Distributed Management of Application Layer Multicast Trees for IPTV Services

D. Díez-Hernández*, J. García-Reinoso*, A. García-Martínez*, A. Bikfalvi† and I. Vidal*
*Departamento de Ingeniería Telemática,
Universidad Carlos III de Madrid
Avda. de la Universidad 30, 28911. Leganés, Madrid.
Email: {ddhernan, jgr, alberto, ividal}@it.uc3m.es
†IMDEA Networks,
Avda. del Mar Mediterraneo 22, 28918, Leganés - Madrid
Email: alex.bikfalvi@imdea.org

***Resumen*—IP multicast is an efficient mechanism to distribute IPTV content towards multiple users, but nowadays Internet Providers filter out this kind of traffic, mainly due to security and accounting issues. In order to overcome this filtering, the same idea of IP multicast can be implemented at the end user terminals, but implementing multicast trees at the application layer, also known as Application Layer Multicast (ALM). ALM has some drawbacks, mainly due to dynamic behavior of users, joining and leaving the trees. In order to minimize the impact of this behavior, this paper proposes a distributed management for ALM in IPTV, instead of a centralized one, where some nodes connected to a tree will be on charge of the management of that tree. Furthermore, all nodes are dynamically configured with the substitute of its parent node in order to accelerate the reconnection process. Results presented in this article show that this proposal provides fast reconstructions and low management load per node, while keeping a balanced tree topology.**

***Palabras Clave*—ALM, video streaming, churn**

I. Introducción

Hoy en día, para transmitir vídeo en tiempo real utilizando el protocolo IP, el método más eficaz es utilizar *IP multicast* debido a que la información sólo se replica en aquellos puntos (routers) donde haga falta, al contrario que en *unicast*, donde la propia fuente debe replicar el contenido tantas veces como receptores tenga. Sin embargo, el uso de IP multicast en entornos multi-proveedor está restringido debido a la dificultad de tarificar este tipo de tráfico, ya que un paquete en origen puede replicarse varias veces en la red, antes de llegar a sus posibles destinos. Esto, unido a problemas de seguridad y escalabilidad, han hecho que IP multicast no sea utilizado masivamente en Internet para transmitir vídeo, por ejemplo. Sin embargo, se puede utilizar una técnica en donde se crean árboles de distribución a nivel de aplicación en vez de hacerlo a nivel de red como lo hace IP multicast. A este tipo de técnicas se les llama *Multicast a Nivel de Aplicación* o *Application Layer Multicast (ALM)* en inglés. En los árboles ALM, los diferentes receptores del contenido pueden actuar a su vez como emisores para otros equipos si disponen de los recursos necesarios, creando un árbol de distribución, cuya raíz será el equipo generador de los contenidos (un Media Server, por ejemplo). El árbol tendrá también nodos *hoja* que recibirán el contenido pero no transmitirán dicho contenido a otros nodos y, quizás, nodos *interiores* que recibirán el contenido de otros nodos y, a su vez, transmitirán el contenido a uno o varios nodos receptores. Se dice que si un nodo U_i envía datos a otro U_j, el primero es *padre* del segundo (y, lógicamente, el segundo será *hijo* del primero).

El principal problema al usar cualquier técnica de ALM residen en el comportamiento dinámico de los usuarios. Si un nodo padre de uno o varios nodos, decide cambiar de canal (o salir del sistema) el sistema deberá reconstruir el árbol recolocando a los hijos que deja *huérfanos*. Esto puede provocar un corte en la reproducción del vídeo en los hijos que han quedado huérfanos, dependiendo del tiempo que se necesite para realizar la reconstrucción del árbol. Por lo tanto, el tiempo de reconstrucción es un parámetro que se debe intentar minimizar. Además, es importante distinguir entre las salidas ordenadas y desordenadas de los usuarios. En el primer caso el usuario que abandona el canal enviará un mensaje a las partes implicadas para notificar su salida, y en el segundo, el abandono se producirá de forma abrupta por algún problema en su funcionamiento. Evidentemente, el comportamiento y el tiempo de reacción del sistema ante una u otra situación será diferente.

El tiempo de reconstrucción depende de varios factores y de la técnica utilizada para realizar la reconstrucción. Normalmente, en las salidas ordenadas, los nodos envían un mensaje a un nodo central que se encarga a su vez de reconstruir el árbol si fuese necesario. El nodo central conoce la estructura completa del árbol y los recursos disponibles de cada uno de los nodos, por lo que puede decidir cómo recolocar a los nodos huérfanos, balanceando la estructura del árbol resultante. Una vez aplicado el algoritmo de reconstrucción, el nodo central envía la información a cada uno de los nodos huérfanos para asignarles nuevos padres. Uno de los mayores inconvenientes de este mecanismo es que un nodo central debe tener la información completa de todos los árboles de distribución para que el algoritmo proporcione un resultado óptimo en cuanto a la distribución de carga de los nodos. Además, el tiempo de reconstrucción del árbol depende del tiempo de ejecución de dicho algoritmo y del tiempo de transmisión y propagación hasta recibir la información con el nuevo padre al que se deben conectar.

En este artículo proponemos un sistema distribuido de administración de los árboles multicast a nivel de aplicación para distribuir contenido IPTV, en donde la información de la estructura del árbol se encuentre distribuida entre los

propios nodos que forman el árbol. Con esta información, algunos nodos del árbol, designados a tal fin, se encargaran de calcular y distribuir la información de los *padres adoptivos* para cada uno de los nodos. Es decir, cada nodo en el árbol estará recibiendo el vídeo de un padre y a su vez tendrá un padre adoptivo, es decir, un padre al que conectarse tan pronto se detecte un problema con su actual padre. Con esto pretendemos minimizar el tiempo de reconstrucción de los árboles y distribuir la carga de administración entre los propios nodos que forman el árbol.

El resto del artículo está organizado de la siguiente manera. La Sección II presenta algunos ejemplos de árboles multicast a nivel de aplicación. La Sección III explica detalladamente la propuesta de administración distribuida de gestión de los árboles multicast a nivel de aplicación, mientras que la sección IV presenta resultados obtenidos mediante simulación, que validan la solución propuesta. La Sección V presenta las conclusiones.

II. Árboles Multicast a Nivel de Aplicación

Para transmitir vídeo a través de la red tradicionalmente se ha usado una arquitectura cliente-servidor, en la que cada usuario se conecta a un servidor que es la fuente del vídeo. Una modificación sobre esta arquitectura es la denominada CDN (*Content Delivery Network*) en la que la fuente de vídeo se distribuye por un conjunto de servidores situados estratégicamente en la red. De esta manera, el cliente se conecta al servidor de contenido más cercano que tenga, distribuyendo la carga del servidor central. El problema de las arquitecturas cliente-servidor para la distribución de vídeo es la escalabilidad, ya que conforme el número de usuarios aumenta, se necesitan más servidores para balancear la carga. Pero desde hace relativamente poco tiempo se han empezado a utilizar redes P2P para solucionar los problemas que presentan las arquitecturas tradicionales para la distribución de vídeo. En este tipo de redes, los usuarios actúan como clientes y como servidores.

Existen dos tipos de servicios de vídeo para los que se pueden utilizar estas redes. El primero es el de vídeo bajo demanda, en el que el usuario elige reproducir un contenido para visualizarlo cuando quiera. Un ejemplo de este tipo de servicios sería *youtube*[1]. El segundo, que es el que nos ocupa en este documento, es el de vídeo en directo. En este tipo de servicio, todos los usuarios desean recibir un vídeo de una fuente determinada que está retransmitiendo en directo. Es decir, los usuarios están sincronizados en la reproducción.

Para la opción de reproducir vídeo en directo, existen actualmente dos soluciones basadas en arquitecturas P2P: las estructuras basadas en árbol (*tree-based*) y las basadas en malla (*mesh-based*).

Las distribuciones basadas en árbol pretenden emular los árboles multicast ideados para la distribución de vídeo y audio. Pero dado que el soporte de IP multicast se encuentra muy restringido por parte de los proveedores, se ha intentado emular el comportamiento de éstas mediante la utilización de los ALM (Multicast a Nivel del Aplicación).

Podemos diferenciar dos tipos de estructuras basadas en árbol. Las redes de un sólo árbol y las redes de múltiples árboles.

En las redes de un sólo árbol existe una fuente de vídeo que envía al resto de usuarios ([1], [2]). Pero a diferencia de la arquitectura cliente-servidor, los usuarios estarán organizados en forma de árbol de forma que cada usuario recibe el vídeo de un único nodo padre y es capaz de reenviarlo a uno o más nodos hijo, dependiendo de los recursos que tenga disponibles. En este tipo de arquitectura se reduce la carga que tenía el servidor, ya que el resto de usuarios que forman la red también actúan como retransmisores del vídeo. Existen principalmente tres aspectos que hay que tener en cuenta a la hora de diseñar un sistema de este tipo. El primero es que cada nivel añade un retardo a la retransmisión del vídeo. Por lo tanto un criterio de diseño es mantener el número de niveles de un árbol bajo un cierto límite que dependerá del número de nodos. Para ello cada usuario debe intentar atender al mayor número de hijos posible. Pero esto en la realidad está claramente limitado por el ancho de banda de subida que tiene cada usuario, que hace que no se pueda retransmitir nada más que a un número limitado de hijos, que en general es muy pequeño. Segundo, tampoco es recomendable conectar muchos nodos hijos a un nodo con gran número de recursos, ya que en caso de abandono del padre, se tendrá que reconectar un elevado número de nodos, lo que podría suponer una reestructuración compleja del árbol. El último aspecto importante se refiere al mantenimiento del árbol. Los usuarios pueden entrar y salir del árbol en cualquier momento, y cuando lo abandonan, sus hijos dejan de recibir vídeo. Para que no se produzcan cortes en la reproducción, el árbol tiene que reconstruirse lo antes posible. En las arquitecturas existentes hasta ahora la construcción y mantenimiento del árbol se realizan de forma centralizada. Es decir, existe un servidor central que controla la posición en la que ha de ubicarse un nuevo usuario y también controla la reconfiguración del árbol en caso de detectar abandonos, ya sea de manera ordenada, por avisos que se produzcan, o desordenada, mediante un soft-state. En cualquier caso, para árboles relativamente grandes, este tipo de sistemas presentan claramente un cuello de botella que puede repercutir en el rendimiento general del árbol, ya que tendría que atender a demasiadas peticiones en función del dinamismo de la red que administra. Con lo cuál, es posible que el árbol no se pueda recuperar suficientemente rápido cuando haya abandonos.

En las redes de múltiples árboles ([3], [4], [5]) se intenta aprovechar los recursos de los nodos que están en el extremo final de cada rama del árbol, ya que son nodos que sólo consumen recursos, pero no se utilizan para enviar vídeo a nadie más. Una red basada en múltiples árboles pretende solucionar este problema creando varios flujos diferentes del mismo vídeo y haciendo que cada uno se transmita por un árbol diferente. De esta manera, cada nodo puede tener un rol diferente en cada uno de los subárboles, pudiendo ser nodos hoja en un árbol e interiores en otros, cediendo así por lo menos parte de su ancho de banda de subida. En este tipo de sistemas, la utilización de varios árboles mejora notablemente el aprovechamiento del ancho de banda de salida de los nodos, pero la manera de buscar nodos con los que conectarse o reconectarse a la red es bastante compleja y requiere tiempos que en determinados entornos pueden ser demasiado largos.

[1] http://www.youtube.com

Aún así, eso puede no ser un problema, ya que al estar conectados a varios árboles, los árboles tienen cierta garantía de seguir recibiendo información en caso de abandonos en alguno de los árboles.

Por último, los sistemas basados en malla son arquitecturas en las que los usuarios intercambian la información del vídeo (o cualquier otro contenido) entre ellos, formando topologías lógicas independientes de la red en la que estén, y además con la característica de que cada nodo puede tener más de una fuente de la que recibir el vídeo, según sus necesidades, y enviar a diferentes nodos, dependiendo de sus recursos. Un ejemplo de un sistema de este tipo sería CoolStream ([6],[7],[8]). Estos sistemas introducen el concepto de *buffer map*, que sirve para representar la disponibilidad de los últimos bloques de diferentes sub-flujos que se quieran reproducir. Esta información se debe intercambiar entre los nodos periódicamente para determinar a qué sub-flujos suscribirse. Por esa razón, la gestión de miembros es muy importante para la construcción y mantenimiento de la red. En general, los sistemas basados en malla son arquitecturas bastante robustas en cuanto a la posibilidad de estar recibiendo vídeo, ya que cada nodo se asegura siempre de tener a un conjunto suficiente de usuarios que le estén proporcionando sus *buffer maps* y así siempre tener datos que mostrar. Pero por otro lado, son estructuras relativamente complejas que requieren un tratamiento complejo de los bufferes de recepción, y que además no deja de estar sujeta a discontinuidades en la reproducción, debido a la selección aleatoria de compañeros con los que asociarse, que puede crear incertidumbre sobre los datos a recibir.

III. Descripción de la propuesta

A continuación describiremos la solución planteada para distribuir la carga de construcción y reconstrucción de los árboles multicast que se utilizarán para la transmisión de vídeo IPTV. Se busca una arquitectura distribuida, descentralizada y uniforme: tal y como se ha diseñado la red en forma de árbol, los elementos de control de la red tienen una visibilidad fronteriza organizada en niveles. Cada elemento de control sólo tendrá información de lo que ocurra en su nivel y en el nivel inmediatamente inferior. Los cambios en una zona de control no afectan a cambios en otras zonas, distribuyendo por lo tanto la información por los niveles del árbol. Es una arquitectura uniforme ya que cualquier elemento que la compone es susceptible de asumir el rol de ser elemento de control.

Una consecuencia de este tipo de arquitectura es una reducción considerable del número de mensajes de control y un mejor aprovechamiento del ancho de banda de la red para la distribución de los contenidos, ya que en otro tipo de arquitecturas totalmente centralizadas hay que mantener comunicación con todos los nodos que forman la estructura para informar de cambios. Sin embargo, en este caso, al tratarse la información de control por niveles, los mensajes que hay que enviar cuando se produce un cambio sólo se producen entre niveles adyacentes. Además, al producirse los mensajes sólo en el entorno de su nivel, la recopilación de información es más sencilla y rápida, y la cantidad de información que hay que recopilar y almacenar en tablas también es mucho menor que en otras arquitecturas P2P.

Con el fin de introducir algunos conceptos y definiciones que son necesarias para desarrollar la propuesta, a continuación se describe la funcionalidad de algunos elementos importantes de la arquitectura propuesta y que se representan en la Fig. 1:

- Canal. Se refiere al árbol de distribución que se genera para reproducir un contenido determinado. Parte de una raíz y es accesible a través de los elementos que componen la red diseñada.
- Nodo. Un Nodo es un usuario que se quiere conectar a un canal para visualizarlo, y para ello tiene que entrar a formar parte del árbol de distribución de ese canal. Cualquier Nodo, cuando se conecta a un árbol, recibirá el vídeo de un único Nodo, que será su padre, y es susceptible de retransmitir ese vídeo a otros nodos, que serán sus hijos, y cuyo número dependerá de los recursos[2]que tenga disponibles. Además, el Nodo sólo enviará y recibirá vídeo relacionado con ese canal. Es decir, que mientras está en un árbol, sólo utilizará contenido manejado por ese árbol. Cada Nodo tienen un identificador único que lo identifica unívocamente dentro de la red. Un dato importante que deben tener todos los nodos es quién va a ser su siguiente padre, denominado padre adoptivo, y esta información se la comunica siempre el Big-Parent de su nivel superior. Así, cuando se pierda la comunicación de un Nodo con su padre actual (que es el que le está enviando el vídeo), el Nodo podrá reconectarse lo antes posible a otro padre.
- Big-Parent. Dentro de cada nivel i existe un único nodo denominado Big-Parent que contiene almacenada información sobre el resto de los nodos de ese nivel i, así como de sus recursos. El Big-Parent constituye el punto de entrada para un Nodo que se quiera conectar a un nodo del nivel i, ya que es el Big-Parent el que le asignará un padre de entre los nodos de su nivel. También se encarga de asignar el *padre adoptivo* a todos los nodos del nivel inferior y de decidir cuál va a ser el Big-Parent de entre ellos. El Big-Parent es un Nodo normal del árbol que recibe y envía vídeo como cualquier otro nodo, sólo que además tiene un papel organizativo dentro de la estructura. Cualquier nodo es candidato a ser Big-Parent cuando entra al árbol y además, cuando un Nodo pasa a ser Big-Parent, no abandona ese rol hasta que abandona el canal. Si abandonase el canal, sería necesario nombrar a otro Nodo para la administración del nivel.
- Generador. Es el nodo fuente y emisor del vídeo. Este nodo no puede desaparecer, ya que constituye el origen del vídeo y es la raíz del árbol para ese canal. A efectos de administración del canal, el comportamiento de este nodo es igual que el de un Big-Parent.
- Bootstrap. Es el nodo en el que están registrados los canales. Cada canal cuenta con un conjunto de nodos Big-Parents, con los que iniciar la comunicación y que se guardan en el Bootstrap de forma dinámica. Cualquier nodo que quiera conectarse a un canal debe comunicarse

[2]En este caso los recursos estarán en función del ancho de banda del nodo, aunque para la realización de las pruebas este parámetro no se ha tenido en cuenta.

antes con el Bootstrap. La manera en que el Bootstrap indica a un nodo con quién debe conectarse se explicará más adelante.

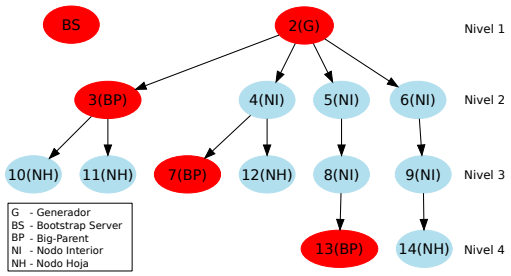

Figura 1. Tipos de Nodos

El concepto de padre adoptivo será también muy importante ya que es el que nos va a permitir reducir los tiempos de reconexión de los nodos. Esto hará que la bajada del buffer de reproducción de cada nodo sea lo menor posible y no se necesiten grandes esfuerzos para recuperarse de las reconexiones (mediante incrementos puntuales del ancho de banda, por ejemplo).

III-A. Estructura de los árboles

La estructura de la red (Fig. 1) está formada por un nodo Generador que es el origen del vídeo y raíz del árbol, y el resto de nodos se organizan por niveles. Cada Nodo puede tener un sólo padre y cero, uno o varios hijos, dependiendo del nivel, del tiempo que lleve conectado y de los recursos disponibles en dicho Nodo. La dimensión vertical del árbol está organizada en niveles, entendiendo que el nivel 2 estará formado por los nodos que tienen como padre directamente al nodo Generador, y el siguiente nivel será el formado por los nodos que tengan como padres a los nodos del nivel anterior y así sucesivamente. Cada nivel tendrá su propio nodo Big-Parent, que se se encargará de administrar los recursos de los nodos de su nivel y también de asignar siguiente padre y Big-Parent entre los nodos del nivel inferior. A nivel de nomenclatura, el nodo Generador será el nivel 1, y los siguientes niveles serán 2, 3, 4, etc. a medida que vayan apareciendo en el árbol. Se dirá que el nivel superior a un nodo que está en el nivel i será el nivel de su padre $(i-1)$, y el nivel inferior el de sus hijos $(i+1)$.

III-B. Mecanismo de conexión

Para conectarse a un canal, el nuevo Nodo envía al Bootstrap server un mensaje con el identificador del canal al cual quiere conectarse (mensaje 1 en Fig. 2). El Bootstrap tiene que seleccionar un Big-Parent del árbol de ese canal y comunicárselo al nuevo Nodo (mensaje 2), indicándole la dirección IP y el puerto del mismo. Cuando el nuevo Nodo recibe la respuesta del Bootstrap, envía un mensaje de *join request* (mensaje 3) al Big-Parent asignado, para indicarle que quiere conectarse. El Big-Parent solicitado puede hacer tres cosas: asignarle un nodo de su mismo nivel (si es que hay recursos) decirle al nodo que se conecte a él mismo si tiene recursos o reenviar la petición a otro Big-Parent de otro nivel. Si el Big-Parent ha decidido que se conecte a él mismo, le añade a su lista de hijos y de nodos del nivel inferior. Si no es así, la selección del padre se hace escogiendo el nodo de su propio nivel que menos recursos ocupados tenga, es decir, el que menos nodos hijos tenga. En el caso en el que el Big-Parent detecte que no hay recursos disponibles en su nivel, devolverá al nodo el identificador de otro Big-Parent[3]. En el caso de tener recursos en su nivel, en el mensaje *join reply* que le devuelve al Nodo, el Big-Parent le especifica cuál va a ser su siguiente padre, su nivel i y también el Big-Parent del nivel i. Una vez que el Nodo se ha conectado en el nivel i, el Big-Parent del nivel superior $(i-1)$ puede convertir al nuevo Nodo en Big-Parent de su nivel (i), si éste es el primer nodo de dicho nivel. Cuando el Nodo recibe la respuesta del Big-Parent (mensaje 4), si su padre va a ser el propio Big-Parent (nivel $i-1$), el nodo se considera conectado y manda un mensaje al Big-Parent de su nivel i para informarle de su existencia y de sus recursos. Con esta información, el Big-Parent de nivel i añadirá al nuevo Nodo a su tabla de nodos en el nivel i.

En el caso de que la conexión se haga con un nodo normal, el Nodo *padre* que recibe este mensaje (5) de *join* acepta la conexión del nuevo Nodo *hijo* y lo añade a su lista de hijos. Además, el nodo *padre* envía un mensaje (7) al Big-Parent de su nivel para decirle que tiene un hijo más (un recurso menos), por lo que el Big-Parent lo añade en su tabla de nodos del nivel inferior $(i+1)$. Finalmente, el nuevo nodo envía un mensaje (8) al Big-Parent de su nivel para que lo añada a su tabla.

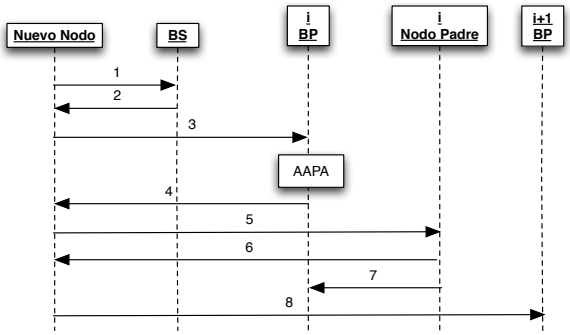

Figura 2. Mecanismo de Conexión a un Nodo normal

III-C. Eliminación de Nodos de un Árbol

Un Nodo se elimina del árbol cuando el usuario abandona el canal o el sistema (mensajes 1 a 5 de la Fig. 3). Antes de abandonar el árbol, el Nodo avisa a su padre para que no le siga enviando vídeo (mensaje 1) y a sus hijos para que se reconecten a otro padre (mensaje 2) y, si el nodo que abandona es Big-Parent, avisa al Bootstrap para que marque el nivel y no asignarle temporalmente nodos. En todo este proceso, tanto el padre como los hijos mandan mensajes y utilizan mecanismos para hacer saber a sus Big-Parents el abandono del nodo. Estos mecanismos se verán a continuación.

El nodo padre del que abandona el canal envía un mensaje (mensaje 3 de la Fig. 3) al Big-Parent de su nivel para comunicarle que tiene un recurso más. A su vez, este Big-Parent avisa (mensaje 4) al Big-Parent del nivel i que un nodo

[3]Una solución similar sería que el Bootstrap server le devuelva al nodo una lista de Big-Parents, por lo que el propio nodo podría elegir otro Big-Parent en caso de que el anterior no devuelva un nodo al que conectarse.

de su nivel ha abandonado el canal[4] por lo que éste ejecuta el algoritmo de asignación de padres adoptivos (AAPA), ya que pudo haber asignado como padre adoptivo al nodo que ha abandonado el canal. Si fuese así, asignaría nuevos padres adoptivos a los nodos correspondientes (mensaje 5).

III-D. Mecanismo de reconexión

Al recibir un mensaje *leave* de su Nodo padre, los *hijos huérfanos* de dicho nodo deberán reconectarse enviando un mensaje (mensaje 6 de la Fig. 3) *reconnect* al nodo que cada uno de ellos tienen asignado como *padre adoptivo*. En el mensaje de *reconnect* un nodo pide reconectarse indicando además a qué nodo estaban conectados, que será el nodo que ha abandonado el canal.

El nodo que recibe el *reconnect* añade a su nuevo hijo a la tabla e informa (mensaje 8) al Big-Parent de su nivel que un nodo de su nivel se ha ido (el que le ha dicho su nuevo hijo) y también los datos de su nuevo hijo. Con esta información, el Big-Parent de nivel i ejecutará el algoritmo de asignación de padres adoptivos y enviará dicha información a los nodos correspondientes (mensaje 9).

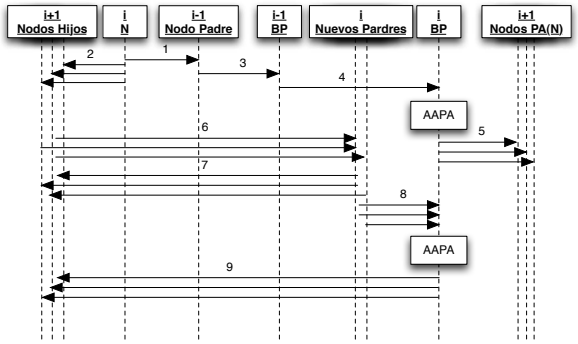

Figura 3. Abandono de Nodos normales y reconexión

III-E. Mecanismo de elección de Big-Parent

El Big-Parent de un nivel i lo elige el Big-Parent del nivel superior $i-1$. El Big-Parent tiene una tabla en la que, aparte de aparecer el identificador de cada nodo, el tiempo que llevan conectados y el padre y siguiente padre que tienen asignados, también aparece el Big-Parent que conoce cada nodo y por supuesto si un nodo es Big-Parent. Para elegir un nuevo Big-Parent en el nivel $i+1$, el Big-Parent de nivel i busca de entre los nodos de nivel inferior aquél que lleve más tiempo conectado, ya que ese Nodo es el que tiene más probabilidad de seguir en el canal ([9]), disminuyendo, en media, el número de mensajes necesarios para la elección y posterior notificación de los Big-Parents.

Con esa información, el Big-Parent de nivel i envía un mensaje a todos los nodos del nivel inferior indicándoles cuál ha sido el Nodo elegido para ser Big-Parent del nivel $i+1$. Cuando cada nodo del nivel inferior reciba el mensaje deberán enviar un mensaje al nuevo Big-Parent con sus tablas de hijos, para que éste conozca toda la información de su nivel y del nivel inferior, incluido el Big-Parent del siguiente nivel. El nuevo Big-Parent también debe enviar un mensaje al Bootstrap para sustituir al que ya había en su nivel e indicarle que ya puede aceptar peticiones de conexión.

[4]Con esto se puede reaccionar ante salidas desordenadas de los nodos. Aunque no se han contemplado las salidas desordenadas en las pruebas, los mecanismos se han diseñado pensando también en ello

III-F. Mecanismo de asignación de Big-Parent en el Bootstrap

Cuando un Nodo se conecta al Bootstrap para obtener un punto de conexión al árbol de un canal, el Bootstrap busca un Big-Parent de forma pseudo-aleatoria de entre los niveles que estén disponibles (que serán todos aquellos cuyo Big-Parent no se haya ido del canal y el nivel se esté reconfigurando todavía), y se lo asignará al nuevo Nodo. Decimos que el algoritmo de búsqueda de Big-Parent es pseudo-aleatoria porque se tiene en cuenta cuántos nodos se han conectado desde el último Big-Parent asignado, no utilizando este último hasta que el número de nodos nuevos haya sobrepasado un cierto límite. Este mecanismo permite controlar la altura de los árboles, ya que se elimina la posibilidad de que nodos consecutivos que entren al canal se aniden formando una rama, sin más nodos en sus niveles correspondientes.

III-G. Mecanismo de abandono de canal y reestructuración del árbol

Cuando un Nodo abandona el canal, lo único que tiene que hacer es enviar un mensaje *leave* a su padre y a cada uno de sus hijos. Además, si el nodo es Big-Parent, envía un mensaje al Bootstrap, para que éste sepa que provisionalmente no hay Big-Parent en el nivel y no asigne nodos ahí hasta que se haya reestructurado. A partir de ese momento, el nodo es libre de abandonar ese canal.

El problema puede surgir, evidentemente, en los nodos que se quedan. Los hijos, al recibir el mensaje *leave*, se reconectan a otro padre siguiendo el mecanismo de reconexión descrito anteriormente.

Cuando el padre recibe el *leave*, envía un mensaje al Big-Parent de su nivel indicándole el nodo que se ha ido. El Big-Parent, de esta manera sabe que el nodo tiene un hijo menos, y además si el que se ha ido fuera Big-Parent, tendría que buscar un nuevo Big-Parent y enviar mensajes según el mecanismo de elección de Big-Parent visto anteriormente. Si el que se ha ido no es Big-Parent, envía un mensaje al Big-Parent del nivel inferior para que éste actualice la tabla de nodos en su nivel.

IV. VALIDACIÓN

Para realizar la validación de la arquitectura diseñada se ha implementado un simulador basado en eventos que es capaz de simular una red con los mensajes que se envían entre los nodos y permite estudiar el comportamiento de éstos frente a cambios que se produzcan en el árbol, en especial las conexiones y reconexiones que se produzcan, y que son las que van a centrar las mediciones realizadas.

Gracias a la implementación de este simulador se podrán obtener datos de árboles de un tamaño considerable y que se podrán procesar más fácilmente. Por ello, al simulador se le ha dotado de un sistema de monitorización que permite obtener casi cualquier dato del estado del árbol o de los nodos.

Para que este simulador sea válido, debe reproducir lo más fielmente posible las condiciones de una red real y el comportamiento de los usuarios para los que se ha diseñado

la arquitectura. Para conseguir las condiciones de una red real, los mensajes que envíen los nodos deben estar sujetos a retardos parecidos a los que se encontrarían en Internet. Es decir, que para cada par de nodos debería existir un retardo diferente que estará en función de la localización de cada nodo en la red. Evidentemente, la red más real que tenemos es Internet, y para intentar reproducir las condiciones completas se necesitaría una simulación muy compleja. Por lo tanto se ha optado por hacer una aproximación basada en datos empíricos obtenidos a través del proyecto PingER[5], el cual realiza estadísticas de retardos entre diferentes puntos de Internet, y que permite elegir varios parámetros para sacar las medidas. En concreto, nosotros nos hemos basado en la tabla resumen del RTT medio para nodos localizados en Europa, y para un tamaño de paquete de 100 bytes. Hemos decidido utilizar datos de un sólo continente porque la probabilidad de que los usuarios que estén viendo una retransmisión en directo sean de una zona concreta es bastante alta. En base a estos datos se ha generado una variable aleatoria normal con media 19.8ms y desviación típica de 16ms para cada enlace entre cada par de nodos. La elección del tipo de variable aleatoria se ha hecho por aproximación a partir de histograma realizado con los datos extraídos del mes de marzo de 2010 del citado proyecto.

Otro dato de entrada importante es el tiempo de estancia de cada usuario en un canal cualquiera. Este dato lo hemos extraído de [10] en donde se obtuvieron diferentes parámetros de un servicio desplegado de IPTV. Aunque el sistema analizado en dicho artículo está basado en una infraestructura dedicada para IPTV desplegada utilizando multicast IP, nosotros tomamos el valor de tiempo de estancia en el canal como referencia, ya que nuestro objetivo es brindar la misma experiencia a los usuarios del sistema P2P propuesto que la que perciben los usuarios de un sistema dedicado. Ya que la información proporcionada en [10] es agregada entre todos los canales ofertados, hemos decidido simplificar el análisis y aplicar la misma distribución de tiempo de estancia a todos los canales simulados. Por este mismo motivo, todos los canales y usuarios tendrán el mismo comportamiento, por lo que los usuarios tendrán un tiempo de estancia en el canal siguiendo la misma distribución y, una vez abandonen su canal actual, elegirán con la misma probabilidad (siguiendo una distribución uniforme) otro canal.

Con todos estos datos que intentan aproximar una red con condiciones reales, se ha creado una simulación en la que existe un Bootstrap server en el que están registrados 4 canales, y que cuenta con un número total de 400 usuarios en el sistema (ya que se utiliza una distribución uniforme para seleccionar el siguiente canal, se espera un número medio de 100 usuarios por canal).

Aunque existen otras condiciones y variables importantes como son el número de recursos (ancho de banda disponible en uplink) y tamaño de buffers en cada equipo, en estas simulaciones se supone que esto no es problema, asumiendo un número infinito de recursos.

[5]http://www-iepm.slac.stanford.edu/pinger/

IV-A. Estructura de los árboles

Una parte importante de la validación es comprobar que los árboles de distribución se crean y reconstruyen de manera adecuada. Para ello hemos obtenido los parámetros típicos de un árbol de distribución para comprobar la morfología media resultante. En la Fig. 4 se representan los valores medios y desviación típica del porcentaje de nodos por cada uno de los niveles. Un resultado interesante es que, aún teniendo número ilimitado de recursos en cada nodo, los nodos padre no tienen muchos nodos hijos conectados ya que la relación entre nodos de cada nivel no es muy grande. Además, aunque el número de nodos por nivel no es constante, sí se puede observar que se distribuye entre los diferentes niveles. Lógicamente, por el orden de conexión, el porcentaje de nodos de los niveles inferiores comienza a decrecer a partir de un cierto punto.

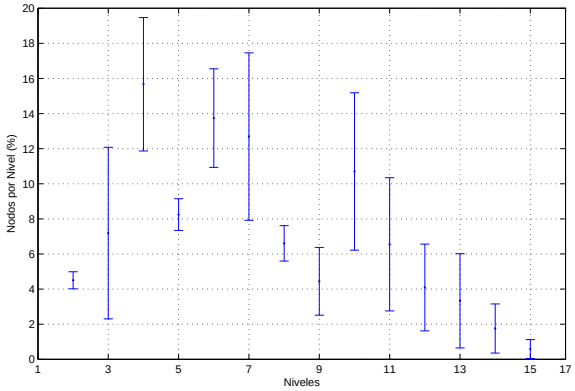

Figura 4. Porcentaje de nodos por nivel

Otro parámetro interesante es el del número de nodos hoja en el árbol. Tras realizar varias simulaciones, hemos obtenido que el número medio de nodos hoja es de $33{,}5\,\%$ del total de nodos y una desviación típica de $1{,}75$. Además, hemos estimado que el número medio de niveles es de $13{,}1$ con una desviación típica de $1{,}25$. Estos valores nos demuestran que el árbol se encuentra acotado tanto en altura como en el número de nodos hijo de cada nodo, y además concuerdan con lo esperado en el diseño descrito en III-F.

IV-B. Tiempos de conexión y reconexión

Uno de los valores más importantes, y motivo por el cual hemos realizado esta propuesta, es el tiempo que necesita un nodo para reconectarse una vez detecta que su padre actual se ha desconectado. Para estimar dicho valor, se han realizado 30 ejecuciones diferentes de 1 día de duración[6] cada una de ellas, obteniendo los resultados que se muestran en la Fig. 5. En esa figura se puede ver la función de distribución (Cumulative Distribution Function o CDF en inglés) para los tiempos de conexión (tiempo necesario para que un nodo se conecte o cambie a otro canal) y de reconexión (tiempo necesario para que un nodo hijo se reconecte a otro nodo padre). De estos resultados, se desprenden las siguientes conclusiones:

- Para el tiempo de conexión, el valor máximo obtenido es de $234ms$ y el mínimo de $35ms$. El valor medio

[6]Se refiere a 24 horas en tiempo de simulación, no en tiempo real.

obtenido entre todas las realizaciones es de $115ms$ con una desviación típica de $38{,}5ms$.

- Para el tiempo de reconexión, el valor máximo obtenido es de $107ms$ y el mínimo de $10ms$. El valor medio de todas las reconexiones es $48ms$ con una desviación típica de $17{,}3ms$

Lo más importante es que el tiempo medio de reconexión de un nodo es muy pequeño, e incluso el valor máximo es aceptable para que un nodo no disminuya mucho su buffer de recepción durante esta fase de reconexión. Analizando detalladamente los resultados, hemos podido comprobar que no se han producido errores en las reconexiones de los nodos con sus *padres adoptivos*. Esto quiere decir que los nodos que se han quedado huérfanos siempre han podido reconectarse al padre adoptivo asignado.

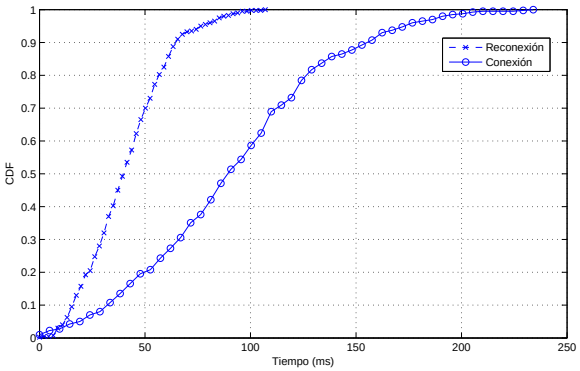

Figura 5. CDF de los tiempos de conexión y reconexión

V. Conclusiones

En este artículo hemos presentado una idea novedosa que pretende mejorar las prestaciones de los árboles multicast a nivel de aplicación. El objetivo principal es el de distribuir la carga de administración entre todos los equipos que forman parte del árbol multicast y, a su vez, minimizar el tiempo de reconexión de un nodo cuando su nodo padre abandona el canal. Mediante simulación, hemos obtenido valores del tiempo de reconexión muy buenos y que harían que el buffer de los clientes disminuyera de manera casi despreciable, por lo que se valida la propuesta. Es también interesante recalcar que se ha comprobado que en los casos peores, los tiempos siguen siendo muy bajos, y esto es debido a que la información de padre adoptivo que tiene cada uno de los nodos es siempre válido, ya que en caso de reconexión, siempre se pueden conectar a dichos nodos padre.

Como se ha comentado en la parte de validación, las simulaciones han sido realizadas suponiendo unas condiciones óptimas en la red, en donde todos los nodos tienen suficiente ancho de banda en subida y el tamaño de los bufferes no representa ningún problema. Este es un aspecto en el que se quiere seguir trabajando y se deja como trabajo futuro la inclusión de restricciones del número de recursos en los diferentes nodos.

Agradecimientos

Este artículo está financiado parcialmente por el proyecto MEDIANET (S-2009/TIC-1468) de la Comunidad de Madrid y por la Cátedra Telefónica en Internet del Futuro para la Productividad de la Universidad Carlos III de Madrid.

Referencias

[1] Y. Chu, S. Rao, S. Seshan, and H. Zhang, "A case for end system multicast," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 8, pp. 1456–1471, 2002.

[2] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable application layer multicast," in *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*. ACM, 2002, p. 217.

[3] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: high-bandwidth multicast in cooperative environments," in *Proceedings of the nineteenth ACM symposium on Operating systems principles*. ACM, 2003, p. 313.

[4] D. Kostić, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: High bandwidth data dissemination using an overlay mesh," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, p. 297, 2003.

[5] A. Nicolosi and S. Annapureddy, "P2PCAST: A peer-to-peer multicast scheme for streaming data," in *1st IRIS Student Workshop (ISW3). Available at: http://www. cs. nyu. edu/nicolosi/P2PCast. ps*. Citeseer, 2003.

[6] B. Li, Y. Qu, Y. Keung, S. Xie, C. Lin, J. Liu, and X. Zhang, "Inside the new coolstreaming: Principles, measurements and performance implications," in *Proc. of IEEE Infocom*. Citeseer, 2008.

[7] S. Xie, B. Li, G. Keung, and X. Zhang, "Coolstreaming: Design, Theory and Practice," *IEEE Transactions on Multimedia*, vol. 9, no. 8, p. 1661, 2007.

[8] X. Zhang, J. Liu, and B. Li, "On large-scale peer-to-peer live video distribution: Coolstreaming and its preliminary experimental results," in *Proc. MMSP*. Citeseer, 2005.

[9] Z. Liu, C. Wu, B. Li, and S. Zhao, "Distilling superior peers in large-scale P2P streaming systems," *Proc. of IEEE INFOCOM, Rio de Janeiro, Brazil*, 2009.

[10] M. Cha, P. Rodriguez, J. Crowcroft, S. Moon, and X. Amatriain, "Watching television over an IP network," in *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*. ACM, 2008, pp. 71–84.