea in P2P is to share resources among all peers. Whenever a peers requests a media file this is searched among the other peers and if found then it is transmitted to the requester. In P2P some limitations are observed. It is adequate for on-demand viewing of delay-tolerant content but not for time-critical video streams because of asymmetric residential user network access [13].

In multicast transmission the video is provided by a media server to the end users. The transmission is done in one of two ways: high quality stream and low quality stream. In the former low resources devices are not able to receive it whilst in the latter video quality is degraded for high resources devices. In order to solve this issue transcoding could be employed to reduce video quality only to low resources devices as needed [5]. Transcoding is detail explained in section 2.

The heterogeneity problem can be solved by different approaches. A classification of them is presented in [11]. This classification is based considering both video rate and transcoder location. In terms of video rate transcoding is done either in *single rate* or *multirate*. The stream adaptation could be done by locating the transcoder either at the end system (*end-to-end service*) or at intermediate network nodes (*active service*). One solution is single-rate adaptation (*single rate, end-to-end service*) in which the sender maintains one video stream which is adjusted according to receiver's available resources. An alternative way of transmission is to send multiple video formats each forming a network stream to groups of users with similar capacities. These are referred to as multicast trees. One proposal on this is called " Simulcasting (*multirate, end-to-end service*) and is presented in [6]. A similar technique consists in using layered media streams (*multirate, end-to-end service*) in which the users subscribe to a number of layers as necessary in order to get the required quality [7]. Here the source or video server generates a multicast tree by each video layer. The network intermediate node can also play a role during the transmission. One proposal based on this approach is the Media Application Gateway (*multirate, active service*) presented in [8]. In this solution the transcoding is performed by special gateways located within the network. This approach can be extended by using active networks [1, 2, 3, 4]. In the active networks solution the transcoding intermediate network nodes are provided with programming capabilities and are able to process user data and self configuration state (*multirate, active service*). Another active approach is layered video in active networks. In this approach the video is encoded into multiple layers: base layer and enhancement layers.

The former provides essential information whilst the latter complementary data in order to produce different video quality levels. This idea was originally proposed by Ghambari in [24]. In this scheme the video is transmitted by the source and each client receives a number of layers depending on both network and end device capacity and the desired quality level. In [14] Kang et al proposed this scheme for an active networks. This solution is called *active traffic control mechanism for layered multimedia multicast* (ATLM). The layered video is a fast adaptation technique since not intermediate video processing is required. However, two main disadvantages are identified. Video formats compatible with layer coding can be used only. This limits the number of available formats for the end users. Another drawback is the granularity. The video resolution depends only on the number of encoded layers whilst the bit rate reduction on the number of them the users can receive. (*multirate, active service*).

In [1] an active network (AN) is defined as "*a network that allows routers to perform computations up to the application layer*". In these networks routers states can dynamically be changed by local code or embedded programs in the transmitted packets. The AN's architectures are classified in: (i) active packets approach, (ii) active nodes approach and (iii) active packets and nodes approach [1]. In these networks the active nodes execute the local transcoding code or the received code contained in the active packet.

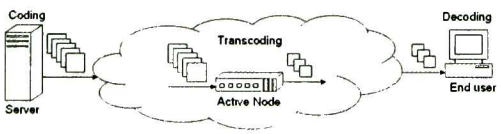


Figure 1: Transcoding in Active Networks.

In the context of multimedia transmission, Quality of experience (QoE) can be an interesting issue. QoE in audio and video transmissions is proposed in [26].

This paper is organised as follows. Section 2 makes an overview of transcoding/transrating techniques. Section 3 presents a survey of video transcoding approaches in active networks. In section 4 a comparison of these approaches is made. Section 5 concludes the paper and presents opened research directions.

2. Transcoding / Transrating techniques

The transcoding is a useful technique to downscale the transmission video requirements for adaptation purposes to both network and receiver capacity. The downscaling is achieved by removing redundant or low priority video information. This is based on a trade-off between video quality loss, bit rate reduction and computation complexity [5] [17].

When transcoding is made only to reduce the bit rate without changing the video format is called transrating. Both terms transcoding and transrating are used alike.

A general transcoding architecture in active networks is shown in figure 1. The server encodes the video in a given format and transmits it to the network. In the case of the active network approach the active nodes transcode the video according to the new requirements. Then the video is transmitted to the receiver. At the end user side the adapted video is received and decoded to be displayed.

The video transcoding can be made in tree domains: i) pixel-domain, ii) DCT-domain and iii) fully compressed-domain (frame level). This is shown in Figure 2.

The transcoding techniques can be selected based on both the Quality of Service (QoS) metrics and end user Quality of Experience (QoE). For instance, when the video resolution reduction is a concern and processing latency is not then the pixel-domain approach could be employed, see Table 1.

Domain	*Quality loss*	*Computation complexity*
Pixel	Low	High
DCT	Medium	Medium
Fully compress	High	Low

Table 1. Comparison between quality loss and computation complexity in the different transcoding domains.

2.1. Pixel-domain

Transcoding in this domain involves complete decoding and encoding of the video. This is because the transcoding operations are made at the pixel level. These include pixel downsampling, chrominance reduction, luminance reduction and frame resizing. This domain requires high computation complexity because of the coding and decoding stages and high memory to temporary storage of the video. However, quality loss is less since what is affected in the video can be known in a precise way. The operations at this domain represented as T1 in Figure 2.

2.2. DCT-domain

The transcoding in this domain is made by manipulating the DTC coefficients. This implies a partial decoding of the video up to the point in which these coefficients are obtained from the bit stream. This is shown as T2 in Figure 2. The use of the coefficients

simplifies the complexity or time consumed operations which could be performed in other domains. Besides the memory requirements to store the already reduced coefficients are much less then the macroblocks pixel information. The operations include colour to monochrome conversion, frequency filtering, data partitioning, quantisation scale adjustment and simple codec conversion.

2.3. Fully compressed-domain

The transcoding in this domain is made directly on I-, P-, and B-frames, see T3 in Figure 2. This is, in other words, at a frame level. In this approach frames are intelligently discarded to adapt to the network or end user conditions. This implies a high quality loss reduction depending on the frame dropping policy. The computational complexity is lower than in any other domain.

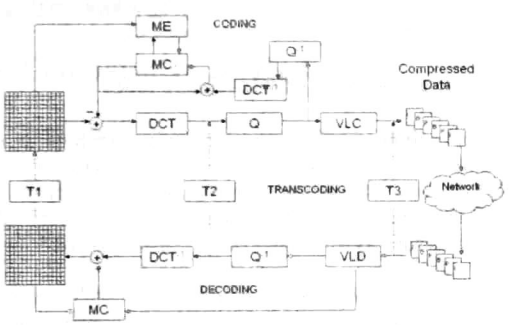


Figure 2. Generic Discrete Cosine Transform-based transcoding architecture.

3. Video Transcoding in Active Networks

In this approach the video is sent by the source in the original format or in the format of the user with higher resources. Then the active nodes transcode the video in order to adapt the original format to a new one requested by a given end user. In this section a detailed survey of the main solutions is presented.

Duysburgh et al [15] identifies two issues to be considered for transcoding data in a multicast session in active networks. The first issue is concern with the application of active networking techniques to achieve transcoding in the active nodes. The second issue is related to the selection of an optimal location of the active nodes in the multicast trees. These two considerations are described in detail in the following sections.

3.1. Application of Active techniques

The transcoding solutions in active networks can be characterised by both the active network implementation technique and the transcoding approach. Two active techniques are identified:

(i) **Code distribution**: it is concern with how the transcoding code is loaded in the active node. This can be done in three different ways:
   a. *Active packets.* It uses capsules (active packets) to carry the data and code to be loaded in the active node
   b. *Active nodes.* The code is preloaded in the active node either manually or using a code-base (code server).
   c. *Active packets and nodes.* This is a combination of the other two. This is part of the required information at the active node is preloaded whist the other transported by the active packets.

(ii) **Node resources adaptation**: it refers to how the node resources are distributed to the different video flows. Two approaches are identified:
   a. *Computation distribution or diffusion.* When a node lacks sufficient processing power for the transcoding either some components or the complete transcoder are migrated to another node.
   b. *Computation optimisation.* In this approach some video quality is lost in order to reduce computational complexity.

In terms of the transcoding method and as previously discussed there are tree domains:

(i) Pixel.
(ii) DCT
(iii) Fully compress.

The state of the art in transcoding using active networks is presented below.

Active Video. Najafi et al [16] propose a new technique which applies to exiting major video distribution schemes (real time transport protocol, multi-layer multicast and application level video gateway). The video operations and the video itself are transported together into active packets referred to as "Active Video" objects. When a video stream arrives to the active node an agent is started. This agent then captures the new environment. This is done by obtaining information about the network and node available resources and whether other agents are already running at the node. If there are already running agents the new one contacts the others to negotiate the *downscaling strategy*. However, no details about how this is done are presented in [10].

SONET (Self-Organising Network Embedded Transcoder). Khan et al in [17] discusses the issues related to adaptive networking in both current infrastructure and active networks. These include location of adaptation, bandwidth and computational power

asymmetry, network embedded computing, dynamic adaptation, self-organisation and seamlessness. Khan et al. proposes a video transcoding system for adaptive networking. The video stream is adapted according the network condition by the transcoding systems as it transits the active network. The adaptation is done considering the *bandwidth* or *bit rate* and *node capacity*. In terms of the former two grades of content-awareness are considered: (i) awareness about the content protocol (video coding standard) and (ii) awareness about the content itself. For the node computing power adaptation it considers: (i) modular self-organisation or computation optimisation and (ii) computation distribution or diffusion. The transcoder deployment is done in two phases: (i) determining the network *constriction points (bottlenecks)* using a topological map and (ii) a system component connection map is built. This map describes how the components have to be topologically connected and located. The user interaction, system initiation and dismantling and setting state configurations are managed by a centralised Meta-Controller (MC).

ARTES 1. In [18] an active node architecture for next generation satellites is presented by Sacks et al.
The architecture consists of several elements. The media provider generates the source video and transmits it in any format. The active nodes under this architecture are link ingress, satellite and link egress node. An additional non-active element is a server called the active component provider. This server hosts the active components and deploys them to the active nodes as required. The architecture provides two active services: transcoding and intelligent dropping. In the case of the transcoding service when a media provider sends a stream of images these are converted by ingress node to JPEG2000 format. This is then forwarded to the satellite. The satellite performs any additional transcoding if required and forwards the stream to the egress node. This finally converts the JPEG200 images to the original format. The resulted stream is finally delivered to end users.

MeGa (Media Gateway). Amir et al [20] presents an active framework that allows the deployment of active services for real time multimedia transcoding. In this framework, the active services (AS) are provided by the active nodes. A pool of these nodes a form clusters. Users that require an active service instantiate an application-level agent called servent (service agent) in one or clusters. If the service is complex this may require several servents and clusters. The servents behave according to the specific active service program and can be controlled by the users. Based on this framework a media gateway service is proposed. The MeGa are servents in charge of bridging two MBone RTP sessions and performing the transcoding as required.

Duysburgh Active Framework. Duysburgh et al [22] propose an active networking framework in which the active nodes are implemented based on the Active Node Transport System (ANTS) [25]. ANTS runs on a java environment. It can execute java byte-code carried in the capsules (active packets). In ANTS the code can be extended with loadable extensions. If the extension is not in the soft-state cache of the active node then it downloads code from a well known and secured code-base (server). The multicast tree set-up is based on a distributed version of the branch attach heuristic algorithm (see section 3.3.2). The transcoders location is determined during the tree set-up. If a node doesn't have enough processing power for transcoding it relocates the transcoder another node which does. Three mechanisms are used for transcoder relocation:

(a) *Push upstream*, the transcoder is moved up in the multicast tree.
(b) *Push Downstream*, the transcoder is moved down in the multicast tree.
(c) *Proxy*, the transcoder is relocated to a neighbour proxy which can be out of the multicast tree.

3.2. Optimisation of Multicast trees

The optimisation of a multicast tree depends in its architecture and choosing a location for transcoding in its active nodes. The design decisions are made in terms of achieving the best possible use of network and node resources such as bandwidth and processing power.

Building a transcoding multicast tree (TMT) is an NP complete problem. Lambrecht et al. in [21] propose three algorithms to find the sub-optimal transcoding locations. These are:

1. **Skinned Steiner Tree Heuristic (SST).** The algorithm is divided in two phases. In the first phase a tree with the sufficient bandwidth to carry the requested format is estimated. In the second phase the tree is skinned in order to estimate the available bandwidth and transcoding requirements

2. **Branch attach Heuristic (BA).** In this algorithm the minimum cost tree towards all clients with the high resolution format is estimated. Then the clients with less resolutions area attached to the previous branches. The latter is iterated for each required video resolution (class).

3. **Distributed BA.** This is a distributed version of the previous BA algorithm. When a new user attempts to join an existing multicast session a request is sent towards the source. If during the request transmission path a node belonging to the tree is found three situations are considered:
   i) If the requested format type is the same as the found node then is extended to the requester.
   ii) If the requested format resolution is lower than in the found node then a transcoder is enabled at

the latter.

iii) If the requested format resolution is higher than the request is forwarded to the next node. If the source node is reached then the stream is transmitted directly from the source to the requester.

If no intermediate node is found in the source path then the stream is transmitted directly from the source to the requester. In this algorithm nodes are allowed to dynamically join and leave to the multicast tree using JOIN and LEAVE capsules [22] [23].

A stability and consistency study of the distributed BA algorithm is presented by Duysburgh et al. in [23]. They identified some changes in the active node tables that may cause information loss. They propose a solution consisting using a field (tc) from the table and temporary leaving an entry. This is done until the J-cln packet is received. At this time the requested video resolution would have been already received.

4. Comparative Analysis

There are different ways to transmit video in active networks as previously described. These approaches can be characterised by its transcoding code distribution method, set-up delay, network and node adaptation, supported media format and transcoding technique (pixel, DCT or fully compressed domain). The set-up delay refers to the time to locate and install the transcoding code to start the transmission. This is defined as *low* if the code is preinstalled at the nodes or previously loaded from a code base, *medium* when the code travels in the active packets and *high* when it is transmitted initially from a code-base or centrally managed (meta-controller). The network and node adaptation refers to capacity to adjust to the available resources.

A comparative analysis from the discussed video transmission approaches in active networks is shown in Table 2. In the approaches the code is managed using *active packets* or *active nodes*. For the former security, packet size and fragmentation are an issue. In the latter the supported media formats is limited by the available codes at the either at the node or code base. In the case of the media application gateway this code distribution is performed manually by the network administrator. This may cause service deployment flexibility issues.

The network congestion adaptation is performed by reducing the bit rate considering either the coding standard or the video content itself. The adaptation is triggered by considering network metrics thresholds such as packet loss, available bandwidth (traffic load) and traffic peaks. The way these metrics are estimated can affect the adaptation performance.

The node adaptation can be based on computation distribution, optimisation or both. The adaptability capacity is extended when both schemes are employed. In terms of the transcoding techniques it is observed that any of the discussed techniques is employed. In other words, none of the techniques are better proffered from the others.

The video transmission solution performance will depend on the design and implementation overall issues and limitations previously discussed. This is a matter of continuous research today.

5. Conclusions

The video transcoding can be made in the pixel, DCT and fully compressed domains (frame level). Each domain implies a trade off between video quality loss, bit rate reduction and computation complexity. The transcoding technique can be selected considering user requirements and both end devices and network available resources.

The transmission in active networks can be done using a centralised media application gateway, layered video and transcoding distributed among the active nodes. In the case of the media application gateway the centralisation may cause congestion in its surrounding links. A limitation also arises if a user requires a service not being already provisioned by the network administrator. The main issue when using the layered coding is granularity and that only video format compatible can be employed. For transcoding in the active networks two issues must be considered: the active networking techniques and the location of the active nodes. The solutions on this area can be characterised by its transcoding code distribution method, set-up delay, network and node adaptation, supported media format, transcoding technique and whether it is distributed or centrally implemented.

The video transmission solution performance using active networks will depend on the design and implementation overall issues and limitations of this characteristics. We identified several opened research questions. What characteristic would be more relevant to a given design? How the network and node dynamics can be linked to the transcoding adaptability? Can quality of experience play a roll in the optimisation design decision? An intermediate solution between centralised vs. distributed can offer a better performance? What fault management considerations need to be addressed? We are currently working on these research lines and expect to present results in further publications.

Approach	Code distribution method	Set-up delay	Network resources adaptation	Node resources adaptation	Media format	Transcoding Domain
Active Video[16]	Active packets	Medium	NS	NS	MPEG4	Fully Compressed
Self-organising Video [17]	Active packets	High, (Centralised)	YES (triggered by coding standard and video content awareness).	YES (computation optimisation and distribution)	MPEG-2	Pixel
ARTES 1 [18]	Active nodes (Code Base)	High (initially loaded) / Low (previously loaded or preinstalled)	NS	YES (computation distribution)	JPEG2000	DWT (equivalent to DCT)
MeGa [20]	Active nodes (NS)	Low	NS	NS	NS	NS
Duysburgh active framework [22]	Active nodes (ANTS-Code Based)	High (initially loaded) / Low (previously loaded or preinstalled)	NS	YES (computation distribution)	PCM, G.711, G.726, G.728, G.729	Voice (Fully decoding required – equivalent to Pixel)

Table 2: Transcoding works in Active Networks. NS = Not Specified

6. References

[1] P. Konstantinos, "Active Networks: Applications, Security, Safety, and Architectures" in *IEEE Communications Surveys Tutorials*, vol. 2, no. 1, 1999.

[2] D. Tennenhouse, "A survey of active network research" in *IEEE Communication Magazine*, vol. 35, pp. 80–86, Jan 1997.

[3] J. M. Smith, K. L. Calvert, S. L. Murphy, H. K. Orman, and L. L. Peterson, "Activating networks: a progress report" in *IEEE Computer*, vol. 32, pp. 32–41, Apr. 1999.

[4] K. L. Calvert, S. Bhattacharjee, E. Zegura, and J. Sterbenz, "Directions in active networks" in *IEEE Commun. Mag.*, vol. 36, pp. 72–78, Oct. 1998.

[5] A. Vetro, C. Christopoulos, Huifang Sun, "Video transcoding architectures and techniques: an overview" in *IEEE Signal Processing Magazine*, Volume 20, Issue 2, pp. 18-29, Mar 2003.

[6] I. Kouvelas, V. Hardman, J. Crowfort, "Network adaptive continuous media application through self organised transcoding", in *Proceedings of the Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV 98, July 1998.

[7] S. McCanne, Van Jacobson, M. Vetterli, "Receiver-Driven Layered Multicast", *SIGCOMM*, Stanford, CA, pp. 117-130, Aug 1996.

[8] E. Amir. S. McCanne, H. Zhang, "An Application Level Video Gateway", in *Proceedings of the ACM Multimedia 95*, San Francisco CA., 1995.

[9] X. Xiao, L. M. Ni, "Internet QoS: a big picture", in *IEEE Network*, vol. 13, pp. 8-18, Apr 1999.

[10] Y. Bai, M. Ito, "Qos control for video and audio communication in conventional and active networks: approaches and comparison", in *IEEE Communications*, Survey Tutorials vol. 6 No.1, pp 43-49, 2004.

[11] J. Liu, Bo Li, Ya-Qin Zhang, "Adaptive Video Multicast over the Internet", in *IEEE Multimedia*, vol. 10 pp. 22-33, Jan-Mar 2003.

[12] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, S. Lim, "A survey and comparison of peer-to-peer overlay Network Schemes", in *IEEE Communications survey and tutorial*, pp 1-22, March 2004.

[13] A. Sentinelli, G. Marfia, M. Gerla, L. Kleinrock, "Will IPTV Ride the peer-to-peer stream?", in *IEEE communications Magazine*, pp 86-92, June 2007.

[14] S. Kang, "The active traffic control Mechanism for Layered Multimedia Multicast in Active Networks", in *8th Int'l. Symp. Modeling, Analysis and Simulation of comp. and telecommunications Sys.*, San Francisco, USA, Aug. 2000.

[15] B. Duysburgh, T. Lambrecht, B. Dhoedt, P. Demeester, "Data Transcoding in Multicast Sessions in active Networks", in *IWAN 2000*, LNCS 1942, pp. 130-144, 2000.

[16] K. Najafi, A. Leon-Garcia, "Active Video: A novel approach to video distribution", in *2nd IFIP/IEEE International Conference Management of Multimedia Networks*, Versailles France, Nov 1998.

[17] J. I. Khan, S. S. Yang, D. Patel, O. Komogortsev, Wansik Oh, Zhong Guo, Q. Gu, P. Mail, "Resource Adaptive Netcentric Systems on Active Network: A self-Organizing video stream that auto Morphs Itself while in Transit via Quasi-Active Network", *DARPA Active Networks Conference and Exposition (DANCE'02)*, pp. 427, 2002.

[18] L. Sacks, H. K. Sellappan, S. Zachariadis, S. Bhatti, P. Kirstein, W. Fritsche, G. Gessler, K. Mayer, "On the manipulation of JPEG2000, in-

flight, using active components on next generation satellites", in *IWAN2005 7th Annual International Working Conference on Active and Programmable Networks* CICA, Sophia Antipolis, La Cote d'Azur, France, Nov 2005.

[19] A. Banchs, W. Effelsberg, C. Tschudin, V. Turau, "Multicasting Multimedia Streams with Active Networks," in *23rd Annual IEEE International Conference on Local Computer Networks (LCN'98)*, pp. 150, 1998.

[20] E. Amir, S. McCanne, R. Katz, "An Active Service Framework and its Application to Real-time Multimedia Transcoding", in *ACM SIGCOMM Computer Communication Review*, v.28 n.4, pp.178-189, Oct. 1998.

[21] T. Lambrecht, B. Duysburgh, T. Wauters, F. De Turck, B. Dhoedt, P. Demeester, "Optimising multimedia transcoding multicast trees", in *Computer Networks*, Vol. 50, Issue 1, pp. 29-45, Jan 2006.

[22] B. Duysburgh, T. Lambrecht, F. De Turck, B. Dhoedt, P. Deemester, "An Active Networking Based Service for Media Transcoding in Multicast Sessions", in *IEEE Transactions on Systems, Man, and Cybernetics*, Part C vol. 34 No. 1, pp. 19-31, Feb 2004.

[23] B. Duysburgh, T. Lambrecht, F. De Turck, B. Dhoedt, P. Demeester, "Design and analysis of a stable set-up protocol for transcoding multicast trees in active networks". *J. Netw. Comput. Appl.* Vol. 30, Iss. 4, pp. 1428-1444, Nov. 2007.

[24] M. Ghambari "Two-layer coding video signals for VBR networks", *IEEE. Journal on Selected areas in communications*, 7:5, pp. 771-781, June 1989.

[25] D. Wetherall, J. Guttang, D. Tennenhouse, "ANTS: A toolkit for building dynamically deploying network protocols", in *IEEE OpenArch98*, pp. 117-129. Apr. 1998.

[26] M Siller and J Woods, "A Quality of Experience Framework for Audio and Video Transmission", IEE WIAMIS 2003 Proceedings, pp. 288-293, 4th European Workshop on Image Analysis for Multimedia Interactive Services, London, UK, April 2003.

CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS DEL I.P.N.
UNIDAD GUADALAJARA

El Jurado designado por la Unidad Guadalajara del Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional aprobó la tesis

Trans-codificación y Transmisión de Video en Redes Convencionales y Activas

del (la) C.

Joel Isaac RAMÍREZ BARBA

el día 15 de Diciembre de 2008.

Dr. Luis Ernesto López Mellado
Investigador CINVESTAV 3B
CINVESTAV Unidad Guadalajara

Dr. Félix Francisco Ramos Corchado
Investigador CINVESTAV 3A
CINVESTAV Unidad Guadalajara

Dr. Mario Angel Siller González Pico
Investigador CINVESTAV 2A
CINVESTAV Unidad Guadalajara

Dra. Alejandra Flores Mosri
Consultoria de seguridad en sistemas y redes
Red Dynamica